Technische Universität Darmstadt
Fachbereich Informatik
Knowledge Engineering Group

Diplomarbeit

Paarweises Lernen von Multilabel-Klassifikationen mit dem Perzeptron-Algorithmus

Eneldo Loza Mencía

31. März 2006

Betreuer: Prof. Dr. Johannes Fürnkranz

Ehrenwörtliche Erklärung

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

Darmstadt, 31. März 2006 Eneldo Loza Mencía

Inhaltsverzeichnis

Notation 7

1 Einleitung 8

2 Grundlagen 11
- 2.1 Klassifikation und Klassifizierung . . . 11
- 2.2 Mächtigkeit der Klassifikation . . . 12
- 2.3 Klassifizieren und Lernen . . . 14
  - 2.3.1 Überwachtes Lernen . . . 14
  - 2.3.2 Bewertung von Klassifizierern . . . 15
  - 2.3.3 Inkrementelles und Batch-Lernen . . . 17
  - 2.3.4 Konservatives Lernen . . . 18
- 2.4 Multilabel-Klassifikation . . . 18
  - 2.4.1 Lernen von Multilabel-Klassifizierern . . . 18
  - 2.4.2 Bewertung von Rankings . . . 20
- 2.5 Textklassifizierung . . . 23
  - 2.5.1 Vorverarbeitung . . . 23
  - 2.5.2 Parsen . . . 24
  - 2.5.3 Vektorraummodell . . . 24
    - 2.5.3.1 Verlust der Wortinterdependenzen . . . 25
    - 2.5.3.2 Wortgewichtung . . . 25
  - 2.5.4 Stoppwörter . . . 26
  - 2.5.5 Grundformenreduktion . . . 27
  - 2.5.6 Feature-Selection . . . 27

3 Der Perzeptron-Algorithmus 30
- 3.1 Das Neuron als Vorbild . . . 30
- 3.2 Künstliches Neuron . . . 30
- 3.3 Beschreibung des Klassifizierers . . . 32
- 3.4 Die Perzeptron-Lernregel . . . 35
- 3.5 Konvergenz und Fehlerabschätzung . . . 38
- 3.6 Lineare Separierbarkeit von Datenmengen . . . 40
- 3.7 Fehlerabschätzung bei nicht separierbaren Daten . . . 42
- 3.8 Varianten . . . 42
  - 3.8.1 Perzeptron ohne Schwellwert . . . 43
  - 3.8.2 Andere Varianten . . . 43
- 3.9 Komplexitätsanalyse . . . 45
- 3.10 Einsatz bei der Textklassifizierung . . . 46

4 Multiclass Multilabel Perceptron **48**
4.1 One-against-all mit Perzeptrons . . . 48
4.2 Beschreibung des MMP . . . 50
4.2.1 Haupteigenschaften . . . 51
4.2.2 Aktualisierungsmethoden . . . 52
4.2.3 Loss-Funktionen . . . 54
4.3 Fehlerabschätzung . . . 55
4.4 Komplexitätsanalyse . . . 56
4.5 Einsatz bei der Textklassifizierung . . . 57

5 Multilabel-Pairwise-Coupling **60**
5.1 Beschreibung der Funktionsweise . . . 60
5.2 Vergleich zum One-against-all-Verfahren . . . 62
5.3 Motivation für den MLPC . . . 63
5.4 Pairwise-Coupling bei linearen Problemen . . . 64
5.5 Gültigkeit für den MMP-Algorithmus . . . 65
5.6 Analyse des Votings . . . 66
5.7 Komplexitätsanalyse . . . 68
5.8 Varianten und alternative Ansätze . . . 71

6 Versuchsergebnisse **74**
6.1 Reuters-Datenbank . . . 74
6.1.1 Klassifikation . . . 74
6.1.2 Datenvorverarbeitung . . . 75
6.2 Implementierung . . . 76
6.3 Versuchsvorbereitung . . . 78
6.3.1 Aufteilung in Trainings- und Testmenge . . . 79
6.3.2 Anzahl der Attribute . . . 79
6.3.3 Wortgewichtung . . . 80
6.3.4 Feature-Selection-Methode . . . 81
6.3.5 Einstellungen des MMP-Algorithmus . . . 83
6.3.6 Einstellungen des MLPC_p . . . 84
6.3.7 Grundeinstellungen . . . 85
6.4 Untersuchungen zum MMP und MLPC_p . . . 85
6.4.1 Direkter Vergleich . . . 85
6.4.2 Laufzeit und Speicherverbrauch . . . 88
6.4.3 Lernkurve . . . 88
6.4.4 Konvergenz und Overfitting . . . 89
6.5 Zusätzliche Untersuchungen . . . 91
6.5.1 PKPD . . . 91
6.5.2 Ausnutzung der Hierarchical Policy . . . 91

7 Zusammenfassung und Ausblick **94**

Literaturverzeichnis **95**

Abbildungsverzeichnis

2.1 Bewertung von Rankings 20

3.1 Grundtypen der Aktivierungsfunktion eines künstlichen Neurons 31
3.2 Lineare Separierbarkeit von vier Punkten im zweidimensionalen Raum 34
3.3 Perzeptron-Lernalgorithmus 36
3.4 Trennende Hyperebenen im zweidimensionalen Raum 39

4.1 One-against-all am Beispiel des Perzeptron 49
4.2 MMP-Algorithmus 51
4.3 MMP-Aktualisierung am Beispiel eines fehlerhaften Rankings 53

5.1 Training des Multilabel-Pairwise-Coupling-Algorithmus 61
5.2 Gegenüberstellung der One-against-all- und Pairwise-Coupling-Methode im zweidimensionalen Raum 65

6.1 Verteilung der Anzahl der relevanten Klassen 75
6.2 Größe der Kategorien 75
6.3 Nachricht im XML-Format: `477551newsML.xml`. Der extrahierte Text und die Topiccodes der Nachricht sind kursiv dargestellt. 77
6.4 Token-File 77
6.5 Feature-Selection 80
6.6 Fehler in Abhängigkeit der Anzahl der Labels eines Beispiels 87
6.7 Lernkurven 90
6.8 Trainings- und Testfehler abhängig von der Anzahl der Epochen 90
6.9 Komplette Übersicht der Feature-Selection für den MMP-Algorithmus 93

Tabellenverzeichnis

5.1 Vergleich der Laufzeit- und Speicherkomplexität 70

6.1 Vergleich der Term-Weighting-Methoden 81
6.2 Vergleich der Methoden der Feature-Selection 82
6.3 Vergleich der MMP-Parameter 83
6.4 Vergleich der Varianten des Perzeptron-Algorithmus 84

6.5 Grundeinstellungen für die Versuche 85
6.6 Direkter Vergleich zwischen MMP und MLPC$_p$auf den Testdaten 86
6.7 Vergleich der hergeleiteten Fehler 86
6.8 Vergleich der besten Einstellungen für MMP und MLPC$_p$auf den Testdaten 87
6.9 Direkter Vergleich zwischen MMP und MLPC$_p$auf den Trainingdaten 87
6.10 Vergleich der PKPD-Methode mit dem Voting 91
6.11 Vergleich des MMP- und MLPC$_p$-Algorithmus bei Ausnutzung der Hierarchie. . . 92

Notation

$\mathbb{R}$	Menge der reellen Zahlen
$\mathcal{M}, \mathcal{M}$ *(kalligraphische Schrift)*	Menge
$\lvert\mathcal{M}\rvert$	Mächtigkeit der Menge $\mathcal{M}$
$2^{\mathcal{M}}$	Potenzmenge der Menge $\mathcal{M}$
$\varnothing$	Leere Menge
n (geneigte Minuskel)	eindimensionale Zahl
N (geneigte Majuskel)	Konstante, feste Variable
$\lvert n\rvert$	Betrag der Zahl v
$\bar{\mathrm{v}}$ *(aufrechne Minuskel)*	Vektor
$\lvert\bar{\mathrm{v}}\rvert$	euklidische Norm eines Vektors $\bar{\mathrm{v}}$

Verwendete Symbole:

$\mathcal{X}$	Menge der Instanzen
$\bar{\mathrm{x}}$	Instanz, Beispiel, Dokument $\bar{\mathrm{x}} \in \mathcal{X}$
N	Anzahl der Attribute
$\bar{\mathrm{x}} = (a_1, \ldots, a_N)$	Attributvektor von $\bar{\mathrm{x}}$
$\mathcal{Y}$	Menge der möglichen Klassen
K	Anzahl der möglichen Klassen $K := \lvert\mathcal{Y}\rvert$
$\bar{\mathrm{y}}$	einzelne Klasse $\bar{\mathrm{y}} \in \mathcal{Y}$
$\mathcal{L}$	Untermenge $\mathcal{L} \subseteq \mathcal{Y}$
$\langle \bar{\mathrm{x}}_1, \bar{\mathrm{x}}_2, \ldots, \bar{\mathrm{x}}_\mathrm{i} \rangle$	Folge von Instanzen
P	Anzahl der Trainingsinstanzen
$\mathcal{S}$	Trainingsmenge
$\langle (\bar{\mathrm{x}}_1, \mathcal{L}_1), \ldots, (\bar{\mathrm{x}}_P, \mathcal{L}_P) \rangle$	Trainingsmenge
Q	Anzahl der Testinstanzen
$\mathcal{T}$	Testmenge
$\Pr(A)$	Wahrscheinlichkeit für Ereignis A
$\mathrm{E}[X]$	Erwartungswert der Zufallsvariablen X
$\mathrm{error}_{\mathcal{X}}$	Wirklicher Fehler
$\mathrm{error}_{\mathcal{T}}$	Fehler auf den Testdaten
δ	Loss-Funktion
$\hat{\delta}$	Durchschnittlicher Loss-Wert
$I[x]$	Indikatorfunktion für Aussage x

1 Einleitung

Einen wichtigen Bereich des *Maschinellen Lernens* bilden Lernalgorithmen, die Zuordnungen von Objekten zu Klassen lernen. Ein in diesem Zusammenhang oft untersuchter Fall ist die Zuordnung von Objekten zu genau einer von mehreren Klassen, sogenannte *Multiklassen-Klassifikationen*. Probleme aus dem realen Leben sind jedoch oft so konstruiert, daß ein Objekt nicht nur einer, sondern zugleich mehreren Klassen angehören kann. Ein weitläufiges Beispiel hierfür ist die Klassifikation von Texten, denn ein Text kann beispielsweise zu einer Reihe von Autoren, verschiedenen Gattungen oder unterschiedlichen Themenbereichen zugeordnet sein. Im Gegensatz zur Multiklassen-Klassifikation können bei dieser Art der Einteilung die Klassen beliebig wie Etiketten an ein Objekt geheftet werden, man bezeichnet sie daher als *Multilabel-Klassifikation*. Mit dieser Problemstellung beschäftigt sich die vorliegende Arbeit.

Die Menge der Algorithmen, die eine Multilabel-Klassifikation lernen können, ist sehr beschränkt. Dies resultiert aus der Tatsache, daß die meisten Lerner nur in der Lage sind, binäre Entscheidungsprobleme zu lösen. D. h. die Objekte können nur einer von zwei unterschiedlichen Klassen angehören. Diese Einschränkung kann beispielsweise durch die interne Beschreibungssprache gegeben sein. Es gibt jedoch verschiedene Ansätze, diese Verfahren für den Multiklassen-Fall zu verallgemeinern [Weston und Watkins, 1999, Hsu und Lin, 2002].

Eine Methode, um binäre Algorithmen auch für das Lernen multipler Zuordnungen zu verwenden, stellen die Class-Binarization-Methoden dar. Bei diesen Verfahren wird ein Multiklassen-Problem in mehrere Zweiklassen-Probleme aufgeteilt. Ein klassisches Beispiel hierfür ist die *One-against-all*-Methode, die für jede Klasse einen Klassifizierer trainiert. Der sogenannte Basislerner erhält zum Anlernen des Klassifizierers als positive Beispiele alle Objekte, die seiner Klasse zugeordnet sind und als negative Beispiele alle restlichen Objekte. Bei der anschließenden Klassifizierung eines Beispiels wird die Vorhersage eines Basisklassifizierers als Zuordnung zu einer Klasse interpretiert. Dieses Verfahren ist somit in der Lage, einen einfachen, binären Lernalgorithmus Multilabel-Klassifikationen lernen zu lassen.

Einen solchen einfachen, binären Lernalgorithmus stellt der *Perzeptron*-Algorithmus dar. Er wurde 1958 von Rosenblatt als Modell einer anpassbaren, lernenden neuronalen Zelle entwickelt. Es stellte sich heraus, daß dieser sehr einfache und schnelle Algorithmus imstande ist, ein binäres Problem zu lösen, wenn die Daten *linear trennbar* sind [Minsky und Papert, 1969]. Dies setzt die mögliche Repräsentation der zu untersuchenden Objekte als Punkte in einem Vektorraum voraus, wie es im Maschinellem Lernen üblich ist. In diesem Zusammenhang bedeutet die lineare Separierbarkeit von zwei Punktmengen nun, daß eine Hyperebene existiert, die den Raum in zwei Hälften teilt, in denen sich jeweils nur Punkte aus einer Klasse befinden. Ein intuitives Beispiel hierfür ist eine Gerade im zweidimensionalen Raum, die Punkte aus zwei Klassen voneinander trennen kann. Trotz dieser Einschränkung zeigte sich, daß das Perzeptron einen effizienten und effektiven Algorithmus darstellt [Freund und Schapire, 1999, Ng u. a., 1997].

Inspiriert durch diese Beobachtungen entwickelten Crammer und Singer [2003] das *Multilabel-Multiclass-Perzeptron* (*MMP*). Dieser Algorithmus basiert auf dem beschriebenen One-against-all-Verfahren, so daß auch hier ein Perzeptron eine Klasse repräsentiert. Doch anstatt eine Menge von relevanten Klassen als Vorhersage zurückzugeben, erstellt er eine Rangordnung der Klassen sortiert nach ihrer Relevanz, das sogenannte *Klassenranking*. In diesem Sinne versucht der Algorithmus Perzeptrons zu trainieren, die alle relevanten Klassen an höherer Stelle als die irrelevanten einordnen. Hierfür müssen die Perzeptrons im Unterschied zum One-against-all-Verfahren in Beziehung zueinander gebracht werden. Dieser Ansatz stellt eine klare Verbesserung gegenüber der One-against-all-Methode dar und konnte zudem, trotz der einfachen Funktionsweise, sehr gute Ergebnisse bei der Klassifizierung von Text erzielen. Darüber hinaus ist er durch die Verwendung von Perzeptrons sehr effizient.

In mehreren Untersuchungen wurde gezeigt, daß eine alternative Class-Binarization-Methode, das sogenannte *Pairwise-Coupling*-Verfahren, klare Vorteile für das Lösen von Multiklassen-Problemen gegenüber der klassischen One-against-all-Methode besitzt [Hsu und Lin, 2002, Fürnkranz, 2002]. Bei diesem Verfahren wird für jedes mögliche Paar von Klassen ein Klassifizierer trainiert. Hierfür werden die Beispiele aus den beiden Klassen als positive und negative Beispiele genutzt. Beispiele aus anderen Klassen werden dabei ignoriert. Für die Zuordnung eines Beispiels zu einer Klasse werden die Vorhersagen der einzelnen Basisklassifizierer in Form einer Abstimmung zusammengeführt. Dabei entspricht eine Einzelvorhersage einer Stimme für eine von zwei Klassen. Es gewinnt schließlich die Klasse mit den meisten Stimmen.

Eine Folge der paarweisen Aufteilung ist, daß die so erzeugten Entscheidungsprobleme kleiner sind als bei der One-against-all-Methode. Dabei besitzen kleinere Probleme die günstige Eigenschaft, daß sie einfacher zu lösen sind. Hierbei sei nochmals das Beispiel des zweidimensionalen Raumes erwähnt. Für eine kleinere Menge von Punkten erscheint es intuitiv einfacher, eine trennende Gerade zu finden, da mehr Raum zwischen den Punkten vorhanden ist. Eine weitere günstige Eigenschaft des paarweisen Ansatzes besteht darin, daß trotz der größeren Anzahl an Basislernern durch die Aufteilungen in kleinere Probleme insgesamt nicht mehr Zeit für das Lernen benötigt wird, als bei der One-against-all-Methode.

Die Hauptmotivation dieser Arbeit bestand nun darin, zu untersuchen, ob sich die genannten Vorteile des paarweisen Ansatzes auch auf den allgemeinen Fall des Lernens von Multilabel-Klassifikationen übertragen lassen. Im Rahmen dieser Untersuchung entstand der *Multilabel-Pairwise-Coupling*-Algorithmus (*MLPC*), der den Schwerpunkt dieser Arbeit darstellt. Bei diesem neuartigen Algorithmus wird kurzgefaßt das übliche Pairwise-Coupling-Verfahren für alle relevanten Klassen eines Beispiels durchgeführt. Für die Klassifizierung eines Beispiels wird wie bereits bei der Multiklassen-Variante eine Wahl unter den Basisklassifizierern abgehalten. Die Klassen geordnet nach der Anzahl der Stimmen ergeben das besprochene Klassenranking.

Inspiriert durch die günstigen Auswirkungen auf den MMP-Algorithmus sowohl für die Leistung als auch für die Laufzeit, wurde der Einsatz des Perzeptron-Algorithmus als Basislerner untersucht. Die Überlegungen hierzu bestätigten die Vorteile der paarweisen Aufteilung auch für lineare Verfahren wie dem Perzeptron. Die Variante des Multilabel-Pairwise-Coupling-Algorithmus mit dem Perzeptron als Basislerner wurde $MLPC_p$ genannt.

Dies ließ die zentrale Frage aufkommen, ob man aus der Überlegenheit des paarweisen Ansatzes gegenüber der One-against-all-Methode auch auf einen Vorteil des MLPC_p-Verfahrens zum nahen verwandten MMP-Algorithmus schließen kann.

Dieser Fragestellung wurde sowohl in theoretischer als auch in praktischer Hinsicht nachgegangen. Für die experimentelle Evaluierung wurde die bereits bei den Untersuchungen des MMP-Algorithmus eingesetzte Textsammlung RCV1 verwendet, die von der Reuters-Nachrichtenagentur zu Forschungszwecken zur Verfügung gestellt wurde.[1] Hierbei handelt es sich um ein Datenarchiv bestehend aus Nachrichtenmeldungen, die aufgrund der Komplexität der Klassifikation und der enormen Datenfülle hohe Anforderungen an einen Multilabel-Lernalgorithmus stellt.

In zahlreichen Versuchen konnten die theoretischen Überlegungen bestätigt werden. Insbesondere konnte eine entscheidende Verbesserung der Qualität der Klassenrankings festgestellt werden.

Die vorliegende Arbeit ist folgendermaßen unterteilt: Kapitel 2 bietet eine Einführung in die Problematik der Multilabel-Klassifikation. Unter anderem wird der wichtige Punkt der Evaluierung von Multilabel-Klassifizierern besprochen. Für die am Ende folgende experimentelle Beurteilung der Algorithmen wird ein kurzer Einblick in das Feld der Textklassifizierung gegeben. Kapitel 3 beschäftigt sich mit dem Perzeptron-Algorithmus, der die Grundlage für die anschließend vorgestellten Lernalgorithmen darstellt. In Kapitel 4 wird der MMP-Algorithmus besprochen. Er dient dem MLPC$_p$-Algorithmus insbesondere als Referenzpunkt. Im zentralen 5. Kapitel wird schließlich der MLPC-Algorithmus vorgestellt und ausführlich untersucht. In Kapitel 6 werden die experimentellen Ergebnisse des Vergleichs zwischen MMP- und MLPC$_p$-Algorithmus präsentiert. Kapitel 7 faßt die wichtigsten Ergebnisse zusammen und gibt einen Ausblick auf weitere Möglichkeiten des MLPC-Algorithmus.

[1] http://about.reuters.com/researchandstandards/corpus/

2 Grundlagen

Gegenstand des Maschinellen Lernens sind künstliche Systeme, die im Stande sind, aus gegebenen Informationen zu lernen. Lernen bedeutet hierbei, daß sie in der Lage sind, Gesetzmäßigkeiten oder Muster in den Lerndaten selbständig zu erkennen und anhand dieser gewonnenen Erkenntnis die Daten zu verallgemeinern. Das System erstellt intern eine Beschreibung der Informationen, diese muß nicht menschenverständlich sein. Man nennt dieses vom System generierte Wissen *Modell*, *Konzept* oder *Hypothese*. Da das System lernt und es sich im Allgemeinen als Algorithmus beschreiben läßt, nennt man es *Lerner* oder *Lernalgorithmus*.

Ein mögliches Szenario wäre, daß der Lerner als Informationen Beispiele von (beliebigen) Objekten erhält mit dem Ziel, die Zuordnung von Objekten zu bestimmten Eigenschaften zu erkennen. In dieser Arbeit beschränken wir uns auf den Fall, daß es sich bei den zu erlernenden Eigenschaften um Klassen handelt, also auf das Lernen von Klassifikationen.

2.1 Klassifikation und Klassifizierung

Die *Klassifizierung* beschreibt den Vorgang der Einteilung von *Objekten* in *Klassen*. Im normalen Sprachgebrauch wird zwischen Klassifizierung und *Klassifikation* nicht unterschieden, in dieser Arbeit hingegen werden die beiden Begriffe nicht synonym verwendet. Während die Klassifizierung die Tätigkeit des Klassifizierens, des Zuordnens, selbst ist, beschreibt die Klassifikation das Ergebnis eines solchen Prozesses. Die Klassifikation eines Objektes oder einer Menge von Objekten ist ausschließlich die Information über die Zugehörigkeit der Objekte zu Klassen, es sagt nichts über die Art und Weise aus, wie man zu dieser Einteilung kommt oder gekommen ist.

Eine Klassifikation ist demnach definiert als die Relation (zwischen Objekten und Klassen) der Zugehörigkeit eines Objekt zu einer Klasse. Formal läßt sich eine Klassifikation durch die Funktion f darstellen.

Sei $\mathcal{X}$ die Menge aller möglichen Objekte, $\bar{x}$ ein Objekt aus dieser Menge, $\mathcal{Y}$ eine vorher definierte Menge von möglichen Klassen und $\bar{y}$ eine Klasse daraus, so beschreibt die boolesche Funktion

$$\begin{aligned} &f : \mathcal{X} \times \mathcal{Y} \to \{wahr, falsch\} \\ &f(\bar{x},\bar{y}) := \begin{cases} wahr & \text{falls } \bar{x} \text{ zu } \bar{y} \text{ zugeordnet ist} \\ falsch & \text{sonst} \end{cases} \quad , \bar{x} \in \mathcal{X}, \bar{y} \in \mathcal{Y} \end{aligned} \tag{2.1.1}$$

jede mögliche Zuordnung eines Objekts zu einer Klasse. Eine Klassifikation wäre somit die Relation $\{(\bar{x},\bar{y}) \mid f(\bar{x},\bar{y}) = wahr\}$ definiert auf $\mathcal{X} \times \mathcal{Y}$. Die Menge der Klassen, in denen ein Objekt $\bar{x}$ eingeordnet ist, läßt sich folgendermaßen definieren:

$$\mathcal{L} := \{y_i \mid f(\bar{x}, y_i) = wahr, y_i \in \mathcal{Y}\} \in 2^{\mathcal{Y}} \qquad , \mathcal{L} \subseteq \mathcal{Y}, \bar{x} \in \mathcal{X} \tag{2.1.2}$$

oder kurz:

$$\mathcal{L} = f(\bar{x}) := \{y_i \mid f(\bar{x}, y_i) = wahr, y_i \in \mathcal{Y}\} \tag{2.1.3}$$

Umgekehrt gibt

$$f^{-1}(\bar{y}) := \{\bar{x}_i \mid f(\bar{x}_i, \bar{y}) = wahr, \bar{x}_i \in \mathcal{X}\} \subseteq \mathcal{X} \qquad , \bar{y} \in \mathcal{Y} \tag{2.1.4}$$

die Menge der Objekte an, die der Klasse $\bar{y}$ angehören.

2.2 Mächtigkeit der Klassifikation

Die wichtigste Eigenschaft einer Klassifikation ist einerseits die Anzahl der möglichen Klassen und andererseits die Anzahl der Klassen, der ein Objekt angehören kann. Bei der Anzahl der möglichen Klassen gibt es zwei Unterscheidungen, die *Zweiklassen-Klassifikation* für genau zwei Klassen und die *Multiklassen-Klassifikation* für beliebig mehr als zwei Klassen. Muss ein Objekt in (genau) eine Klasse eingeordnet werden, handelt es sich um eine Klassifikation mit *ein*facher Zuordnung[1], kurz *einfache Klassifikation*. Darf ein Objekt mehreren Klassen angehören, handelt es sich um eine Klassifikation mit mehrfacher Zuordnung[2], kurz *mehrfache Klassifikation*.

Es ergeben sich vier mögliche Kombinationen für die Beschreibung der *Mächtigkeit einer Klassifikation*:

Zweiklassen-Klassifikation mit einfacher Zuordnung Die einfachste Klassifikation stellt die *einfache Zwei-Klassen-Klassifikation* dar. Es gibt nur zwei mögliche Klassen und im Normalfall bedeutet die Zuordnung eines Objekts zu der einen Klasse das Vorhandensein einer Eigenschaft (*Positiv-* oder *Wahr*-Klasse) und die Zuordnung zu der anderen Klasse die Verneinung (*Negativ-* oder *Falsch*-Klasse) der Eigenschaft. Ein Objekt muss (genau) einer der beiden Klassen zugeordnet sein. Ein Beispiel ist die Kennzeichnung einer Email als Spam oder Nicht-Spam. Im Allgemeinen ist bei Verwendung der Begriffe Zweiklassen-Klassifikation oder *binäre Klassifikation*[3] die einfache Zweiklassen-Klassifikation gemeint.
Für f, $\mathcal{Y}$ bzw. der Anzahl der Klassen K, und $\mathcal{L}$ gelten folgende Bedingungen:

$$\begin{aligned} K_{bin} &= |\mathcal{Y}| = 2 && \text{(Zweiklassen)} \\ \mathcal{L}_{bin} &= f_{bin}(\bar{x}) \in \{\{y_1\}, \{y_2\} \mid y_1, y_2 \in \mathcal{Y}\} \\ |\mathcal{L}_{bin}| &= 1 && \text{(einfache Zuordnung)} \end{aligned} \tag{2.2.1}$$

[1] engl.: single-label classification
[2] engl.: multi-label classification
[3] engl.: binary classification, dichotomous classification

Zweiklassen-Klassifikation mit mehrfacher Zuordnung Die Zweiklassen-Klassifikation mit mehrfacher Zuordnung ist eine Klassifikation mit zwei möglichen Klassen und der möglichen Zuordnung zu der einen, zu der anderen Klasse, zu keiner oder zu beiden gleichzeitig. Diese Klassifikation kommt in der Praxis fast nicht vor. Bei der Einteilung in zwei Klassen schließen sich beide Klassen meist gegenseitig aus, und deshalb ist in der Regel eine mehrfache Zuordnung nicht erwünscht.[4]

Multiklassen-Klassifikation mit einfacher Zuordnung Bei der *einfachen Multiklassen-Klassifikation* ist eine Einteilung in mehr als zwei Klassen möglich. Dabei schließen sich die verschiedenen Klassen aus, es gibt keine Überschneidungen zwischen den Klassen, d. h. ein Objekt kann nicht gleichzeitig mehr als einer Klasse angehören. Das ist die geläufigste Klassifikation, die im Maschinellen Lernen betrachtet wird. Wenn von Multiklassen-Klassifikation[5] die Rede ist, ist in der Regel die Multiklassen-Klassifikation mit einfacher Zuordnung gemeint.

Beispiel: Bei der optischen Zeichenerkennung wird ein Zeichen, dargestellt als Folge von Bildpunkten, als ein Buchstabe erkannt. Nimmt man die Folge von Bildpunkten als Objekt und die möglichen Buchstaben als Klassen (jeder Buchstabe entspricht einer Klasse), handelt es sich hierbei klar um das Problem einer Multiklassen-Klassifikation. Man beachte hierbei auch, daß ein Zeichen nicht gleichzeitig als mehr als ein Buchstabe erkannt werden kann, ein Zeichen kann nicht gleichzeitig *„A“ und „B“* sein.

Es gelten folgende Bedingungen:

$$\begin{aligned} K_{mc} &> 2 && \text{(Multiklassen)} \\ \mathcal{L}_{mc} &= f_{mc}(\bar{\mathrm{x}}) \in \{\{y_i\} \mid y_i \in \mathcal{Y}\} \\ |\mathcal{L}_{bin}| &= 1 && \text{(einfache Zuordnung)} \end{aligned} \tag{2.2.2}$$

Multiklassen-Klassifikation mit mehrfacher Zuordnung Im Gegensatz zur einfachen Multiklassen-Klassifikation kann ein Objekt in einer *mehrfachen Multiklassen-Klassifikation* durchaus gleichzeitig in mehr als einer Klasse auftauchen. Jedes Objekt gehört einer oder mehr als einer Klasse an, oder sogar keiner oder allen Klassen[6]. So kann eine Erzählung gleichzeitig ein Drama als auch eine Komödie sein. Beide Genres können sich überschneiden. Die Klassen, die einem Objekt zugeordnet sind, nennt man in diesem Fall auch *Labels*, da die Klassen wie Etiketten beliebig an das Objekt geheftet werden können. Da die mehrfache Zuordnung im Allgemeinen nur im Multiklassen-Fall Anwendung findet, wird abkürzend für die mehrfache Multiklassen-Klassifikation der englische Begriff für mehrfache Zuordnung *Multilabel-Klassifikation* gebraucht.
Es gilt:

$$\begin{aligned} K_{bin} &> 2 && \text{(Multiklassen)} \\ \mathcal{L}_{ml} &= f_{ml}(\bar{\mathrm{x}}) \in 2^{\mathcal{Y}} \\ 0 \leq |\mathcal{L}_{ml}| &\leq K_{bin} && \text{(mehrfache Zuordnung)} \end{aligned} \tag{2.2.3}$$

[4] Die mehrfache Zweiklassen-Klassifikation entspricht in der Logik der Definition der pseudodichotomen im Gegensatz zur dichotomen Einteilung eines Begriffs (wikipedia: *„Pseudodichotomische Einteilung eines Begriffsumfangs“*, wikipedia: *„Dichotomie“*)

[5] engl.: multi-class-classification, polytomous classification

[6] die letzten beiden Fälle kann man auffangen, indem man neue Klassen *„Keine Zuordnung“* bzw. *„alle Klassen“* einfügt

2.3 Klassifizieren und Lernen

Im Maschinellem Lernen wird ein Objekt üblicherweise durch eine geordnete Zusammenstellung seiner Eigenschaften in Form eines *n-Tupels* repräsentiert. Um sich auf die Eigenschaften eines Objekts zu beziehen, verwendet man im Allgemeinen die Begriffe *Attribute* oder *Features*, statt n-Tupel findet auch die Bezeichnung *Vektor* Gebrauch. Diese Form der Repräsentation von Objekten in einem hochdimensionalen Vektorraum nennt sich *Vektorraummodell*.

Möchte man nun Objekte eines Typs untersuchen, d. h. lernen, so legt man vorher fest, welche Eigenschaften für die Analyse verwendet werden sollen. Die Form und Größe der Tupel, die die Objekte beschreiben, ist anschließend für alle Objekte festgelegt (für diesen einen Lernvorgang). Die Menge der Objekte bildet die Datenbasis für die Analyse. Sind die Objekte in Form von Tupel repräsentiert, so spricht man von *Instanzen* oder *Beispielen*. Eine Instanz ist somit definiert als

$$\begin{aligned} &\bar{x} \in \mathcal{X} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \\ &\bar{x} = (a_1, \ldots, a_N),\ a_j \in \mathcal{A}_j \end{aligned} \tag{2.3.1}$$

$\mathcal{X}$ ist die Menge aller möglichen Instanzen, N die Anzahl der Attribute, $\mathcal{A}_i$ sind die einzelnen Attribute und a_i die Ausprägung in der Instanz $\bar{x}$.

Da wir den Fall der Klassifikation von Objekten betrachten, besitzt jedes Objekt eine Einteilung in Klassen, jedes Objekt $\bar{x}_i$ eine Zuordnung zu $\mathcal{L}_i$. $\mathcal{L}_i$ ergibt sich aus Gleichung (2.1.2). Eine Klassifikation besteht somit aus Paaren $(\bar{x}_i, \mathcal{L}_i)$.

Beispiel: Es soll das Hautkrebsrisiko von Personen anhand der Patientendaten eines Hautarztes untersucht werden. Der Objekttyp ist somit Mensch, als Eigenschaften werden Alter und Hauttyp genommen. Es wird festgelegt $\mathcal{A}_{alter} = \mathbb{N}$, $\mathcal{A}_{typ} = \{$*Keltischer Typ, Germanischer Typ, Mischtyp, Mediterraner Typ*$\}$. $\mathcal{X} = \mathcal{A}_{alter} \times \mathcal{A}_{typ}$ würde alle Menschen abdecken. Die Krebsrisikoklassen wären $\mathcal{Y} = \{$*niedrig, mittel, hoch*$\}$. Eine Instanz könnte dann $\bar{x}_1 = (25,$ *Mischtyp*$)$ sein, die Klassifikation dieses Beispiels $(\bar{x}_1, \{mittel\})$.

Im vorangegangen Beispiel würde ein Lernalgorithmus die Patientendaten, also die Erfahrung des Arztes, als Eingabe erhalten. Da der Algorithmus mit diesen Daten *„trainiert“* wird, spricht man in diesem Zusammenhang von *Trainingsdaten*, *Trainingsbeispielen* oder einer *Trainingsmenge*. Ist zu jeder Person auch das Krebsrisiko bekannt und verfügt der Algorithmus über diese Daten, liegt der im folgenden Abschnitt vorliegende Fall vor.

2.3.1 Überwachtes Lernen

Man unterscheidet Lernprobleme u. a. nach der Verfügbarkeit einer Klassifikation der Beispiele. Ist zu jedem Beispiel die Klassenzugehörigkeit bekannt, spricht man von *überwachtem Lernen*[7]. Ist die Klassifikation nicht vollständig vorhanden, teilt man je nach Verfügbarkeit wiederum in *semi-überwachtes*, *Reinforcement-* und *nicht-überwachtes* Lernen ein. Wir beschränken uns in dieser Arbeit auf den Fall, daß die komplette Klassifikation bekannt ist.

[7] engl.: supervised learning

Bei unserem Arztbeispiel würde das bedeuten, daß ein Lerner als Trainingsmenge eine Folge von Paaren $(\bar{x}_i, \mathcal{L}_i)$ bekommen würde. Sei P die Anzahl der Beispiele, so ist die Trainingsmenge

$$\langle(\bar{x}_1, \mathcal{L}_1), (\bar{x}_2, \mathcal{L}_2), \ldots, (\bar{x}_P, \mathcal{L}_P)\rangle$$

Ein Lernverfahren erstellt anhand dieser Trainingsdaten ein Modell und ist nun in der Lage, für ein beliebiges Beispiel eine *Vorhersage* über die Klassenzugehörigkeit zu treffen. Das fertige Modell wird *Klassifizierer* genannt. Der Zweck eines Klassifizierers ist es, sich im Ergebnis möglichst wie die Klassifikationsfunktion f aus Gleichung (2.1.1) zu verhalten. Die Klassifizierfunktion eines Klassifizierers c ist somit:

$$\begin{aligned} &\mathcal{S} := \langle(\bar{x}_1, \mathcal{L}_1), \ldots, (\bar{x}_P, \mathcal{L}_P)\rangle \\ &c_{\mathcal{S}} : \mathcal{X} \to 2^{\mathcal{Y}} \\ &c_{\mathcal{S}}(\bar{x}) \stackrel{!}{=} f(\bar{x}) = \mathcal{L} \qquad , \forall \bar{x} \in \mathcal{X} \end{aligned} \tag{2.3.2}$$

Der tiefgestellte Index $\mathcal{S}$ gibt an, auf welchen Daten sich die Hypothese des Algorithmus stützt und kann, wenn klar ist, welche Trainingsmenge verwendet wird, weggelassen werden. Weiterhin gibt das Ausrufezeichen über dem Gleichheitszeichen die angesprochene Soll-Beziehung an. Es ist das Ziel eines Lerners, eine Klassifizierfunktion c zu erstellen, die Gleichheit zu der Klassifikationsfunktion f erreicht.

2.3.2 Bewertung von Klassifizierern

Um festzustellen, ob der Klassifizierer gut klassifiziert, also ob die Vorhersagen sich mit den tatsächlichen Klassifikationen decken, könnte man den Klassifizierer auf den Trainingsbeispielen anwenden und die Vorhersagen bewerten. Trifft der Lerner für die Trainingsbeispiele stets die richtige Vorhersage, so heißt der Algorithmus *konsistent*. Interessant ist aber die Genauigkeit auf beliebigen Beispielen. Erst anhand dieser Messung läßt sich feststellen, wie gut das erstellte Modell verallgemeinert. Der triviale Algorithmus, der sich alle Trainingsbeispiele merkt und die gemerkte Klassifikation zurückgibt, falls er das Beispiel schon kennt, ist zwar konsistent, besitzt aber kein gutes Modell, da er bei unbekannten Beispielen keine vernünftige Vorhersage treffen kann, er generiert kein zusätzliches Wissen über die Daten. Demzufolge ist es sinnvoll, die Bewertung nicht auf den Daten durchzuführen, die für das Training verwendet wurden, da das Modell von den Trainingsdaten abhängt.

Für die Bewertung eines Algorithmus werden deshalb zum Trainieren nicht alle verfügbaren Beispiele verwendet. Die verfügbaren Beispiele werden in eine Trainingsmenge, mit der der Lerner trainiert wird, und eine *Testmenge*, mit der die Güte des Algorithmus bestimmt wird, aufgeteilt. Die Idee dahinter ist erstens, daß das erzeugte Modell unabhängig von den Testbeispielen ist und eine Bewertung auf den Testdaten dadurch *„fairer“* ist und zweitens, daß die Testdaten stellvertretend sind für beliebige Beispiele. In der Regel wird für die Trainingsmenge ein Anteil von zwei Dritteln und für die Testmenge das restliche Drittel gewählt.

Die Leistung eines Klassifizierers auf einer Beispielmenge mißt man anhand des *Fehlers* auf der Beispielmenge. Die Leistung eines Algorithmus auf den Trainingsdaten ist der *Trainingsfehler*. Der *wirkliche Fehler* ist der Fehler, den ein Algorithmus auf beliebigen Beispielen machen würde. Dieser kann nicht berechnet werden, da dafür alle möglichen Beispiele mit Klassenzugehörigkeit

bekannt sein müßten. Die Hoffnung ist nun, daß der *Fehler auf den Testbeispielen* eine gute Annäherung an den wirklichen Fehler liefert.

Betrachte man $\bar{x}$ als Zufallsvariable auf der Verteilung $\mathcal{X}$, d. h. jedes $\bar{x} \in \mathcal{X}$ hat eine Wahrscheinlichkeit $\Pr(\bar{x})$ gezogen zu werden, sei $\mathcal{L}$ die Klassifikation von $\bar{x}$, sei δ eine Funktion, die ein aufsteigendes Maß für die Güte einer Vorhersage liefert, seien $\mathcal{S}$ und $\mathcal{T}$ Mengen von Beispielen aus $\mathcal{X}$ und voneinander unabhängig, d. h. ihre Schnittmenge sei leer (Trainingsmenge und Testmenge), so ist der wirkliche Fehler $\text{error}_{\mathcal{X}}$ und der Fehler auf $\mathcal{T}$ $\text{error}_{\mathcal{T}}$ des Klassifizierers $c_{\mathcal{S}}$ definiert als

$$\begin{aligned} &\delta : 2^{\mathcal{Y}} \times 2^{\mathcal{Y}} \to [0,\infty) \\ &\text{error}_{\mathcal{T}}(c_{\mathcal{S}}) = \frac{1}{Q} \sum_{\bar{x} \in \mathcal{T}} \delta(c_{\mathcal{S}}(\bar{x}), f(\bar{x})) \qquad , Q = |\mathcal{T}| \\ &\text{error}_{\mathcal{X}}(c_{\mathcal{S}}) = \mathop{\mathrm{E}}_{\bar{x} \in \mathcal{X}} [\delta(c_{\mathcal{S}}(\bar{x}), f(\bar{x}))] \end{aligned} \tag{2.3.3}$$

Die *Loss-Funktion* δ bewertet eine einzelne Vorhersage eines Klassifizierers. Sie wird auch *Kosten-Funktion* genannt. Eine perfekte Klassifizierung hat den Wert 0, sonst größer als 0. δ ermöglicht die differenzierte Bewertung einer Falschvorhersage. Die einfachste Loss-Funktion bewertet alle falschen Prognosen mit 1. Die Gleichbehandlung ist aber nicht immer von Interesse, z. B. ist die falsche Klassifizierung einer normalen Email als Spam schwerwiegender als der umgekehrte Fall der Klassifizierung einer Spam als normaler Email.

Wie Gleichung (2.3.3) zeigt, ist der Fehler auf einer bekannten Beispielmenge die durchschnittliche Bewertung der Loss-Funktion der Klassifizierung der darin enthaltenen Beispiele. Wir können somit $\text{error}_{\mathcal{T}}(c_{\mathcal{S}})$ kompakter schreiben als

$$\hat{\delta}(c_{\mathcal{S}}, \mathcal{T}) := \frac{1}{|\mathcal{T}|} \sum_{\bar{x} \in \mathcal{T}} \delta(c_{\mathcal{S}}(\bar{x}), f(\bar{x})) \tag{2.3.4}$$

Der Trainingsfehler eines Klassifizierers ist somit beispielsweise $\hat{\delta}(c_{\mathcal{S}}, \mathcal{S})$.

In Abschnitt 2.4.2 werden wir weitere Loss-Funktionen sehen. Es sei hier beispielhaft die erwähnte, als IsErrorLoss bezeichnete Funktion definiert, die nur eine korrekte und eine falsche Vorhersage kennt:

$$\delta_{IsErr}(c(\bar{x}), f(\bar{x})) := I[c(\bar{x}) \neq f(\bar{x})] \tag{2.3.5}$$

$I[x]$ ist die *Indikatorfunktion*, die einer wahren Aussage 1 und einer falschen 0 zuordnet:

$$I[x] := \begin{cases} 1 & \text{falls } x \text{ wahr ist} \\ 0 & \text{sonst} \end{cases} \tag{2.3.6}$$

Allgemein gilt, je höher die Anzahl der Testbeispiele Q, desto besser ist die Annäherung $\text{error}_{\mathcal{X}} \approx \text{error}_{\mathcal{T}}$. Konfidenzintervalle für die Approximation und eine ausführliche Beschreibung sind in [Mitchell, 1997, Kap. 5.2] nachzulesen.

Die Anzahl der verfügbaren Beispiele mit Klasseninformation ist oft stark begrenzt, da Klassifizieren (durch Beobachtung oder durch Menschen) teuer ist. Sowohl für ein gutes Training als auch für eine gute Evaluation werden aber möglichst viele Beispiele gebraucht. Hinzu kommt, daß sich

bei einer anteiligen Teilung[8] unter Umständen keine repräsentativen Trainings- und Testmengen ergeben, die Objekte einer bestimmten Klasse könnten beispielsweise mehrheitlich nur in der Trainingsmenge enthalten sein. Es existieren deshalb verschiedene Verfahren um die Zuverlässigkeit zu erhöhen. Von randomisierenden (*stratification*) hin zu kombinierenden (*cross-validation*) Verfahren. Für eine weiterführende Erörterung sei auf [Witten und Frank, 1999, Kap. 5.3, 5.4] verwiesen.

Beispiel: Nehmen wir an, im Beispiel auf Seite 14 besäße der Arzt 300 vollständige Patientendaten. Um einen Algorithmus zu analysieren, wird er mit den ersten 200 Beispielen trainiert und anschließend der daraus erstellte Klassifizierer auf die restlichen 100 Beispiele angewandt. Der Algorithmus klassifiziert 15 Beispiele falsch, damit ist unter Verwendung von δ_{isErr} der Fehler auf den Testdaten $\text{error}_{\mathcal{T}} = 15/100 = 0{,}15$. Damit ist zu erwarten, daß der wirkliche Fehler $\text{error}_{\mathcal{X}}$ um 15% liegt. Die Zuverlässigkeit dieser Approximation kann erhöht werden, indem in weiteren Durchläufen andere als die ersten 200 Beispielen zum Training verwendet werden und die Ergebnisse kombiniert werden. Ist die Leistung des Lernverfahrens zufriedenstellend, kann es nun mit den vollen 300 Beispielen trainiert werden und für neue Patienten anhand ihres Hauttyps und Alters und der gesammelten Erfahrung eine Prognose über das Hautkrebsrisiko treffen.

2.3.3 Inkrementelles und Batch-Lernen

Ein entscheidender Punkt für die Arbeitsweise eines Lernalgorithmus ist die Art der Verfügbarkeit der Trainingsbeispiele. Es gibt Lernalgorithmen, die für den Trainingsvorgang unbeschränkt über alle Beispiele verfügen müssen, die Trainingsmenge muß im Voraus vollständig bekannt sein und kann nicht nachträglich verändert werden. Das erstellte Modell steht nach dem Training fest. Diese Art des Lernens nennt sich *Batch-Lernen* oder *Offline-Lernen*.

Es ist jedoch nicht immer möglich, im Vorhinein alle Details der Trainingsmenge zu kennen und die Trainingsmenge insgesamt zur Verfügung zu stellen. Neue Beispiele könnten hinzukommen, der Speicherplatz für die Beispiele könnte beschränkt sein. Dies ist oft erwünscht oder wird sogar gefordert. Die Erfahrung, auf der das Modell basiert, kann sich im Laufe der Zeit verändern oder erweitern. Der Lerner muss das Modell aktualisieren können. Ein Lernalgorithmus, der einen Strom bzw. eine Folge von Beispielen erhält und der nach jedem Beispiel sein Modell anpaßt, lernt *inkrementell* oder *online*.

Ein direkter Vergleich von Batch- und Online-Algorithmen ist wegen der unterschiedlichen Bedingungen nicht durchführbar. Es ist jedoch möglich, einen Online-Lerner auf eine Weise einzusetzen, so daß er sich wie ein Batch-Algorithmus verhält und somit als Batch-Algorithmus einzusetzen ist. Die einfachste Methode besteht darin, den Algorithmus inkrementell alle Instanzen aus der Trainingsmenge lernen zu lassen und den nach dem letzten Beispiel generierten Klassifizierer zum klassifizieren (bzw. testen) zu verwenden. Zu beachten ist, daß das Ergebnis, das Modell, stark von der Folge von Beispielen abhängig ist, also von der Reihenfolge, in der dem Lernalgorithmus die Trainingsbeispiele übergeben werden. Oft läßt sich das Ergebnis des Lerners verbessern, indem er die Folge von Trainingsbeispielen mehrmals lernt. Diese Iterationen werden *Epochen* genannt.

[8] z. B. die erwähnte $\frac{2}{3}$-$\frac{1}{3}$-Teilung

2.3.4 Konservatives Lernen

Eine *konservatives* Lernverhalten ist eine Eigenschaft eines inkrementellen Lernverfahrens. Ein Lernalgorithmus arbeitet konservativ, wenn jede Anpassung der Hypothese dadurch verursacht wird, daß die aktuelle Hypothese nicht konsistent mit dem aktuellen Trainingsbeispiel ist. D. h. das Modell des Lerners wird nur dann aktualisiert, wenn das aktuelle Modell das erhaltene Trainingsbeispiel falsch klassifiziert. Verändert es das Modell auch bei einem korrektem Verhalten beim aktuellen Trainingsbeispiel, so arbeitet es nicht konservativ.

2.4 Multilabel-Klassifikation

Texte können nach verschiedenen Eigenschaften klassifiziert werden. So kann ein Text nach Thema, Sprache, Autor, Stil und verschiedenen anderen Gesichtspunkten unterschieden werden. Bei vielen dieser Zuordnungen ist eine Multilabel-Klassifikation anzutreffen, d. h. daß ein Dokument mehr als einer einzigen Kategorie zugeordnet werden kann und daß mehr als zwei Kategorien existieren. Beispielsweise läßt die Zuordnung von einem Dokument zum Autor mehr als nur einen Autor je Dokument zu.

2.4.1 Lernen von Multilabel-Klassifizierern

Für die automatische Klassifizierung von Multilabel-Textsammlungen sind demnach Klassifizierer notwendig, die in der Lage sind, eine Menge von Labels zurückzugeben. In Gleichung (2.3.2) ist ein solcher Klassifizierer spezifiziert. Die meisten existierenden Klassifizierer sind jedoch nur dafür ausgelegt, eine einzige Klasse vorherzusagen. Neue Verfahren sind somit nötig. Die Forschung im Bereich der reinen Multilabel-Algorithmen ist jedoch noch nicht weit fortgeschritten. Damit bekannte Lernalgorithmen dennoch auf Multilabel-Probleme anwendet werden können, existieren verschieden Ansätze, um sie an diese Problemklasse anzupassen. Die meisten setzen dabei erstens voraus, daß der Lerner Multilabel-Klassifikationen als Eingabe akzeptiert, und zweitens, daß der Klassifizierer zusätzlich Klassenkonfidenzen oder Klassenrankings ausgeben kann. Was dies bedeutet, wird im folgenden beschrieben.

Die erste Voraussetzung ist nicht selbstverständlich. Viele Lernverfahren funktionieren nach dem Prinzip, daß sie den Instanzenraum in disjunkte Bereiche aufteilen, die den Klassen entsprechen. Instanzen mit mehrfacher Zuordnung widersprechen diesem Prinzip und können nicht direkt von diesen Lernern modelliert werden.

Die zweite Bedingung bedeutet, daß der Klassifizierer im Stande sein muß, seine Vorhersage zu begründen. Dies ist möglich, wenn er intern den möglichen Ausgaben (bei Multiklassen-Lernern eine Klasse) eine Gewichtung zuordnet, anhand er über die Vorhersage entscheidet. Betrachten wir dazu einen *Bayes-Klassifizierer*. Dieser berechnet bei Eingabe einer Instanz zu jeder Klasse die Wahrscheinlichkeit, daß die Instanz dieser Klasse angehört. Die Klassifizierung entspricht dann der Klasse mit der höchsten Wahrscheinlichkeit. Die errechneten Gewichtungen für die Klassen nennt man *Konfidenzen* oder *Konfidenz-Werte*. Beinahe jeder Klassifizierer läßt sich so anpassen, daß er Konfidenzen angeben kann. Das Ergebnis ist dann ein Vektor $\bar{p} = (p_1, \ldots, p_K) \in \mathbb{R}^K$, wobei jedes

p_i der Konfidenz für die Klasse y_i entspricht. Sind für p_i nur endliche Werte möglich, so läßt sich stets eine Rechenvorschrift angeben, die den Konfidenzvektor in folgende Darstellung überführt.

$$\bar{\mathrm{p}} = (p_1, \ldots, p_K) \in [0,1]^K \,,\; \sum_{i=1}^{K} p_i = 1 \,, \text{ wobei hohe Werte von } p_i \text{ einer hohen Konfidenz entsprechen} \tag{2.4.1}$$

Diese Darstellung kann man als Wahrscheinlichkeitsverteilung der Klassen interpretieren. Der Klassifizierer gibt für jedes mögliche Ereignis der Klassenzugehörigkeit $X = y_i$ die Wahrscheinlichkeit unter der Bedingung an, daß Instanz $\bar{\mathrm{x}}$ vorliegt.

$$p_i = \Pr(X = y_i \,|\, \bar{\mathrm{x}}) = \Pr(f(\bar{\mathrm{x}}, y_i) = \mathit{wahr}) \tag{2.4.2}$$

Für solch ein zu einem probabilistischen Klassifizierer umgebautes Verfahren können wir aufbauend auf Gleichung (2.3.2) folgende Definition der Klassifizierfunktion verwenden

$$\begin{aligned} &c : \mathcal{X} \times \mathcal{Y} \to [0,1]^K \,,\; K = |\mathcal{Y}| \\ &c(\bar{\mathrm{x}}, \bar{\mathrm{y}}) = \Pr(f(\bar{\mathrm{x}}, \bar{\mathrm{y}}) = \mathit{wahr}) \,,\; \sum_{i=1}^{K} c(\bar{\mathrm{x}}, y_i) = 1 \\ &\text{oder kurz} \\ &\bar{\mathrm{p}} := (c(\bar{\mathrm{x}}, y_1), \ldots, c(\bar{\mathrm{x}}, y_K)) \end{aligned} \tag{2.4.3}$$

Anhand dieser Definition läßt sich eine Rangliste der Klassen angeben, das sogenannte *Klassenranking*, mit der wahrscheinlichsten Klasse an erster Stelle. Zur Einfachheit sei der Fall, daß zwei Klassen gleich bewertet werden, ausgeschlossen, so daß wir von einer strengen Totalordnung der Klassen ausgehen können und das nachfolgend präsentierte $\bar{\mathrm{r}}$ eine Permutation ist. In der Praxis erfolgt bei Gleichheit eine willkürliche Bevorzugung einer Klasse.

$$\begin{aligned} &\pi^n = \{(s_1, \ldots, s_n) \,|\, 1 \le s_i \le n, s_i \ne s_j, \forall j, i \ne j\} \subset [1, \ldots, n]^n \\ &\bar{\mathrm{r}} = (r_1, \ldots, r_K) \in \pi^K \\ &r_i := |\{p_i \,|\, p_i \le p_j, 1 \le j \le K\}| \,,\; \forall i, j.\; p_i \ne p_j \end{aligned} \tag{2.4.4}$$

Der Wert $\bar{\mathrm{r}}_\mathrm{i}$ gibt somit die Position der Klasse y_i im Ranking an.

Im folgenden werden einige Ansätze beschrieben, mit denen aus dem Konfidenzvektor oder dem Ranking ein Klassifizierer gewonnen werden kann, wie er anfangs in Gleichung (2.1.1) definiert wurde. Die einfachste Methode besteht darin, einen Schwellwert für die Konfidenz zu bestimmen. Die Klassen mit einem Konfidenzwert über dem Schwellwert θ werden dann vom Klassifizierer vorhergesagt.

$$c(\bar{\mathrm{x}}) := \{y_i \,|\, \bar{\mathrm{p}}_\mathrm{i} > \theta\} \tag{2.4.5}$$

Die Schwierigkeit besteht darin, einen passenden Schwellwert zu bestimmen. Ein guter Schwellwert kann entweder theoretisch oder experimentell ermittelt werden. Für den ersten Fall müssen die Konfidenzwerte des Algorithmus eine direkte Aussage zulassen, wie dies beispielsweise bei probabilistischen Klassifizierern der Fall ist. Für die experimentelle Ermittlung werden z. B. unterschiedliche Schwellwerte auf dem Traininssatz erprobt. Diese und andere Ansätze werden in [Sebastiani, 2002, Kap. 6.1] beschrieben.

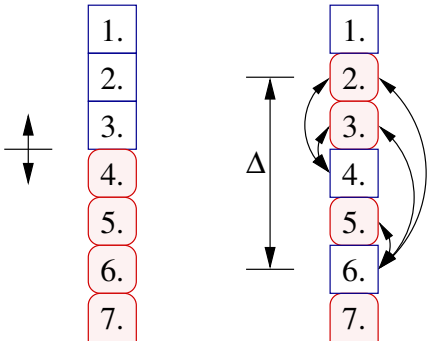


Abbildung 2.1: Bewertung von Rankings. Abgerundete Quadrate stellen die irrelevanten Klassen dar. Links: perfektes Ranking, rechts: $\delta_{IsErr} = 1$, $\delta_{margin} = 6 - 2 = 4$, $\delta_{ErrSetSize} = 5$ (Anzahl der Pfeilbeziehungen)

Eine andere Idee ist es, einen zusätzlichen Lernalgorithmus den Schwellwert lernen zu lassen. Dieser numerische Lerner[9] würde zu den Instanzen in der Trainingsmenge oder auch zu den Konfidenzvektoren den benötigten Schwellwert als Zielwert erhalten [siehe z. B. Elisseeff und Weston, 2001, Abschnitt 4]. Dafür ist allerdings die Vorhersage des *Basisklassifizierers* erforderlich. Bei einer fehlerhaften Vorhersage gestaltet sich zudem die Wahl des *„richtigen“* Schwellwerts problematisch.

Eine ähnliche Idee ist, einem zusätzlichen Lernverfahren die Anzahl der relevanten Klassen auf Basis der Instanzen lernen zu lassen. Dieser Ansatz wäre nicht mehr vom Basisklassifizierer abhängig. Zu dieser Idee wurden allerdings keine Referenzen gefunden.

2.4.2 Bewertung von Rankings

Wie aufgezeigt, existieren Verfahren, die aus Konfidenzwerten oder einem Ranking eine Menge von relevanten Klassen bestimmen können. Wir können uns deshalb auf die Analyse von Algorithmen konzentrieren, die für diese nachträgliche Bearbeitung des Resultats gute Basisdaten liefern, d. h. Konfidenzwerte oder Rankings. Zudem beschränken wir uns auf die Analyse von Rankings, da diese zwischen verschieden Klassifizierern im Gegensatz zu den Konfidenzwerten direkt verglichen werden können und deshalb allgemeingültiger sind. Zusätzlich ist eine Berechnung des Rankings aus den Konfidenzwerten ohne Verlust von Informationen für die Rangordnung möglich.

Lernalgorithmen, die als Trainingsmenge Beispiele mit einer Menge von zugeordneten Klassen erhalten und als Vorhersage eine geordnete Liste von Klassen sortiert nach deren Relevanz zurückgeben, nennt man *Klassen-Ranking-Algorithmen*.

Es bleibt nun festzustellen, was man als gute Basisdaten bezeichnen kann, d. h. wie man ein Ranking als Vorhersage eines Klassifizierers hinsichtlich einer gegebenen, zu lernenden Klassifikation einer Instanz beurteilt. Im folgendem werden einige Evaluierungsmethoden vorgestellt. In Abbildung 2.1 sind beispielhaft zwei Rankings mit Bewertungen dargestellt.

[9] engl.: numeric predictor

Perfektes Ranking Wir beginnen mit der auf Rankings angepaßten Definition der perfekten Klassifizierung: ein Ranking ist perfekt, genau dann wenn nur die relevanten oder positiven Klassen die obersten Plätze einnehmen. Die Reihenfolge, in der die relevanten Klassen angeordnet sind, ist dabei beliebig. Formal läßt sich das perfekte Ranking durch die Erweiterung der IsError-Lossfunktion aus Gleichung (2.3.5) auf Rankings beschreiben [Crammer und Singer, 2003].

$$\begin{aligned} &\delta_{IsErr} : \pi^K \times 2^{\mathcal{Y}} \to \{0,1\} \\ &\delta_{IsErr}(\bar{r}, \mathcal{L}) := 1 - \prod_{i=1}^{|\mathcal{L}|} I[y_{r_i} \in \mathcal{L}] \end{aligned} \tag{2.4.6}$$

Somit gilt $\delta_{IsErr} = 0$ bei einem perfekten Ranking und $\delta_{IsErr} = 1$ sobald eine negative Klasse über einer positiven in der Rangordnung steht. Aus der Definition der perfekten Klassifizierung folgt die Bedingung für alle Loss-Funktionen auf Rankings, daß sie ausschließlich dann den Wert Null besitzen wenn $\delta_{IsErr} = 0$ gilt.

Margin-Loss Der Nachteil des IsError-Loss ist, daß keine Differenzierung der nicht perfekt klassifizierten Rankings stattfindet, da sie alle den Fehler 1 erhalten. Eine Möglichkeit dies zu berücksichtigen, bietet das folgende einfache Verfahren, das wir *Margin-Loss* nennen. Der Abstand der relevanten Klassen zu den irrelevanten Klassen sei definiert als Differenz des Ranges der am schlechtesten platzierten, relevanten Klasse und der besten irrelevanten Klasse. Bei einem perfekten Ranking sei der Abstand 0. Formal läßt sich der MarginLoss folgendermaßen wiedergeben.

$$\begin{aligned} &\delta_{margin} : \pi^K \times 2^{\mathcal{Y}} \to \{0, \ldots, K-1\} \\ &\delta_{margin}(\bar{r}, \mathcal{L}) := \max(0, \max_i \{i \,|\, y_i \in \mathcal{L}\} - \min_j \{j \,|\, y_j \notin \mathcal{L}\}) \end{aligned} \tag{2.4.7}$$

Der Margin-Loss ist in Abbildung 2.1 als vertikaler Abstand zwischen der schlechtesten positiven Klasse und der besten negativen Klasse angegeben.

ErrorSetSize-Loss Ein ähnliches Maß stellt das *ErrorSetSize-Loss* dar. Er spielt insbesondere bei der Betrachtung des MMP-Algorithmus in Abschnitt 4 eine wichtige Rolle. Mit dem Error-Set bezeichnen Crammer und Singer [2003] die Menge der Paare von positiven und negativen Klassen, die in einem Ranking falsch geordnet wurden. Ein Paar ist falsch klassifiziert, wenn die irrelevante Klasse in der Rangordnung vor der relevanten steht. Die Menge der Fehlerpaare ist somit $\{(i,j) \,|\, \bar{r}_i > \bar{r}_j, y_i \in \mathcal{L}, y_j \notin \mathcal{L}\}$. Der ErrorSetSize-Loss gibt dabei die Anzahl der Fehlerpaare eines Rankings zurück.

$$\begin{aligned} &\delta_{ErrSetSize} : \pi^K \times 2^{\mathcal{Y}} \to \{0, \ldots, K^2/4\} \\ &\delta_{ErrSetSize}(\bar{r}, \mathcal{L}) := \left|\{(i,j) \,|\, r_i > r_j, y_i \in \mathcal{L}, y_j \notin \mathcal{L}\}\right| \end{aligned} \tag{2.4.8}$$

Die Anzahl der falschen Paare ist im schlimmsten Fall, wenn alle positiven Klassen die untersten Plätze belegen, durch $|\mathcal{L}|(K - |\mathcal{L}|)$ nach oben beschränkt. Die Fehlerpaare sind in Abbildung 2.1 mit Pfeilbeziehungen markiert.

Um den Fehler eines Klassifizierers auf einem Testdatensatz zu ermitteln, werden die Ranking-Fehler akkumuliert, d. h. über alle Testinstanzen summiert. Der durchschnittliche Fehler gibt dann den akkumulierten Fehler geteilt durch die Anzahl der Instanzen an. Der durchschnittliche IsError-

Loss besitzt eine interessante Eigenschaft. Er gibt den Anteil der Beispiele an, die der Klassifizierer nicht perfekt klassifizieren konnte. Wie beobachtet werden konnte, sind alle Losses Null wenn der IsError-Loss Null ist. Die ergibt die nützliche Eigenschaft, daß der akkumulierte Fehler jeder Loss-Funktion nur auf den falsch klassifizierten Daten ermittelt wird. So läßt sich anhand des durchschnittlichen IsError-Fehlers zusätzlich zu dem durchschnittlichen Fehler auf allen Daten der durchschnittliche Fehler auf den falsch klassifizierten Beispielen angeben. Dazu teilt man den durchschnittlichen Fehler durch das Mittel des IsError-Fehler

$$\hat{\delta}^*_{lossfunc} := \hat{\delta}_{lossfunc} / \hat{\delta}_{IsErr} \tag{2.4.9}$$

Recall und Precision *Recall* und *Precision* sind zwei Maße aus dem *Information Retrieval* und werden dort zur Bewertung von Suchergebnissen verwendet. Recall gibt die Vollständigkeit eines Suchergebnisses an, d. h. der Anteil der gefundenen relevanten Objekte verglichen zu der Gesamtanzahl der relevanten Objekte. Precision ist ein Maß für die Genauigkeit des Ergebnisses. Es gibt das Verhältnis zwischen der Anzahl der gefundenen relevanten Objekte und der Gesamtanzahl der gefundenen Objekte an.

Um diese Maße auf den Fall der Textklassifizierung zu übertragen, interpretiert man das zu klassifizierende Dokument als Suchanfrage und das zurückgegebene Ranking als Suchresultat. Bei Berücksichtigung des gesamten Rankings für die Berechnung ergibt sich allerdings kein vernünftiges Ergebnis, da Recall und Precision für alle möglichen Rankings gleich sind. Man beschränkt sich deshalb für die Berechnung das Ranking bis zu einem festgelegten Rang. Alle folgenden Klassen werden als nicht zurückgegeben angesehen. Wie ein solcher Schwellwert festgelegt werden kann, wurde zuvor beschrieben.

Wir formalisieren nun Recall und Precision. Sei i ein beliebiger Rang von einem Ranking $\bar{r}$, dann bezeichnet TP_i die Anzahl der richtig positiven Klassen bis zum Rang i (Anzahl der relevanten Klassen im Ergebnis), FP_i die Anzahl der falsch positiven Klassen (irrelevante Klassen im Ergebnis), TN_i die Anzahl der richtig negativen Klassen (irrelevante Klassen außerhalb der Ergebnismenge) und dementsprechend FN_i die Anzahl der falsch negativen Klassen bis zum Rang i (relevante Klassen außerhalb des Ergebnisses). Recall und Precision bis Rang i sind dann folgendermaßen definiert:[10]

$$recall_i := \frac{TP_i}{TP_i + FN_i} \tag{2.4.10}$$

$$precision_i := \frac{TP_i}{TP_i + FP_i} \tag{2.4.11}$$

Da der höchste Recall- und Precision-Wert jeweils 1 ist, können wir darauf aufbauend eine Lossfunktion definieren.

$$\begin{aligned} &\delta_{rcli} : \pi^K \times 2^{\mathcal{Y}} \to [0,1] \\ &\delta_{rcli}(\bar{r}, \mathcal{L}) := 1 - \frac{TP_i}{TP_i + FN_i} \end{aligned} \tag{2.4.12}$$

[10] Bei den Berechnungen von Recall und Precision ist zu beachten, daß eine gesonderte Behandlung der Null-Division nötig ist. Dies kann beispielsweise durch eine Laplace-Verschiebung oder durch Neudefinition der Null-Division erfolgen: $x/0 := 0\,,\ \forall x \in \mathbb{R}$. Aus Gründen der Übersichtlichkeit wird in dieser Arbeit fortan darauf verzichtet.

$$\begin{aligned} &\delta_{prc_i} : \pi^K \times 2^{\mathcal{Y}} \rightarrow [0,1] \\ &\delta_{prc_i}(\bar{r}, \mathcal{L}) := 1 - \frac{TP_i}{TP_i + FP_i} \end{aligned} \tag{2.4.13}$$

Eine Erhöhung des Precision-Wert geht im Allgemeinen mit einem niedrigeren Recall-Wert einher, und umgekehrt. Ein Abwägen beider Maße ist bei der Auswahl von i entscheidend. Eine genauere Beurteilung erlaubt eine Visualisierung in Form einer sogenannten *Recall-Precision-Kurve*. Hierzu werden in einem zweidimensionalen Graph die Punkte $(recall_1, precision_1)$ bis $(recall_K, precision_K)$ eingetragen und miteinander verbunden. Anhand dieser Kurve ist auch eine allgemeine Bewertung des Rankings möglich. Je näher sich die Kurve dem rechten oberen Punkt $(1,1)$ nähert bzw. je größer die Fläche unter der Kurve ist, desto besser ist das Ranking.

2.5 Textklassifizierung

Je nach der Beschaffenheit der zu klassifizierenden Daten ergeben sich unterschiedliche Voraussetzungen und Forderungen an die Lernalgorithmen. Eigenarten und Eigenschaften der verschiedenen Datentypen lassen sich speziell ausnutzen oder müssen gesondert berücksichtigt werden. Dies trifft auch für die *Klassifizierung von Text* zu. Dieser Abschnitt liefert einen kurzen Einblick in die Thematik der Textklassifizierung, soweit es erforderlich ist, um das Vorgehen in dieser Diplomarbeit zu verstehen. Einen ausführlichen Überblick über die Thematik liefert Sebastiani [2002].

Text ist definiert als im Wortlaut festgelegte, inhaltlich zusammenhängende Folge von Aussagen[11] oder auch als einen zusammenhängenden Bereich geschriebener Sprache[12]. Hier sei Text definiert als eine in sich geschlossene Folge von sprachlichen Einheiten. Die kleinste sprachliche Einheit ist das Wort, eine alphanumerische Zeichenkette. Übergeordnete sprachliche Einheiten sind Satz, Absatz, Kapitel, Überschrift etc. Sie werden nach bestimmten Regeln (z. B. der Syntax einer Sprache) konstruiert. Eine wichtige Eigenschaft eines Textes ist seine Struktur. Das Vorhandensein von übergeordneten sprachlichen Einheiten bestimmt die Komplexität der Struktur eines Textes. Ein strukturloser Text besitzt als sprachliche Einheit nur das Wort.

Eine andere Bezeichnung für einen Text ist *Dokument*. Angelehnt an die englische Bedeutung für *„term“* wird anstatt Wort auch der Ausdruck *Term* verwendet. Insbesondere bei der Textklassifizierung werden Klassen auch *Kategorien* genannt, demzufolge wird auch *Kategorisierung* für Klassifizierung verwendet.

2.5.1 Vorverarbeitung

Bevor eine Klassifizierung möglich ist, müssen die vorhandenen Daten so vorbereitet werden, daß ein Lerner sie verwendet kann. Die Lernalgorithmen, die wir in dieser Arbeit betrachten, benötigen als Eingabe Objekte in Form von Tupel von Attributen, wie in Abschnitt 2.3 (S. 14) erläutert. Für die Transformation des Textes in den Vektorraum sind zwei Schritte notwendig, das Parsen

[11] DUDEN - Das große Wörterbuch der deutschen Sprache, dritte Auflage, 1999
[12] wikipedia: *„Text“*

des Textes und anschließend die eigentliche Transformation in den Vektorraum. Je nach verwendetem Vokabular in den Texten und Größe des Korpus enstehen hochdimensionale Vektoren mit einer großen Anzahl an irrelevanten oder redundanten Attributen. Um die Menge der Attribute zu reduzieren, entfernt man in einem erstem Schritt als irrelevant schon bekannte Attribute in der Stoppwort-Auswahl. Redundante Features werden durch das Word-Stemming, einer morphologischen Analyse, zusammengefaßt. Die Anzahl der übriggebliebenen Attribute kann anschließend nochmals nach einer Analyse ihrer Relevanz für die Klassifizierung durch eine Feature Selection reduziert werden.

2.5.2 Parsen

Texte sind oft *„verpackt“* in Dateiformaten wie Word-Dateien, HTML-Dateien oder PDF-Dokumenten. Aber auch wenn der Text in Form von Klartext vorliegt, muß er vor der Weiterverarbeitung in eine interne Form transformiert werden. Nachdem gegebenenfalls der Klartext aus den *„Verpackungen“* extrahiert wurde, zerlegt ein lexikalischer Scanner die als simple Aneinanderreihung von Zeichen vorliegenden Eingabedaten in Token, in unserem Fall sind dies die Wörter. Diesen Schritt nennt man auch *Tokenization*. Das Ergebnis ist die interne Darstellung einer Folge von Wörtern, also ein strukturloser Text. Soll die Struktur des Textes für die Klassifizierung berücksichtigt werden, ist zudem noch eine syntaktische Analyse notwendig. Da die *„Verpackung“* nicht nur Informationen über die Darstellungsform des Textes, sondern auch Information über die Struktur enthalten kann, sollten diese Zusatzinformationen auch verwendet werden.

2.5.3 Vektorraummodell

Das Vektorraummodell eines Textes ist die algebraische Repräsentation eines Textes als Vektor in einem n-dimensionalen Raum. Im folgenden wird die einfachste Variante der möglichen Methoden der Transformation beschrieben.

Für jedes Wort im *Vokabular* der Textsammlung wird eine Dimension verwendet. Das Vokabular ist die Menge aller Wörter, die in allen Texten des Textkorpus auftauchen. Der Wert einer Vektorkomponente ist das Gewicht des Wortes, das der Dimension entspricht, im Text. Das Gewicht des Wortes (engl. *term weight*) ist ein Maß für die Bedeutung des Wortes in dem Dokument und hängt direkt von der Anzahl der Vorkommnisse des Wortes im Text ab. Die Berechnung wird weiter unten beschrieben.

Seien T der Textkorpus, t_i die Texte, $\mathcal{W}$ das Vokabular, $\mathcal{W}_T$ das Vokabular über dem Textkorpus T, tw die Funktion, die einem Wort in einem Text ein Gewicht zuordnet:

$$\begin{aligned} &T = \{t_1, \ldots, t_{|T|}\} \\ &\mathcal{W}_T := \{w \mid w \in t \in T\} = \{w_1, \ldots, w_N\},\ N := |\mathcal{W}_T|,\ \mathcal{W}_T \subseteq \mathcal{W} \\ &t_i = \langle w_{i_1}, w_{i_2}, \ldots, w_{i_{n_i}} \rangle,\ n_i := |t_i|,\ 0 < i_k \leq N \\ &\mathrm{tw} : T \times \mathcal{W}_T \to \mathcal{A},\ \mathcal{A} \subseteq \mathbb{R} \end{aligned} \tag{2.5.1}$$

So sieht die Repräsentation $\bar{\mathrm{x}}_\mathrm{i}$ eines Textes t_i folgendermaßen aus:

$$\begin{aligned} &\bar{\mathrm{x}}_\mathrm{i} \in \mathcal{A}^N \\ &\bar{\mathrm{x}}_\mathrm{i} = (a_1, \ldots, a_N),\ a_j = \mathrm{tw}(t_i, w_j) \end{aligned} \tag{2.5.2}$$

2.5.3.1 Verlust der Wortinterdependenzen

Bei der vorliegenden Methode gehen alle Informationen über die *Wortinterdependenzen* verloren. Die Wörter werden unabhängig voneinander betrachtet, die Reihenfolge der Wörter und der *Kontext* gehen verloren. Grundlage für diese Vorgehensweise ist die vereinfachende Annahme der bedingten Unabhängigkeit der Wörter, gegeben eine Klassifikation, wie sie auch beim *Naive-Bayes-Lerner* verwendet wird [Mitchell, 1997, S. 177]. Dies bedeutet, daß, gegeben eine Kategorie, die Wahrscheinlichkeiten des Vorkommens von zwei Wörtern unabhängig voneinander sind. Formal läßt sich das folgendermaßen beschreiben:

$$\Pr(w_i, w_j \mid \bar{y}) = \Pr(w_i \mid \bar{y}) \cdot \Pr(w_j \mid \bar{y}) \qquad \forall \bar{y} \in \mathcal{Y},\ \forall w_i, w_j \in \mathcal{W}, w_i \neq w_j \tag{2.5.3}$$

Die Annahme der Unabhängigkeit der Wörter[13] stellt eine grobe Vereinfachung des Problems dar, reduziert gleichzeitig aber auch die Komplexität der Modelle und Berechnungen. In der Praxis hat sich herausgestellt, daß diese Vereinfachung bei der Textklassifikation trotz Informationsverlust gute Ergebnisse liefert [Baldi u. a., 2003, S. 83].

Andere Modelle versuchen die Information über Wortinterdependenzen in Texten beizubehalten. Beispielsweise werden für eine Dimension statt einem einzelnen Wort Blöcke von Wörtern, z. B. *n*-hintereinander liegende Wörter (n-Gramme) oder Phrasen, genommen.

2.5.3.2 Wortgewichtung

Die einfachste Zuordnung von Gewichten zu Wörtern in einem Dokument stammt aus mengentheoretischen Überlegungen und besteht darin, einem Wort das Gewicht 0, wenn es in dem Dokument nicht enthalten ist, und 1, wenn es enthalten ist, zuzuordnen. Diese Darstellung wird *set-of-words*- oder boolesches Vektor-Modell genannt.

Die andere naheliegende *term weighting*-Methode sieht vor, einem Wort die Anzahl der Vorkommnisse, die *Worthäufigkeit*[14], im Dokument zuzuordnen. Die Gewichte sind möglicherweise normalisiert, geben also den Anteil der Worthäufigkeit an der Gesamtanzahl der Wörter im Dokument an. Diese Repräsentationsform wird als *bag-of-words*-Modell bezeichnet.

Dieses Modell geht davon aus, daß ein Wort, das in einem Dokument häufig vorkommt, für dieses Dokument wichtig ist, deshalb erhält es ein hohes Gewicht. Diese Ansicht beschränkt sich jedoch lokal auf ein Dokument. Es berücksichtigt nicht die globale Bedeutung eines Wortes. So ist die Wichtigkeit eines Wortes mit einer hohen Worthäufigkeit im Text, das global auch häufig vorkommt, verglichen mit der eines Wortes mit der gleichen lokalen Häufigkeit, jedoch mit einer niedrigen globalen Präsens, geringer.

Um diesen Aspekt zu berücksichtigen wird bei der Gewichtung die Worthäufigkeit oft mit der *Dokumenthäufigkeit* kombiniert. Die Dokumenthäufigkeit eines Wortes ist die Anzahl bzw. der Anteil der Dokumente, in denen das Wort vorkommt. Oft wird die *„logarithmierte“* Inverse der Dokumenthäufigkeit verwendet, die *inverse Dokumenthäufigkeit*, als Maß der Seltenheit eines Wortes (je höher der Wert, desto seltener).

[13] auch als Orthogonalität bezeichnet
[14] engl.: term frequency

Ausgehend von Gleichung (2.5.1), sei n die Funktion, die die Anzahl der Vorkommnisse eines Wortes in einem Text zurückgibt, so wird die Worthäufigkeit tf (*term frequency*) und die Dokumenthäufigkeit df bzw. die inverse Dokumenthäufigkeit idf (*inverse term frequency*) für ein Wort w in einem Text t_i wie folgt berechnet:

$$\begin{aligned} n(t_i, w) &:= \sum_{0<k\leq|t_i|} I[w = w_{i_k}] \\ \mathrm{tf}(t_i, w) &:= \frac{n(t_i, w)}{|t_i|} \\ \mathrm{df}(w) &:= \frac{|\{t_i \mid w \in t_i\}|}{P} \\ \mathrm{idf}(w) &:= \log\left(\frac{1}{\mathrm{df}(w)}\right) \end{aligned} \tag{2.5.4}$$

Nun werden die Worthäufigkeit und die Dokumenthäufigkeit zu der endgültigen Wortgewichtung kombiniert. Hierfür existiert eine lange Liste von Varianten [Sebastiani, 2002, Fußnote 5]. Die Variante, die am häufigsten zum Einsatz kommt, ist die Variante von Salton et al., bei der die Worthäufigkeit und Dokumenthäufigkeit einfach miteinander multipliziert wird [Baldi u. a., 2003, S. 84]:

$$\mathrm{tw}_{salton}(t_i, w) := \mathrm{idf}(w) \cdot \mathrm{tf}(t_i, w) \tag{2.5.5}$$

Es wird deutlich, daß die Gewichtung eines Wortes desto größer ist, je häufiger das Wort in dem Text vorkommt *und* je seltener das Wort in der Textsammlung ist.

Eine weitere *TF-IDF*-Variante wird in Abschnitt 6.3.3 vorgestellt.

Ist die Methode der term-weight-Berechnung so konzipiert, daß ein Wort mit der Worthäufigkeit Null auch eine Gewichtung von Null erhält, so enstehen sehr *„spärliche“* Vektoren, d. h. nur eine kleine Anzahl der Komponenten der Vektorrepräsentation sind ungleich Null, da Text die Eigenschaft hat, daß die Anzahl der in einem Text vorkommenden Worte viel kleiner ist als die Größe des Vokabulars: $|t| \ll |\mathcal{W}|$. Diese Eigenschaft kann für die Gestaltung der Speicherverwaltung und der Algorithmen vorteilhaft ausgenutzt werden.

2.5.4 Stoppwörter

Als Relevant für die Klassifizierung gelten Wörter, die durch ihr Vorhandensein bzw. ihrer Häufigkeit in einem Text Informationen über die Klassenzugehörigkeit liefern. Tritt ein Wort beispielsweise nur in einem Dokument auf, so ist dieses für die Klassifizierung irrelevant, da dieses Wort zwar einen starken Bezug zu dem Dokument, jedoch nicht zu den zu dem Dokument zugehörigen Klassen besitzt.

So wie sehr seltene Wörter irrelevant für die Klassifizierung sind, so sind dies auch sehr häufige Wörter. Ein Wort, das in allen oder fast allen Dokumenten einer Sammlung vorkommt, besitzt keinen Informationsgehalt für die Klassifizierung. Solche Wörter sind u. a. die häufigsten in einer Sprache verwendeten Wörter: Artikel, Konjunktionen, Präpositionen, Pronomen etc. Ist die Sprache einer Textsammlung bekannt, so ist es möglich, ohne weitere Kenntnisse über die Texte eine Anzahl

von Wörter zu bestimmen, die mit großer Wahrscheinlichkeit auch in der vorliegenden Sammlung sehr oft vorkommen und damit für die Klassifizierung irrelevant sind.

Solche Wörter nennt man *Stoppwörter*[15] und werden bei der Vorverarbeitung der Texte entfernt[16]. Die Anzahl solcher Wörter beträgt im Englischen beispielsweise bei Google 36[17], bei der Stoppwort-Liste des SMART Retrieval-Systems 571[18]. Beispiele sind *„a“*, *„about“*, *„be“*, *„how“*, *„of“*, *„on“*, *„that“*, *„the“*. Eine so geringe Zahl verglichen mit der Größe des Vokabulars, die bei Texten im Allgemeinen bei über 10.000 liegt und bis in den sechsstelligen Bereich reichen kann, läßt auf den ersten Blick an der Wirkung einer *Stoppwortreduktion* zweifeln. Wenn man jedoch bedenkt, daß die Anzahl der wirklich in einem Text verwendeten Wörter gering ist und daß die Stoppwörter in beinahe jedem (längeren) Text vorhanden sind, stellt man schnell fest, daß eine Stoppwortreduktion stark zur Verbesserung der Effizienz und Effektivität einer Klassifizierung beitragen kann.

2.5.5 Grundformenreduktion

Die *Grundformenreduktion*, auch *Stemming* genannt, ist eine morphologische Analyse, bei der die Wörter auf ihren Wortstamm zurückgeführt werden. Varianten eines Wortes enstehen z. B. durch Flexion oder Hinzufügen von Affixen. Ein Beispiel ist die Zurückführung von *„Klassifikation“* und *„Klassifizierung“* auf die Grundform *„klassif“*. Ein weitergehender Schritt wäre die Zurückführung auf *„klass“* und damit auch die Einbeziehung von *„Klasse“*, *„Klassen“* und anderen Varianten.

Anhand des Beispiels sieht man deutlich den Vorteil und den Nachteil des Stemmings. Semantisch äquivalente Wörter werden zusammengefaßt, so daß Redundanzen eingespart werden und die Dimensionalität verringert wird. Doch es werden auch Wörter zusammengefaßt, die von der Bedeutung (nur) ähnlich oder sogar vollständig unterschiedlich sind. Dadurch können entscheidende Informationen für die Klassifizierung verloren gehen.

Der *Porter-Stemmer-Algorithmus*[19] stellt ein bekanntes Stemming-Verfahren für Texte in englischer Sprache dar.

2.5.6 Feature-Selection

Leider bleibt auch nach den zuvor vorgestellten Verarbeitungsschritten die Liste der verwendeten Wörter lang, da ein recht hoher Anteil an irrelevanten Wörtern verbleibt. Um die Menge dieser Wörter auf eine vorgegebene Anzahl zu beschränken, bedarf es einer *Feature-Selection* auf den verbleibenden Daten. Das Verfahren entscheidet aufgrund der Relevanz der Wörter für die Klassifizierbarkeit, um welche Wörter der Datensatz reduziert wird. Dabei werden die Wörter nach ihrer Relevanz sortiert und entweder ab einem bestimmten Grad an Irrelevanz oder ab einer vorher festgelegten Anzahl der zu verbleibenden Wörter aus dem Vokabular entfernt. Für die Analyse der Relevanz wird im Normalfall nur die Trainingsmenge herangezogen, da sonst beim Training

[15] engl.: stop words
[16] ob dies vor oder nach der Transformation in den Vektorraum geschieht, ist nicht von Bedeutung
[17] www.ranks.nl/tools/stopwords.html
[18] ftp://ftp.cs.cornell.edu/pub/smart/english.stop
[19] tartarus.org/~martin/PorterStemmer/

Informationen über die Testmenge in die Bildung der Hypothese einfließen würde, was nicht Realitätsbedingungen entsprechen würde.[20] Im folgenden werden einige der für die Berechnung der Relevanz existierenden Ansätze vorgestellt.

Dokumenthäufigkeit Bei diesem Verfahren werden die Terme nach absteigender Dokumenthäufigkeit sortiert und solche entfernt, die einen vorgegebenen Wert oder aber eine vorgegebene Position in der Reihenfolge unterschreiten. Dieses Verfahren geht davon aus, daß selten verwendete Wörter irrelevant für die Klassifizierung sind, da sie zu spezifisch sind. Dieser Ansatz wird oft als ineffektiv und sogar naiv bezeichnet. Es wird weithin davon ausgegangen, daß selten vorkommende Wörter eine hohe Information für die Klassifizierung besitzen, da sie oft einer bestimmten Klasse zuzuordnen sind. Das Verfahren wird aus diesem Grund oft nur zur Effizienzsteigerung verwendet, da die Berechnung der Relevanz im Vergleich zu den anderen Methoden linear in der Anzahl Dokumente realisiert werden kann und keine Information zur Klassenzugehörigkeit nötig ist. Somit ist eine schnelle Reduzierung der Dimensionalität möglich. Die Annahme der schlechten Eignung dieses Verfahrens zur Feature-Selection wird von Untersuchungen durch Yang und Pedersen [1997] in Frage gestellt, die diesem Verfahren eine gute Leistung bei der Feature-Selection von Texten zugestehen. Die Ergebnisse aus Abschnitt 6.3.4 bestätigen das gute Abschneiden.

Information Gain Der *Informationsgewinn*[21] eines Attributs gibt an, wieviel Information für die Unterscheidung zwischen verschiedenen Klassen gewonnen wird oder verloren geht, wenn dieses Attribut weggelassen wird. So ist zum Beispiel ein Wort, das in Dokumenten einer bestimmten Klasse sehr häufig vorkommt, in Dokumenten der anderen Klassen jedoch kaum oder sogar nicht auftritt, als ein starker Informationsträger für die Abgrenzung dieser Klasse gegenüber den übrigen Klassen einzustufen. Ein bekanntes Beispiel für den Einsatz dieses Maßes stellt der ID3-Algorithmus zur Attributsauswahl in den Knoten von Entscheidungsbäumen dar [Mitchell, 1997, S. 55].

χ^2-Maß Das *χ^2-Maß* mißt die Abhängigkeit zwischen einem Attribut und einer Klasse. Eine große Korrelation zwischen Attributausprägung und Klassenzugehörigkeit wird als Relevanz eines Attributs für die Klassifizierung angesehen. Um die statistische Beziehung zwischen Attributen und Klassen zu berechnen, wird für jedes Dokument die Häufigkeit des Eintretens einer Attribut-Klasse-Kombination ermittelt. Das gleichzeitige Auftreten oder Ausbleiben einer Kombination über alle Dokumente hinaus deutet auf eine hohe Abhängigkeit zwischen beiden Variablen hin. Aus diesen Häufigkeiten wird zu jedem Attribut-Klasse-Paar ein entsprechender χ^2-Wert berechnet. Der maximale χ^2-Wert eines Attributs über alle Klassen wird anschließend für die Sortierung der Attribute verwendet. Die Auswahl der Attribute erfolgt ab einer festgelegten Position in der Reihenfolge.

Die beiden zuletzt beschriebenen Verfahren unterscheiden sich von der Sortierung nach Dokumenthäufigkeit in einem wichtigen Punkt. Sie benötigen für die Berechnung Klasseninformationen, d. h. die Klassifikation der einzelnen Instanzen muss vorher bekannt sein. Deshalb ist ihre Einsatzfähigkeit gegenüber der Selektion nach der Dokumenthäufigkeit eingeschränkt. Es muß allerdings erwähnt werden, daß auch der Feature-Selection nach Dokumenthäufigkeit eine Mindestanzahl von

[20] Als Ausnahme kann auf diese Bedingung verzichtet werden, falls nur ein Vergleich zwischen verschiedenen Algorithmen angestrengt wird und keine allgemeine Aussage über die Fähigkeit des Lerners unter Realitätsbedingungen angestrebt wird.

[21] engl.: information gain

Dokumenten (ohne Klassenzugehörigkeit) bekannt sein muß, um die Dokumenthäufigkeit wenigstens abschätzen zu können. Da häufig verwendete Attribute selektiert werden, ist auch die Mindestanzahl an Dokumenten, ab der das Verfahren erfolgsversprechend eingesetzt werden kann, im Vergleich zu Verfahren, die die Klassenzugehörigkeit benötigen, gering.

Zusätzlich ist die Laufzeit und der Speicherverbrauch der Feature-Selektion nach Dokumenthäufigkeit geringer. Die Berechnungsvorschriften für die vorgestellten Verfahren befinden sich in Abschnitt 6.3.4. Weitere Verfahren, wie *mutual information* und *term strength*, sind in Yang und Pedersen [1997] nachgeschlagen werden.

3 Der Perzeptron-Algorithmus

Das *Perzeptron* ist ein binärer Lernalgorithmus, der 1958 von Rosenblatt entwickelt wurde [Rosenblatt, 1958]. Er zählt zu den linearen Verfahren. Das Perzeptron basiert auf der Idee eines *künstlichen Neurons*. Ein künstliches Neuron ist die mathematische Modellierung bzw. Nachahmung einer biologischen neuronalen Zelle.[1] Um die Funktionsweise eines Perzeptrons zu verstehen, ist es hilfreich, einen Blick auf das natürliche Vorbild zu werfen.

3.1 Das Neuron als Vorbild

Neuronen sind Zellen im Körper eines Lebewesens, die für die Aufnahme, Weitergabe und Verarbeitung von Reizen zuständig sind. Sie bilden die grundlegende Einheit des Nervensystems. Ein Neuron ist über biologischen Übertragungskanäle, die so genannten Synapsen, mit vielen anderen Neuronen verbunden. Ein Neuron empfängt über seine Dendriten Signale der verbundenen Nervenzellen oder auch von Sinneszellen, die Signale aus der Umwelt aufnehmen (Informationsaufnahme). Die interneuronalen Signale sind elektrische Impulse, Spannungspotenziale, und können unterschiedliche Spannungsstärken annehmen. Im Zellkern setzt ein Neuron die empfangenen Signale zusammen, indem es sie nach einem im Neuron festgelegten Muster zu einem Potenzial kombiniert (Informationsverarbeitung). Dieses Potenzial wird zu einem Impuls am Ausgang des Neurons, am Axon, umgewandelt (Aktivierung) und über Synapsen an andere Neuronen weitergeleitet (Informationsweiterleitung). Vereinfacht bzw. idealisiert unterscheidet man zwischen einer Aktivierung und einer Nicht-Aktivierung des Neurons (durch die Eingangsimpulse), es sind jedoch auch Zustände dazwischen möglich.[2]

3.2 Künstliches Neuron

Ein künstliches Neuron ist eine reellwertige Funktion, die eine Menge von Eingangsgrößen auf eine Ausgangsgröße abbildet. Die Funktionsargumente entsprechen dabei den elektrischen Eingangssignalen an den Dendriten des biologischen Neurons und der Funktionswert dem Ausgangsimpuls am Axon. Die Verarbeitung der Eingangsdaten zu einem Ausgangswert erfolgt dabei wie beim Original in zwei Schritten.

In einem ersten Schritt werden die Eingangswerte $a_1, \ldots, a_N$ anhand der im Neuron hinterlegten Gewichtungen der Eingänge $w_1, \ldots, w_N$ kombiniert. Obgleich für die Kombinierung jede beliebige Verknüpfung von Operationen in Frage käme und biologische Neuronen auch eine Vielzahl davon ermöglichen (Addition, Multiplikation, Minima-, Maxima-, Mittelwertberechnung ...), beschränkt

[1] Beschreibung des künstlichen Neurons nach [Reif, 2000, Abschnitt 8.2] und [MacKay, 2005, Abschnitt 39.1]
[2] vgl. www.drd.de/helmich/bio/neu/reihe1/ur11/neuron.html

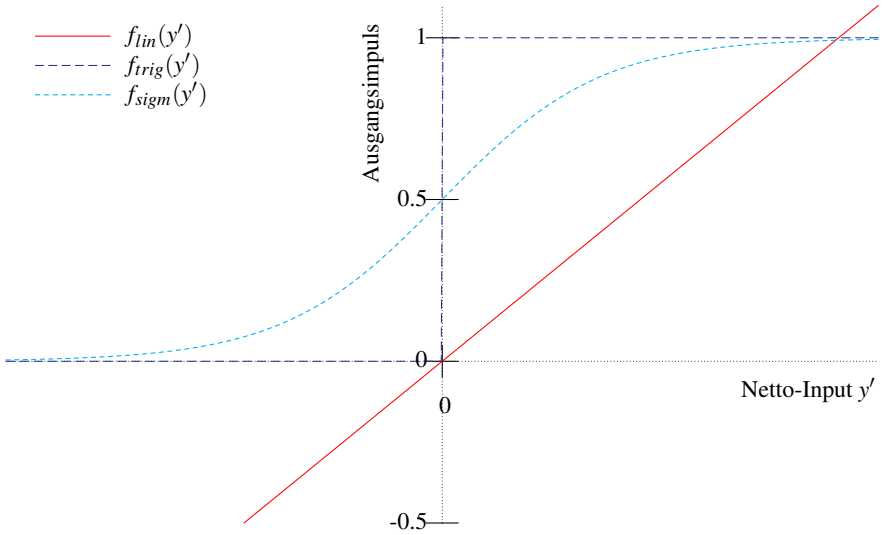


Abbildung 3.1: Grundtypen der Aktivierungsfunktion eines künstlichen Neurons

man sich beim Modell des künstlichen Neurons auf die Linearkombination der Eingangswerte mit den Eingangsgewichten als Koeffizienten.

Betrachtet man die Eingangswerte und Gewichte als Vektoren $\bar{x} = (a_1, \ldots, a_N)$ und $\bar{w} = (w_1, \ldots, w_N)$ und sei $\bar{x} \cdot \bar{w}$ das Skalarprodukt beider Vektoren, so läßt sich die gewichtete Summe der Eingangswerte folgendermaßen darstellen (Vektorraum ist der euklidische Raum $\mathbb{R}^N$):

$$y' = \sum_{i=0}^{N} w_i a_i = \bar{w} \cdot \bar{x} \tag{3.2.1}$$

In einem zweiten Schritt berechnet die Aktivierungsfunktion aus der Linearkombination y', auch *Netto-Input* genannt, das Ausgangssignal des Neurons. Üblicherweise wird als Aktivierungsfunktion eine der drei nachfolgend beschriebenen Funktionsgrundtypen verwendet, graphisch dargestellt in Abbildung 3.1.

Lineare Funktion Die lineare Funktion gibt als Ausgabewert den Eingangswert ohne Manipulation zurück.

$$f_{lin}(y') = y' \tag{3.2.2}$$

Diese Funktion besitzt für die Modellierung von Neuronalen Netzen keinen großen Nutzwert. Ein Netz von so entworfenen künstlichen Neuronen läßt sich durch ein einfaches künstliches Neuron ersetzen, da die lineare Kombinierung von linearen Funktionen wiederum linear ist.

Schwellwertfunktion Mittels der Schwellwertfunktion wird ein idealisiertes Neuron modelliert. Übertrifft die Kombination der Eingangsreize einen festgelegten Wert, wird das Neuron aktiviert und gibt einen Impuls ab. Es erlaubt, wie ein Computerbit oder eine ideale Diode, genau zwei Zustände des Neurons.

$$f_{trig}(y') = \begin{cases} 1 & \text{falls } y' \geq 0 \\ 0 & \text{falls } y' < 0 \end{cases} \tag{3.2.3}$$

Bei Gleichung (3.2.3) wurde der Schwellwert auf 0 und der aktivierende Impulswert auf 1 bzw. das Nicht-Aktiv-Signal auf 0 festgelegt. Die abgebildete Gleichung stellt nur die Grundfunktion vor, d. h. sie ist repräsentativ für alle möglichen Schwellwertfunktionen. Andere Schwellwertfunktionen unterscheiden sich nur in den Parametern und lassen sich auf die Grundfunktion zurückführen.

Sigmoide Funktion Die *sigmoide Funktion* bildet beliebige reelle Werte auf ein festgelegtes Intervall ab.

$$f_{sigm}(y') = \frac{1}{1+\exp^{-y'}} \tag{3.2.4}$$

Der Netto-Input y', der je nach Konstellation sehr hohe bzw. sehr niedrige Werte einnehmen kann, kann somit in das Intervall projiziert werden, das für (gültige) Impulse vorher festgelegt wurde, üblicherweise in den reellen Wertebereich $(0,1)$ oder $(-1,1)$. Im Unterschied zur Schwellwertfunktion erlaubt das ausgegebene Signal der sigmoiden Funktion eine differenziertere Aussage. Die sigmoide Funktion stellt eine Schwellwertfunktion mit weicher Grenze dar, die im Gegensatz zur Funktion mit starrer Schwelle stetig ist und sich aus diesem Grund mathematisch besser modellieren läßt. Zudem nähert sie sich am ehesten dem Verhalten einer biologischen Nervenzelle. Auch Gleichung (3.2.4) stellt nur die Grundfunktion für die sigmoiden Aktivierungsfunktionen dar.

Das Modell des künstlichen Neurons läßt sich nun als Vorlage für einen einfachen Klassifizierer verwenden. Ein Neuron mit Schwellwertfunktion entscheidet aufgrund der eingehenden Reize, ob es aktiviert wird oder nicht. Es ist offensichtlich in der Lage, zwischen zwei Klassen von Reizen zu unterscheiden, gleicht in der Funktion somit einem binären Klassifizierer. Die eingehenden Reize entsprechen den zu klassifizierenden Objekten und die Aktivierung der Zuweisung zu einer der zwei möglichen Klassen. Die interne Kombinierung der Eingangsreize und die Wahl der Aktivierungsfunktion stimmt mit der Hypothese eines binären Lerners überein.

Einen solchen Klassifizierer nennt man Perzeptron, seine Funktionsweise wird im nächsten Abschnitt beschrieben.

3.3 Beschreibung des Klassifizierers

Als Eingangswerte verwendet man die als Vektor repräsentierten Objekte $\bar{x} = (a_1, \ldots, a_N)$. Sie unterliegen einer Zweiklassen-Klassifikation mit einfacher Zuordnung. Als Aktivierungsfunktion verwendet man die Schwellwertfunktion. Die Hypothese des Lerners setzt sich aus dem Schwellwert ϑ der Schwellwertfunktion und der Gewichtung der Eingangswerte, repräsentiert durch einen Vektor $\bar{w} = (w_1, \ldots, w_N)$, zusammen. Die Aktivierung des Neurons entspricht der Klassifizierung als positives Beispiel, eine Nicht-Aktivierung der Zuordnung zu den negativen Beispielen. Sei y_0 die Klasse

der positiven Beispiele, y_1 die der negativen Beispiele. Es ergibt sich folgende Klassifizierfunktion für das Perzeptron

$$c_{\bar{w},\vartheta} : \mathcal{X} \to 2^{\mathcal{Y}}$$
$$c_{\bar{w},\vartheta}(\bar{x}) = \begin{cases} \{y_0\} & \text{, falls } \sum_{i=0}^{N} a_i w_i \geq \vartheta \\ \{y_1\} & \text{, falls } \sum_{i=0}^{N} a_i w_i < \vartheta \end{cases} \tag{3.3.1}$$

Betrachtet man die Objekte als Punkte in einem N-dimensionalen euklidischen Raum, so beschreibt ein Perzeptron eine (N-1-dimensionale) Hyperebene in diesem Raum. Diese Hyperebene teilt den Raum in zwei Hälften, in der einen befinden sich die Objekte der Positivklasse, in der anderen die Objekte der Negativklasse. Die interne Beschreibung $\bar{w}$ gibt den Normalenvektor der Hyperebene und ϑ den Abstand zum Ursprung an. Der Normalenvektor steht dabei senkrecht zur Ebene, spannt sie somit auf, und gibt die Orientierung der gerichteten Hyperebene an, er zeigt in Richtung der positiven Beispiele. Die Gleichung der Hyperebene läßt sich wie folgt angeben:

$$H : \bar{x} \cdot \bar{w} = \vartheta \tag{3.3.2}$$

Der Normalenvektor ist nicht notwendigerweise normalisiert, so gibt ϑ im Vergleich zur Hesseschen Normalform nicht direkt den Abstand zum Ursprung an. Der (senkrechte) Abstand zum Ursprung beträgt $\frac{\vartheta}{|\bar{w}|}$. Der nächste Punkt der Hyperebene vom Ursprung ist damit $\frac{\vartheta}{|\bar{w}|} \cdot \frac{\bar{w}}{|\bar{w}|} = \frac{\vartheta}{\bar{w}^2}\bar{w}$ (Stützpunkt).

Genau betrachtet gehören nach Gleichung (3.3.1) auch die Objekte *in* der Hyperebene zur positiven Klasse. Die richtige Herangehensweise wäre, den Fall $\sum_{i=0}^{N} a_i w_i = \vartheta$ getrennt zu betrachten, da Punkte *„in der Mitte“* korrekterweise nicht pauschal einer der Klassen zugeordnet werden können. Punkte auf der Hyperebene würde man als unklassifizierbar, beiden Klassen angehörend oder keiner Klasse angehörend ansehen. Das Perzeptron wäre *symmetrisch*, die Invertierung der Klassifikation würde die Invertierung der Klassifizierung bedeuten. Nachteil dieser exakteren Differenzierung ist die Veränderung der Antwortmöglichkeiten des Klassifizierers, der nun mehr als zwei Antworten kennt. Dies ist nur möglich, wenn die Mächtigkeit der Klassifikation dies zuläßt oder der Empfänger der Antwort, der Benutzer des Klassifizierers, den zusätzlichen Antworttyp verarbeiten kann. Um dieses Problem zu umgehen, aber dennoch die positive Klasse durch die Einbeziehung der Punkte auf der Hyperebene nicht zu bevorzugen (Balanciertheit), ist es denkbar, die Punkte auf der Hyperebene zufällig einer der Klassen zuzuordnen. Bei dieser Lösung würde allerdings die *Determiniertheit* des Perzeptrons verloren gehen, folglich auch die Symmetrie. Aus diesen Gründen wird im Allgemeinen die Sichtweise des Perzeptrons als Neuron akzeptiert, der ab einer bestimmten Signalschwelle auslöst. Die Auswirkungen der inkludierenden Abgrenzung auf die Symmetrie können als gering eingeschätzt werden (das Verhalten eines nach Gleichung (3.3.1) definierten Perzeptron kann in der Praxis quasi als symmetrisch betrachtet werden). Zudem wird nachfolgend gezeigt, daß im Fall der korrekten Klassifizierbarkeit einer Datenmenge durch einen Perzeptron sich immer ein Perzeptron angeben läßt, das relativ zu dieser Datenmenge symmetrisch ist.

Die geometrische Betrachtung macht auf einfache Art und Weise klar, worin die Grenzen der Verwendung eines Perzeptrons als Lerner liegen. Sind die Punkte der positiven Beispiele nämlich nicht von den Punkten der negativen Beispiele im euklidischen Raum durch eine Hyperebene trennbar, so kann ein Perzeptron das Klassifizierproblem nicht (fehlerfrei) lösen. Als Beispielszenario mag hier der zweidimensionale Raum dienen.

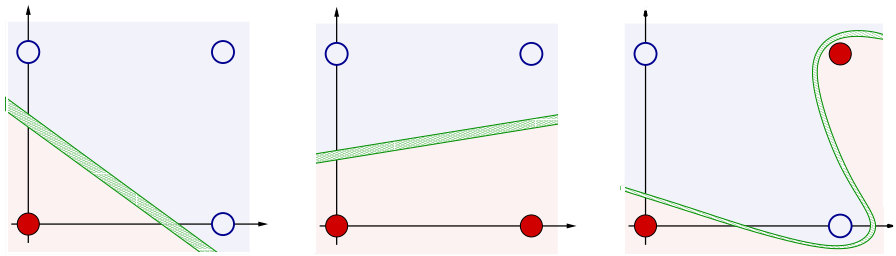


Abbildung 3.2: Lineare Separierbarkeit von vier Punkten im zweidimensionalen Raum. Linke und mittlere Anordnung linear trennbar, rechte Anordnung nicht durch eine Gerade trennbar

Beispiel: Zwei (ungleiche) Punkte sind im zweidimensionalen euklidischen Raum immer durch eine Gerade (in diesem Raum eine Hyperebene) trennbar. Bei drei Punkten läßt sich in den meisten Konstellationen eine trennende Gerade angeben. Der Spezialfall, bei dem drei Punkte auf einer Geraden liegen, läßt jedoch schon keine trennende Gerade mehr zu. Mit wachsender Anzahl von Punkten erhöht sich die Anzahl der Konstellationen, wo eine Separierbarkeit nicht mehr möglich ist. Betrachten wir vier Punkte $(0,0)$, $(1,0)$, $(1,1)$ und $(0,1)$, die zusammen die Ecken eines Quadrats bilden (siehe Abbildung 3.2). Jeder Punkt ist von den anderen drei Punkten durch eine Gerade trennbar, und jeder Punkt ist zusammen mit einem seiner Nachbarpunkte (nächste Punkte entlang der Quadratskanten) von den anderen beiden Punkten (dem anderen Nachbarpunkt und dem gegenüberliegenden Punkt) trennbar. Doch es gibt keine Gerade, die zwei gegenüberliegende Punkte von den beiden Nachbarpunkten trennen könnte.

Bisher wurde Separierbarkeit nicht formal definiert, dies wird nun nachgeholt. Zwei Punktmengen $\mathcal{A}$ und $\mathcal{B}$ in einem N-dimensionalen Raum $\mathbb{R}^N$ heißen *linear trennbar* oder *linear separierbar*, wenn es einen Vektor $\bar{w}$ und ein ϑ gibt, so daß gilt

$$\sum_{i=0}^{N} a_i w_i = \begin{cases} \geq \vartheta & \forall \bar{x} \in \mathcal{A} \\ < \vartheta & \forall \bar{x} \in \mathcal{B} \end{cases} \tag{3.3.3}$$

Sind $\mathcal{A}$ und $\mathcal{B}$ disjunkt, so sagt man auch vereinfachend, die Menge $\mathcal{A} \cup \mathcal{B}$ sei linear separierbar. Übertragen auf unseren Fall bedeutet dies: sind die Beispielmengen $f^{-1}(y_0)$ und $f^{-1}(y_1)$ aus der binären Datenmenge $\mathcal{X}$ linear separierbar, so existiert (mindestens) ein Perzeptron, das die Beispiele aus $\mathcal{X}$ richtig klassifiziert. Ein Perzeptron ist somit potentiell in der Lage, linear separierbare binäre Datenmengen ohne Fehler zu klassifizieren.

Von Interesse ist zudem die Definition der absoluten linearen Separierbarkeit und der damit verbundenen Aussage. Zwei Punktmengen $\mathcal{A}$ und $\mathcal{B}$ heißen *absolut linear separierbar*, wenn es einen Vektor $\bar{w}$ gibt, so daß gilt

$$\sum_{i=0}^{N} a_i w_i = \begin{cases} > \vartheta & \forall \bar{x} \in \mathcal{A} \\ < \vartheta & \forall \bar{x} \in \mathcal{B} \end{cases} \tag{3.3.4}$$

Daraus folgt unmittelbar: sind zwei Punktmengen $\mathcal{A}$ und $\mathcal{B}$ endlich und linear trennbar, so sind sie auch absolut trennbar. Für linear trennbare Daten gilt somit, daß sich ein Perzeptron finden läßt, das symmetrisch in der Klassifizierung ist.

Die Darstellung der Klassifizierfunktion läßt sich dahingehend verändern, daß als Schwellwert 0 verwendet werden kann. Dies erleichtert die mathematische Analyse. Wir ergänzen für den soeben genannten Zweck den Vektor $\bar{x}$ durch eine Komponente mit dem konstanten Wert 1 und $\bar{w}$ erhält als zusätzliche Gewichtung für den neuen Eingang den Schwellwert $-\vartheta$. Die kompaktere Darstellung sieht nun folgendermaßen aus:

$$c_{\bar{w}} : \mathcal{X} \to 2^{\mathcal{Y}}$$
$$c_{\bar{w}}(\bar{x}) = \begin{cases} \{y_0\} & \text{, falls } \sum_{i=0}^{N+1} a_i w_i \geq 0 \\ \{y_1\} & \text{, falls } \sum_{i=0}^{N+1} a_i w_i < 0 \end{cases} \tag{3.3.5}$$

Die Übereinstimmung dieser Darstellung mit Gleichung (3.3.1) ist leicht durch folgende Rechnung nachprüfbar: $\sum_{i=0}^{N+1} a_i w_i = (\sum_{i=0}^{N} a_i w_i) + a_{N+1} w_{N+1} = (\sum_{i=0}^{N} a_i w_i) - \vartheta$. Die geometrische Interpretation verändert sich dahingehend, daß die trennende Hyperebene durch den Ursprung verläuft. Die Gleichung der Hyperebene lautet nun $H_{\bar{w}} : \bar{x} \cdot \bar{w} = 0$ mit den erweiterten Vektoren. Da das Verhalten des Perzeptrons vollständig durch den Vektor $\bar{w}$ festgeschrieben ist, verwendet man oft den Begriff Perzeptron als Bezeichnung für den Gewichts- bzw. Normalenvektor des Perzeptrons, so wie ebenfalls der Begriff Instanz allgemein für den Vektor, der die Instanz beschreibt, verwendet wird.

3.4 Die Perzeptron-Lernregel

Bisher haben wir uns nur mit der Methode der Klassifizierung beschäftigt. Diese entspricht im Grunde der Funktionsweise eines künstlichen Neutrons und war schon vor dem Perzeptron bekannt. Rosenblatt entwickelte nun mit seinem Perzeptron ein Verfahren, um die neuronale Zelle zu trainieren, dies war seine eigentliche Errungenschaft.

Der Perzeptron-Lernalgorithmus ist ein einfaches überwachtes Lernverfahren. Er arbeitet inkrementell, erhält also eine Folge von Trainingsinstanzen mit bekannter Klassenzugehörigkeit. Der Pseudocode ist in Abbildung 3.3 angegeben. Zusammengefaßt kann man den Algorithmus folgendermaßen beschreiben: durchlaufe alle Trainingsinstanzen und klassifiziere jede Instanz mit dem aktuellen Perzeptron. Wird die Instanz richtig klassifiziert, so tue nichts, sonst addiere den Instanzvektor zum Perzeptronvektor, wenn das Beispiel ein positives Beispiel ist, und subtrahiere den Instanzvektor vom Perzeptronvektor, wenn es sich um ein negatives Beispiel handelt. Dabei wird die Instanz jeweils mit einem Lernfaktor skaliert.

Da der Perzeptron-Algorithmus inkrementell lernt, ist nach jeder Iteration, d. h. nach jedem zu lernenden Beispiel, ist die Angabe der Hypothese möglich. Die Anzahl der Lernbeispiele ist nicht beschränkt oder vorher festgelegt und die Reihenfolge ist vorgegeben. Im vorliegenden Pseudocode allerdings wird das Perzeptron als Batch-Algorithmus dargestellt. Die Trainingsmenge ist vorher bekannt und ein Ergebnis nach dem letzten Trainingsbeispiel zu erwarten. Die Umstellung im Sinne von Abschnitt 2.3.3 soll die Darstellung und die nachfolgende Analyse vereinfachen. Der inkrementelle Charakter des Algorithmus ist jedoch klar zu erkennen. Die Verarbeitung der Trainingsmenge erfolgt in einer festen (vorgegebenen) Reihenfolge und der Gewichtsvektor muß rekursiv in jeder Iteration bestimmt werden, da er für die Vorhersage in Zeile 8 benötigt wird.

Das Hinzuaddieren oder Subtrahieren der Instanz kann geometrisch folgendermaßen interpretiert werden. Wird eine Instanz falsch klassifiziert, so bedeutet dies, daß sich die Instanz auf der falschen

Eingabe: Trainingsmenge $\langle(\bar{x}_1, \mathcal{L}_1), (\bar{x}_2, \mathcal{L}_2), \ldots, (\bar{x}_P, \mathcal{L}_P)\rangle$, Lernrate θ

```
1:  initialisiere w̄_1
2:  k ← 1                                          ▷ initialisiere Laufvariable
3:  while i ≤ P do                                 ▷ Iteriere über alle x̄_i in Trainingsmenge
4:      if L_i = {y_0} then                        ▷ x̄_i ist ein positives Beispiel
5:          η_i ←  1                               ▷ falls falsche Vorhersage, addiere x̄_i
6:      else                                       ▷ x̄_i ist ein negatives Beispiel
7:          η_i ← −1                               ▷ falls falsche Vorhersage, subtrahiere x̄_i
8:      if c_{w̄_k}(x̄_i) ≠ L_i then                 ▷ falls falsch klassifiziert
9:          w̄_{k+1} ← w̄_k + θ η_i x̄_i              ▷ aktualisiere das Perzeptron
10:         k ← k + 1                              ▷ erhöhe Laufvariable
11: return w̄_{k+1}                                ▷ Perzeptron auf Basis der Trainingsmenge
```

Abbildung 3.3: Perzeptron-Lernalgorithmus

Seite der Hyperebene befindet bzw. die Hyperebene den Raum an der falschen Stelle teilt. Der Normalenvektor der Hyperebene zeigt nicht genügend in Richtung des positiven Beispiels. Um dies zu korrigieren, wird dem Normalenvektor der Ortsvektor der Instanz aufaddiert, was als Folge eine Translation des Zielpunktes des Normalenvektors (wenn man als Startpunkt den Ursprung annimmt) in Richtung der Instanz hat. Der Normalenvektor wird sozusagen in Richtung der Instanz gedreht bzw. gezogen. Wenn wir den Abstand eines falsch klassifizierten Punktes zur Hyperebene als negativ bezeichnen, so erhöht sich der Abstand nach der Aktualisierung, der Winkel zwischen Ortsvektor und Normalenvektor verringert sich. Bei falsch klassifizierten negativen Beispielen wird analog der Normalenvektor von der Instanz sozusagen weggedrückt.

Es wurde bisher anschaulich dargestellt, daß nach einer Aktualisierung des Normalenvektors eine falsch klassifizierte Instanz näher an der richtigen Hälfte des Raumes liegt als zuvor (von der Hyperebene aus betrachtet, die Instanz bewegt sich nicht). Dies wird nun rechnerisch anhand der Veränderung des Skalarprodukts nach der Aktualisierung gezeigt. Betrachten wir dazu ohne Beschränkung der Allgemeinheit den Fall, daß ein positives Beispiel falsch klassifiziert wurde $\bar{w}_k \bar{x}_i < 0$. Es läßt sich leicht zeigen, daß das Skalarprodukt nach der Aktualisierung größer wird:

$$\bar{w}_{k+1}\bar{x}_i = (\bar{w}_k + \theta\bar{x}_i)\bar{x}_i = \bar{w}_k\bar{x}_i + \theta\bar{x}_i^2 > \bar{w}_i\bar{x} \tag{3.4.1}$$

Das Skalarprodukt zweier Vektoren hängt unmittelbar mit dem Winkel zwischen beiden Vektoren zusammen

$$\frac{\bar{w}\bar{x}}{|\bar{w}||\bar{x}|} = \cos\sphericalangle(\bar{w}, \bar{x}) \tag{3.4.2}$$

Auch der Abstand zur Hyperebene (gemessen als Länge der senkrechten Projektion des Ortsvektors auf den normalisierten Normalenvektor) läßt sich anhand dieser Gleichung bestimmen

$$\frac{\bar{w}\bar{x}}{|\bar{w}|} = |\bar{x}|\cos\sphericalangle(\bar{w}, \bar{x}) \tag{3.4.3}$$

Das berechnete Skalarprodukt bei der Klassifizierung dient uns als Maß für den Abstand zur Hyperebene, somit als Maß für die Anpassung der Hypothese an ein Beispiel.

Die Rechnung macht allerdings auch klar, daß es sich um ein *nicht konsistentes* Verfahren handelt. Das Skalarprodukt $\bar{w}_{k+i}\bar{x}_i$ ist zwar größer als $\bar{w}_k\bar{x}_i < 0$, aber nicht notwendigerweise positiv. Eine weitere Eigenschaft der Lernregel ist ihr *konservatives* Verhalten.

Die Lernrate θ wurde bisher nicht betrachtet. Sie gibt an, wie stark das Perzeptron bei jeder Aktualisierung modifiziert werden soll. So kann z. B. festgelegt werden, daß das Perzeptron sich anfangs stark anpaßt, große Lernschritte in Richtung Lösung geht, und zum Ende hin die Anpassung schwächer wird, das bereits Gelernte gering verändert wird. Der Effekt ist, daß sich die Zielposition des Perzeptrons einpendelt. Diese Methode gewichtet die Trainingsinstanzen ungleich, *„frühe“* Instanzen werden stärker gewertet als *„späte“* Instanzen. Wir beschränken uns auf den Fall der konstanten Lernrate, da dies die Analyse vereinfacht und verständlicher macht.

Betrachten wir in diesem Zusammenhang eingehender die Zeile 9 des Algorithmus, die Aktualisierung des Perzeptrons $\bar{w}_{k+1} = \bar{w}_k + \theta\eta_i\bar{x}_i$. Ist die Lernrate θ konstant und das initiale Perzeptron der Nullvektor, so läßt sich $\bar{w}_{k+1}$ als Linearkombination der Trainingsinstanzen angeben. Für den Beweis behelfen wir uns einer modifizierten Schreibweise. Sei $\bar{w}'_i$ das Perzeptron nach dem i-ten Beispiel, sei $\alpha_i := I[c_{\bar{w}'_i}(\bar{x}_i) \neq \mathcal{L}_i]$ (α_i ist 1, falls $\bar{x}_i$ falsch klassifiziert wurde, sonst 0). Die Gleichung der Aktualisierung kann dann als $\bar{w}'_{i+1} = \bar{w}'_i + \theta\alpha_i\eta_i\bar{x}_i$ angegeben werden ($\bar{w}'_i$ entspricht $\bar{w}_k$ mit $k = 1 + \sum_{j=1}^{i-1}\alpha_j$).

$$\bar{w}'_{i+1} = \sum_{j=1}^{i} \theta\alpha_j\eta_j\bar{x}_j = \theta\sum_{j=1}^{i}\alpha_j\eta_j\bar{x}_j \tag{3.4.4}$$

Wie man anhand der Gleichung sehen kann, verursacht ein konstantes θ lediglich eine Skalierung der Gewichte des Perzeptrons (und des Skalarprodukts bei der Klassifizierung), erfüllt somit keinen Zweck beim Training und kann aus diesem Grund ohne Beschränkung der Allgemeinheit ignoriert bzw. auf 1 gesetzt werden.

An dieser Stelle sei auch die Initialisierung des Perzeptrons erläutert. Im allgemeinen ist es möglich, jeden beliebigen Vektor als Initialvektor zu verwenden. Eine Möglichkeit besteht darin, wie im vorangegangenen Absatz erwähnt, das Perzeptron mit dem Nullvektor zu initialisieren. Diese Option steht im Widerspruch zur geometrischen Definition eines Perzeptrons, da ein Nullvektor keine Hyperebene aufspannen kann. Ein weiteres Problem besteht in der Asymmetrie der Klassifizierfunktion. Erhält ein Perzeptron anfangs nur positive Beispiele, so werden diese fälschlicherweise richtig klassifiziert und somit für die Modellierung des Perzeptrons vollständig ignoriert. Dieser Problematik entgeht man mit der Option, die erste Trainingsinstanz als Initialvektor anzunehmen (mit Vorzeichen je nach Klassenzugehörigkeit, da sonst wieder der Nullvektor entstehen kann). Schließlich besteht noch die Möglichkeit, unabhängig von der Trainingsmenge einen beliebigen Vektor zu verwenden, der entweder immer gleich ist (ein Spezialfall hiervon ist der Nullvektor) oder jedes Mal neu bestimmt wird. Das Hauptproblem bei dieser Option ist, daß die Größenordnung der Trainingsbeispiele nicht berücksichtigt wird. So kann es vorkommen, daß der Normalenvektor sehr lang im Vergleich zu den Eingangsvektoren ist und deshalb nur schwer durch Addition bzw. Subtraktion in die richtige Position gebracht werden kann.

3.5 Konvergenz und Fehlerabschätzung

Eine interessante Frage ist, ob die Perzeptron-Lernregel im Falle einer separierbaren Trainingsmenge in der Lage ist, eine trennende Hyperebene zu finden. Die Eigenschaft eines Lerners, für eine gegebene Trainingsmenge nach endlicher Anzahl Lernschritte eine Hypothese zu finden, die keine Fehler auf den gegebenen Daten verursacht, nennt man *Konvergenz*. Aus der fehlenden Konsistenz ist klar, daß nach einmaligem Durchlaufen der Trainingsmenge keine Konvergenz erreicht werden kann. Wie in Abschnitt 2.3.3 beschrieben, ist es jedoch möglich, bei der Umstellung von inkrementellen auf Batch-Lernen die Trainingsbeispiele mehrmals zu durchlaufen. Jede dieser Iterationen bezeichnet man als eine Epoche. Dafür ist keine Umstellung des Algorithmus notwendig, das Wiederholen der Trainingsfolge bei Erreichen des letzten Beispiels

$$\langle(\bar{x}_1, \mathcal{L}_1), (\bar{x}_2, \mathcal{L}_2), \ldots, (\bar{x}_P, \mathcal{L}_P), (\bar{x}_1, \mathcal{L}_1), \ldots, (\bar{x}_P, \mathcal{L}_P), (\bar{x}_1, \mathcal{L}_1), \ldots\rangle$$

als Eingabe für den Algorithmus hat den gewünschten Effekt ohne zusätzliche Änderungen vornehmen zu müssen.

Es wurde schon früh gezeigt, daß bei einer linear separierbaren Trainingsmenge das Perzeptron nach endlicher Anzahl Trainingsbeispiele konvergiert [Minsky und Papert, 1969]. Es stellt sich allerdings die Frage, ob es sinnvoll ist, den Perzeptron-Algorithmus bis zur Konvergenz trainieren zu lassen (vorausgesetzt es ist bekannt, daß die Daten linear separierbar sind).

Das erste Gegenargument ist die Güte der gefundenen Hyperebene. Der Perzeptron-Algorithmus findet zwar eine gültige Hyperebene, doch nicht notwendigerweise eine gute. Eine gute Hyperebene zeichnet sich dadurch aus, daß sie eine gute Verallgemeinerung der Daten darstellt, somit auch neue, bisher unbekannte Daten gut klassifizieren kann. Die beste Generalisierung für eine linear trennbare Datenmenge ist die *optimale separierende Hyperebene*[3]. Sie ist definiert als die Hyperebene mit maximalem Abstand zum nächsten Punkt bzw. die Hyperebene mit maximalem Rand (engl. margin) zu den Punkten [Cortes und Vapnik, 1995, S. 8]. Es ist sehr unwahrscheinlich, daß ein Perzeptron diese Hyperebene findet. Eine Veranschaulichung von verschiedenen trennenden Hyperebenen im Vergleich zur optimalen Hyperebene findet sich in Abbildung 3.4. Ein *Overfitting* an die Trainingsbeispiele scheint möglich [Baldi u. a., 2003, S. 98f]. Betrachten wir den Fall von nicht ausbalancierten Klassen, d. h. von Klassen mit großen Unterschied in der Anzahl der Elemente. Es ist zu erwarten, daß durch die große Anzahl Aktualisierungen verursacht durch die Punkte der großen Klasse die Hyperebene am Ende sehr nah an der Punktwolke der kleinen Klasse verläuft. Das Beispiel verdeutlicht, daß ein konvergiertes Perzeptron nicht notwendigerweise neue Daten besser klassifiziert als ein nicht bis zur Konvergenz trainiertes Perzeptron.

Ein weiteres Gegenargument ist die Laufzeit. Aus der Konvergenz des Perzeptrons leitet sich ab, daß die Anzahl der Trainingsfehler beschränkt ist. Es ist zu erwarten, daß die Anzahl der Fehler je Epoche asymptotisch abnimmt. Das Perzeptron nähert sich gegen Ende pro Epoche nur geringfügig an die Gesamtzahl der Fehler, die Lernschritte werden kleiner. Nahe an der Konvergenz ist der Nutzen einer erneuten Trainingsepoche gering. Beim Trainieren von künstlichen neuronalen Netzen beispielsweise, wo eine Konvergenz oft gewünscht wird, sind Epochenzahlen im Tausender-Bereich durchaus üblich [siehe z. B. Kuncheva, 2004, S. 85]. Schließlich ist noch zu beachten, daß bei mehr als einer Epoche das Perzeptron die Eigenschaft des inkrementellen Lernens verliert.

[3] engl.: optimal separating hyperplane *oder* maximal margin hyperplane

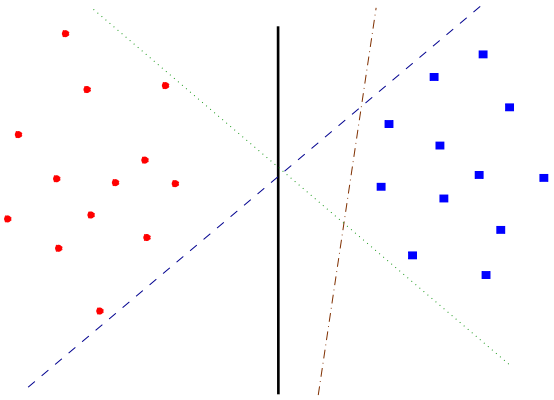


Abbildung 3.4: Trennende Hyperebenen im zweidimensionalen Raum. Die nicht gestrichelte Gerade stellt die optimale trennende Hyperebene dar.

Trotzdem ist es sinnvoll, sich mit der Konvergenz und Anzahl Trainingsfehler bis zu diesem Punkt zu beschäftigen. Zum einen ist die Anzahl der Fehler bis zur Konvergenz ein gutes Maß für die Beurteilung, ob ein Perzeptron im Vergleich zu anderen Lernern eine gegebene Datenmenge gut modellieren kann. Zum anderen bedeutet eine maximale Anzahl Fehler bis zur Konvergenz auch eine Schranke für die Anzahl der Fehler bei z. B. nur einer Epoche.

Wir beschäftigen uns im folgendem deshalb mit einem Beweis für die Konvergenz des Perzeptrons im Falle separierbarer Trainingsdaten und der Berechnung der damit verbundenen maximalen Anzahl der Trainingsfehler. Die hier vorgestellte Herleitung geht auf Analysen von Freund und Schapire [1999] und Bishop [1995, Kap. 3.5.3] zurück, die wiederum auf Arbeiten von beispielsweise Block [1962] und Novikov [1962] beruhen.

Sei $\langle(\bar{x}_1,\mathcal{L}_1),(\bar{x}_2,\mathcal{L}_2),\ldots,(\bar{x}_P,\mathcal{L}_P),(\bar{x}_{P+1},\mathcal{L}_{P+1}),\ldots\rangle$ eine unendliche, zyklische Folge von Trainingsbeispielen mit $\bar{x}_i = \bar{x}_{i \bmod P+1}$. Sei R die maximale Norm der Instanzen $R \geq |\bar{x}_i|$. Vektor $\bar{v}$ beschreibe eine beliebige trennende Hyperebene. Es gelte $|\bar{v}| = 1$ und der minimale Abstand eines Punktes zur Hyperebene sei gegeben durch $\mu \leq \eta_i(\bar{v}\cdot\bar{x}_i)\,,\ \forall i$. Sei $\bar{w}_k$ das Perzeptron nach der $(k-1)$-ten Aktualisierung (siehe Abbildung 3.3). $(k-1)$ ist somit die Anzahl der Trainingsfehler. Geschieht der k-te Fehler auf $\bar{x}_i$, so bedeutet dies $\eta_i(\bar{w}_k\cdot\bar{x}_i)\leq 0$. Anschließend wird das Perzeptron aktualisiert: $\bar{w}_{k+1} = \bar{w}_k + \eta_i\bar{x}_i$. Wir gehen ohne Beschränkung der Allgemeinheit vom Nullvektor als initialen Perzeptron aus.

Für das Skalarprodukt des Perzeptrons mit $\bar{v}$ gilt

$$\bar{w}_{k+1}\bar{v} = \bar{w}_k\bar{v} + \eta_i(\bar{v}\bar{x}_i) \geq \bar{w}_k\bar{v} + \mu \tag{3.5.1}$$

Wir lösen bis $k=1$ auf und erhalten $\bar{w}_{k+1}\bar{v} \geq k\mu$. Wir haben damit eine untere Schranke für $\bar{w}_{k+1}\bar{v}$ angegeben, die linear mit k wächst. Wir suchen nun eine obere Schranke. Für $\bar{w}^2_{k+1}$ (bzw. $|\bar{w}_{k+1}|^2$) gilt

$$\bar{w}^2_{k+1} = (\bar{w}_k + \eta_i\bar{x}_i)^2 = \bar{w}^2_k + 2\eta_i(\bar{w}_k\cdot\bar{x}_i) + \bar{x}^2_i \leq \bar{w}^2_k + R^2 \tag{3.5.2}$$

da $|\bar{x}_i| \leq R$ und $\eta_i(\bar{w}_k\cdot\bar{x}_i)\leq 0$. Daraus ergibt sich durch Auflösung der Rekursion $\bar{w}^2_{k+1} \leq kR^2$. Nach dem Ziehen der Wurzel erhalten wir $|\bar{w}_{k+1}| \leq \sqrt{k}R$. Aus $|\bar{w}_{k+1}||\bar{v}| \geq |\bar{w}_{k+1}\bar{v}|$ und $|\bar{v}|$ können

wir nun eine obere Schranke für $\bar{w}_{k+1}\bar{v}$ angeben

$$\begin{aligned}\sqrt{k}R &\geq |\bar{w}_{k+1}||\bar{v}| \\ &\geq |\bar{w}_{k+1} \cdot \bar{v}| \;\geq\; k\mu\end{aligned} \tag{3.5.3}$$

Die obere Schranke wächst langsamer als die untere, womit ab einem endlichen k kein Perzeptron mehr existieren kann, der beide Schranken erfüllt. Da k die Anzahl der Aktualisierungen widerspiegelt und nach oben beschränkt sein muss, ist somit bewiesen, daß der Perzeptron-Algorithmus auf linear separierbaren Daten konvergiert. Durch Auflösung der Ungleichung $k\mu \leq \sqrt{k}R$ nach k läßt sich zudem eine obere Schranke für die Anzahl der Fehler k angeben

$$k \leq \left(\frac{R}{\mu}\right)^2 \tag{3.5.4}$$

Die Schranke für k ist am engsten für eine Hyperebene mit maximalem Margin μ, also für die optimal separierende Hyperebene. Eine geringe Anzahl Trainingsfehler bis zur Konvergenz deutet demnach darauf hin, daß das Perzeptron eine gute Verallgemeinerung der Trainingsdaten darstellt. Ist ein gültiges $\bar{v}$ für eine Beispielmenge bekannt, so läßt sich die maximale Anzahl der Trainingsfehler eines Perzeptrons auf diesen Daten angeben. Eine niedrige Schranke deutet auf die gute Fähigkeit des Perzeptron-Algorithmus hin, eine gegebene Datenmenge verallgemeinern zu können.

Diese letzte Feststellung führt uns zu der wichtigen Frage, wovon die Fähigkeit des Perzeptrons abhängt, eine Datenmenge gut verallgemeinern zu können. Durch das Wissen dieser Beziehung kann im Vorhinein für einen bestimmten Datentyp entschieden werden, ob das Perzeptron dafür geeignet ist. Skalieren wir die Trainingsbeispiele mit $1/\kappa$. Wir erhalten als Schranke $\frac{1}{\mu'^2}$ mit $\mu' = \mu/\kappa$. Wir erkennen, daß die Anzahl der Fehler von dem Freiraum zwischen den Punktmengen der positiven und negativen Beispiele abhängt. Beträgt dieser Freiraum in einem Datensatz beispielsweise an der engsten Stelle 1% des Durchmessers der Ausbreitung der Punkte im Raum, so beträgt die maximale Anzahl Fehler, die ein Perzeptron auf diesen Daten machen wird, 100.

Diese Aussage läßt vermuten, daß die Anzahl der Attribute keine großen Auswirkungen auf die Leistung des Perzeptrons hat, ein Perzeptron demzufolge in hoch dimensionalen Räumen gute Ergebnisse liefern könnte, also auch für die Textklassifizierung geeignet ist [Freund und Schapire, 1999].

3.6 Lineare Separierbarkeit von Datenmengen

Die vorgestellte Schranke stellt ein nützliches Werkzeug für die Bewertung des Perzeptrons dar. Allerdings sind die Aussagen nur gültig, wenn die Trainingsdaten linear separierbar sind. Es ist demnach interessant herauszufinden, ob sich aus den Merkmalen der Datenmenge im Vorhinein eine Aussage treffen läßt, ob die Daten separierbar sind. Das Thema der linearen Separierbarkeit wurde ausführlich von Vapnik untersucht [u. a. Vapnik, 2000]. Die für uns relevanten Aussagen seien im folgendem erörtert.

Sei $Q_\alpha(\bar{x}) \in \mathcal{M}, |\mathcal{M}| = 2$ eine Funktionsschar von binären Funktionen abhängig von einer Menge von Parametern α, $\bar{x}$ bezeichne einen Punkt im Raum. Sei $\{Q_\alpha \mid \forall\alpha\}$ die Menge aller möglichen Funktionen Q_α. Sei $\{\bar{x}_1, \ldots, \bar{x}_i\}$ eine Punktmenge mit einer einfachen binären Klassifikation.

Wenn, gegeben eine Punktmenge $\{\bar{x}_1, \ldots, \bar{x}_i\}$, zu jeder der 2^i möglichen Klassifikationen dieser Punktmenge eine Funktion Q_α aus $\{Q_\alpha \mid \forall\alpha\}$ angegeben werden kann, die die Punkte korrekt klassifiziert (trennt), dann, heißt es, wird die Punktmenge $\{\bar{x}_1, \ldots, \bar{x}_i\}$ von $\{Q_\alpha\}$ *„zerschmettert“* (engl. *shatter*).

Nehmen wir als Q die Klassifizierfunktion c des Perzeptrons aus Gleichung (3.3.1) und als α den Normalenvektor $\bar{w}$ und den Schwellwert ϑ. Betrachten wird das Beispiel auf Seite 34, wir bewegen uns wieder im zweidimensionalen euklidischen Raum. Eine Punktmenge mit zwei beliebigen Punkten kann durch $\{c_{\bar{w},\vartheta}\}$ zerschmettert werden, da sich zu jeder Klassifikation in jedem Fall eine Funktion $c_{\bar{w},\vartheta}$ (eine Gerade) finden läßt, die beide Punkte korrekt klassifiziert. Drei beliebige nicht auf einer Gerade liegende Punkte werden auch von $\{c_{\bar{w},\vartheta}\}$ zerschmettert. Drei in Linie liegende Punkte können wiederum nicht mehr zerschmettert werden. Vier Punkte können in keinem Fall mehr zerschmettert werden, da keine vier Punkte angegeben werden können, die bei allen möglichen Klassifikationen von $\{c_{\bar{w},\vartheta}\}$ abgedeckt werden, obwohl einzelne Kombinationen separiert werden können.

Die *VC-Dimension*[4] gibt das Vermögen einer Funktionsmenge $\{Q_\alpha\}$ an, eine Problemklasse einer gewissen Komplexität modellieren zu können. Sie ist definiert als die maximale Anzahl Punkte, die von $\{Q_\alpha\}$ zerschmettert werden kann. Bei unserem zweidimensionalen Perzeptron beträgt die VC-Dimension demzufolge 3. Wie bereits am Beispiel in der Ebene gesehen, garantiert eine VC-Dimension von h nicht, daß h Punkte durch ein Q korrekt klassifiziert werden können.

Ein N-dimensionales Perzeptron mit Schwellwert (die Dimension bezieht sich auf den Normalenvektor) besitzt eine VC-Dimension von $N+1$ [Vapnik, 2000, S. 81]. Es kann somit höchstens $N+1$ Punkte mit beliebiger Klassifikation korrekt klassifizieren. Aber es sagt sowohl nichts darüber aus, ob eine gegebene Trainingsmenge mit mehr als $N+1$ Punkten klassifiziert werden kann , also auch, ob eine gegebene Trainingsmenge mit $N+1$ oder weniger Punkten klassifiziert werden kann (man erinnere sich an die klassifizierbaren 4-Punkt-Kombinationen und die unklassifizierbaren 3 Punkte auf einer Gerade in der euklidischen Ebene). Man kann aber sagen, daß sich mit wachsender Anzahl Punkte die Anzahl der Punktkombinationen reduziert, die klassifiziert werden können.

Bisher können wir bei gegebener Trainingsmenge keine definitive Aussage darüber treffen, ob sich die Beispiele linear separieren lassen, nur eine Vermutung. Die folgende sehr anschauliche Definition der linearen Separierbarkeit ist uns dabei nützlich. Seien $\mathcal{A}$ und $\mathcal{B}$ zwei Punktmengen in $\mathbb{R}^N$, so sind diese durch eine Hyperebene trennbar, wenn die Schnittmenge ihrer konvexen Hüllen leer ist. Die konvexe Hülle einer Punktmenge $\{\bar{x}_1, \ldots, \bar{x}_i\}$ sei dabei definiert als $\{\bar{x} \mid \bar{x} = \sum_{j=1}^{i} \beta_j \bar{x}_j, \beta_j \geq 0\}$. Ein Beweis für den Satz findet sich in [Burges, 1998, S. 36].

Auch diese Vorgehensweise ist in der Praxis ungeeignet, um im Vorhinein zu bestimmen, ob eine Trainingsmenge linear trennbar ist. Es müßte für jedes Trainingsbeispiel ausgerechnet werden, ob es sich in der konvexen Klasse der Gegenseite befindet.

Aufgrund der geringen Aussagekraft für die lineare Separierbarkeit einer Datenmenge der vorgestellten Analyse erscheint es sinnvoll, sich mit dem Fall zu beschäftigen, daß sich die Trainingsdaten nicht separieren lassen.

[4] Vapnik-und-Chervonenkis-Dimension

3.7 Fehlerabschätzung bei nicht separierbaren Daten

Wir betrachten nun den Fall, daß sich eine Datenmenge nicht linear separieren läßt. Dies kann sowohl dann der Fall sein, wenn sich bei einer binären Datenmenge aufgrund der Beschaffenheit und Komplexität der Domäne der Objekte bei Darstellung im Vektorraummodell die Beziehung zwischen den beiden Punktmengen nicht linear beschreiben läßt, als auch, wenn ein Rauschen in den Daten eine Separierbarkeit verhindert (ein Objekt kann beispielsweise zweifach in einer Datenmenge enthalten sein und aufgrund einer Fehlklassifikation unterschiedliche Zuordnungen besitzen).

Auch für diesen untrennbaren Fall läßt sich eine Schranke für die Anzahl der Trainingsfehler angeben. Folgender Satz geht auf [Freund und Schapire, 1999] zurück.

Sei $\langle(\bar{x}_1,\mathcal{L}_1),(\bar{x}_2,\mathcal{L}_2),\ldots,(\bar{x}_P,\mathcal{L}_P)\rangle$ eine Trainingsmenge mit $R \geq |\bar{x}_i|$. Sei $\bar{v}$ ein beliebiger Vektor mit $|\bar{v}| = 1$ und sei $\mu > 0$. Sei die Abweichung eines Beispiels $\mathcal{X}[i]$ definiert als

$$d_i = \max\{0, \mu - \eta_i(\bar{v} \cdot \bar{x}_i)\} \tag{3.7.1}$$

und sei $D := \sqrt{\sum_{i=1}^{P} d_i^2}$. Dann ist die Anzahl der Fehler des Perzeptron-Lernregel auf dieser Trainingsfolge beschränkt durch

$$\left(\frac{R+D}{\mu}\right)^2 \tag{3.7.2}$$

Für den Beweis wird das Problem in einen höher dimensionalen Raum verlagert, wo die Datenmenge separierbar ist, und die Schranke aus Gleichung (3.5.4) angewandt. Ein ausführlicher Beweis findet sich in [Freund und Schapire, 1999, S. 5f]. Eine engere Schranke $(\frac{R+\sqrt{\mu D}}{\mu})^2$ wurde kürzlich von Shalev-Shwartz und Singer [2005] vorgestellt.

Die Abweichung d_i ist als der negative Abstand zu einer Hyperebene zu verstehen, die im Abstand μ vor der durch $\bar{v}$ aufgespannten Hyperebene liegt. Die Hyperebene erhält sozusagen einen Schutzzone, in der sich die Punkte nicht aufhalten dürfen (eine solche Hyperebene wird von Vapnik [2000, S. 132] *A-margin hyperplane* genannt). Eine gute Verallgemeinerung der Daten ist eine Hyperebene, die die Abweichung D möglichst minimiert. Die direkte Folge davon ist, daß ein Perzeptron mit geringer Anzahl Trainingsfehler eine gute Verallgemeinerung der Trainingsdaten darstellt.

Wir stellen fest, daß es in der Praxis nicht von großer Bedeutung ist, ob die Daten linear trennbar sind. Man wird nämlich auch bei vorhandener Separierbarkeit aus den anfangs in Abschnitt 3.5 genannten Gründen die Epochenzahl beschränken, so daß das Ergebnis wie bei dem unseparierbaren Fall eine möglichst gute Anpassung an die Daten darstellen wird, aber kein perfektes Modell sein wird.

3.8 Varianten

Das Perzeptron ist ein relativ altes Lernverfahren und war deshalb oft Gegenstand der Forschung. Auch aufgrund der Einfachheit des Algorithmus wurden im Laufe der Zeit viele Varianten entwickelt. Für diese Arbeit von besonderem Interesse ist die Variante des Perzeptrons ohne Schwellwert, da das Verfahren hauptsächlich in dieser Form zum Einsatz kommt.

3.8.1 Perzeptron ohne Schwellwert

In Gleichung (3.3.1) wurde das Perzeptron mit Schwellwert vorgestellt, in Gleichung (3.3.5) wurde der unbequeme Schwellwert durch Erweiterung des Vektorraum in die letzte Dimension integriert, dafür allerdings auch der gesamte Algorithmus in einen anderen Raum als den ursprünglichen Raum der Objekte verlagert. In der nun vorgestellten Variante wird auf die Dimensionserweiterung und den Schwellwert verzichtet. Sie wird deshalb als *Perzeptron ohne Schwellwert*[5] bezeichnet. In der Forschung und Literatur ist dies die bevorzugte Variante, das Perzeptron wird oft allgemein in dieser Form beschrieben[6]. Die resultierende Gleichung ist in (3.8.1) dargestellt. Die Perzeptron-Lernregel bleibt von dieser Änderung unberührt.

$$
\begin{aligned}
&c_{\bar{\mathrm{w}}} : \mathcal{X} \to 2^{\mathcal{Y}} \\
&c_{\bar{\mathrm{w}}}(\bar{\mathrm{x}}) = \begin{cases} \{y_0\} & \text{, falls } \sum_{i=0}^{N} a_i w_i = \bar{\mathrm{x}}\bar{\mathrm{w}} \geq 0 \\ \{y_1\} & \text{, falls } \sum_{i=0}^{N} a_i w_i = \bar{\mathrm{x}}\bar{\mathrm{w}} < 0 \end{cases}
\end{aligned}
\tag{3.8.1}
$$

Der Vorteil ist die schnelle Handhabe und die direkte geometrische und mathematische Interpretation. Das Skalarprodukt mit dem Normalenvektor gibt direkt den Winkel zur Hyperebene an ohne Umwege über die Verrechnung mit dem Schwellwert oder den mit der Dimensionserweiterung einhergehenden Interpretationsschwierigkeiten. Der Nachteil ist der Verlust eines Freiheitsgrades der Hyperebene, welche nun immer durch den Ursprung verläuft. Der Hypothesenraum ist kleiner. Dies hat Auswirkungen auf die Aussage, ob ein Perzeptron zwei Punktmengen trennen kann. Als Beispiel sei kurz die Trennung von zwei Punkten im zweidimensionalen Raum erwähnt. Während sich immer eine Gerade finden läßt, die zwei beliebige Punkte trennen kann, trifft dies bei im Falle von Geraden durch den Ursprung nicht mehr zu. Die Definition der Trennbarkeit von zwei Mengen durch ein Perzeptron in Gleichung (3.3.3) ist verständlicherweise hinsichtlich des Schwellwerts zu verändern, es gilt $\vartheta = 0$.

Eine interessante Feststellung ist allerdings, daß offensichtlich die VC-Dimension des Perzeptrons ohne Schwellwert im Vergleich zum Perzeptron mit Schwellwert um eins reduziert ist. Die VC-Dimension von einem N-dimensionalen Perzeptron ohne Schwellwert ist somit N. Das läßt stark vermuten, daß bei großen N der Unterschied in der Separierbarkeit gegenüber der Variante mit Schwellwert nicht ins Gewicht fällt. Diese Variante stellt demzufolge bei Problemstellungen mit einer hohen Dimensionalität wie z. B. der Textklassifizierung keine bedeutenden Einschränkungen dar.

3.8.2 Andere Varianten

Die *Delta-Regel*[7] oder das *Gradient-Descent-Verfahren* ist eine Familie von Varianten, die auch bei nicht linear separierbaren Daten gegen ein Ergebnis konvergiert [Mitchell, 1997, S. 89]. Andere Namen sind *LMS, Adaline-* oder *Widrow-Hoff-Regel*. Bei der Delta-Regel wird eine Funktion E definiert, die jeder möglichen Hypothese ein Maß für den Fehler auf der Trainingsmenge zuordnet, folglich $\mathbb{R}^N \to [0,n), n < \infty$. Die Funktion beschreibt einen N-dimensionalen elliptisch paraboloiden

[5] engl.: unthresholded perceptron
[6] vgl. dazu [Crammer und Singer, 2003], [Freund und Schapire, 1999], [Sebastiani, 2002, S. 25], [Mitchell, 1997, S. 89]
[7] engl.: delta rule

Raum in $\mathbb{R}^{N+1}$, bildlich vorgestellt eine nach unten eingedrückte Hyperebene. Es existiert demnach ein Punkt, wo die Funktion einen globalen Tiefpunkt hat, also der Fehler minimal ist, unabhängig davon, ob die Daten linear separierbar sind. Der Algorithmus startet nun mit einem beliebigen Vektor, berechnet den Fehler, indem die Trainingsmenge einmal durchlaufen wird, und berechnet anhand einer einfachen Rechenvorschrift den Gradienten-Vektor an dieser Stelle des Paraboloiden. Der negative Gradientenvektor zeigt dabei in Richtung des steilsten Abstiegs. Die Hypothese wird in diese Richtung verschoben und dieser Vorgang so oft wiederholt, bis der tiefste Punkt erreicht ist.

Eine Möglichkeit, eine nicht linear separierbare Datenmenge in eine linear separierbare Datenmenge zu verwandeln, bilden *Kernel-Funktionen* [Cortes und Vapnik, 1995]. Bei dieser Methode wird der Instanzenraum um weitere Dimensionen erweitert (ähnlich der Erweiterung des Raumes aus Gleichung (3.3.5), um den Schwellwert zu integrieren). Die neuen Komponenten werden nicht-linear aufgrund der gewählten Kernel-Funktion aus den vorhandenen Attributen berechnet. Bei geschickter Wahl der Kernel-Funktion und Anzahl zusätzlicher Dimensionen wird das Problem in ein einfaches, linear-lösbares transformiert. Der Perzeptron-Algorithmus ist durch dieses Verfahren leicht auf nicht linear separierbare Daten zu erweitern. Bestimmte Kernel-Funktionen lassen sich im Zusammenhang mit dem Skalarprodukt sehr effizient berechnen, so daß die Laufzeit des Perzeptrons kaum beeinträchtigt wird. Einige Beispiele für Perzeptron-basierte Verfahren mit Kernel-Funktionen sind das *Voted-Perceptron* [Freund und Schapire, 1999] und das *Kernel-Adatron* [Frieß u. a., 1998].

Erwähnenswert sind in diesem Zusammenhang *Support-Vektor-Maschinen* [Cortes und Vapnik, 1995, Burges, 1998]. Dieses Verfahren konstruiert anhand der Trainingsbeispiele die optimale separierende Hyperebene. Da es dabei auf die Existenz separierender Hyperebenen angewiesen ist, kommen fast ausschließlich Kernel-Funktionen zum Einsatz. Der große Vorteil ist natürlich, daß die gefundene Hyperebene eine sehr gute Verallgemeinerung der Daten darstellt (die best-mögliche anhand der gegebenen Daten). Der Nachteil ist, daß das Verfahren quadratische Zeit beansprucht.

Ein kürzlich von Shalev-Shwartz und Singer [2005] vorgestellter Algorithmus, genannt *Ballseptron*, versucht die Schwäche des Perzeptrons der geringen Generalisierung durch die gefundenen Lösungen zu bekämpfen. Es wird eine Kugel um jeden Punkt der Trainingsmenge definiert, der die Hyperebene, wie auch der Punkt selbst nicht überschreiten darf. Im Falle der Falschklassifizierung eines Beispiels ist die Funktionsweise identisch mit der des Perzeptrons. Dringt die Kugel eines (richtig klassifizierten) Punktes in die gegenüberliegende Hälfte ein, so wird die Hyperebene gerade so gedreht, daß die Kugel die Ebene nicht mehr berührt. Dadurch wird eine Hyperebene gefunden, die als Abstand zu den Punktmengen mindestens den Durchmesser der Kreise besitzt. Diese *A-margin hyperplanes* wurden bereits im Zusammenhang mit der Fehlerabschätzung bei nicht separierbaren Daten erwähnt (Abschnitt 3.7). Ein Ballseptron findet Hyperebenen, die im Stande sind, Daten besser zu verallgemeinern als die Lösungen eines Perzeptrons. Durch die Festlegung eines Mindestabstands wird allerdings die Fähigkeit beschränkt, die Daten zu trennen [Vapnik, 2000, S. 132]. Wird der Durchmesser der Kugeln größer gewählt als der Freiraum zwischen den Punktwolken, kann keine Lösung mehr gefunden werden.

3.9 Komplexitätsanalyse

Im nachfolgenden Abschnitt untersuchen wir die Anforderungen an Laufzeit und Speicherverbrauch des Perzeptrons. Zur Erinnerung seien nochmals die wichtigsten Variablen für die Analyse genannt: N gibt die Anzahl der Attribute respektive Dimensionen an, P die Anzahl der Beispiele. Zusätzlich sei M die durchschnittliche Anzahl verwendeter Attribute in einer Instanz, folglich die Anzahl der Attribute ungleich Null. Wir gehen von dem Perzeptron ohne Schwellwert aus. Für die Variante mit Schwellwert ergeben sich nur vernachlässigbare Abweichungen.

Beginnen wir mit dem Speicherverbrauch. Für das Training muss mindestens ein Perzeptron gespeichert werden, also ein N-dimensionaler Vektor. Dies veranschlagt einen Speicherverbrauch von $O(N)$. Bei der inkrementellen Version ist die Rechnung damit beendet. Werden allerdings mehrere Epochen gelernt und dies geschieht für den Lernalgorithmus nicht transparent, d. h. daß der Algorithmus selbst die Trainingsmenge zum wiederholten Durchlaufen speichern muss, so muss dafür zusätzlich Speicher berechnet werden. Das Perzeptron selbst benötigt jedoch weder zum Lernen noch zum Speichern der Hypothese mehr als $O(N)$ Speicher.

Wir definieren ein Attribut einer Instanz als belegt, wenn dieser ungleich Null ist. Sind viele Attribute unbelegt, der Attributvektor spärlich[8] belegt, so lohnt sich eine Index-Speicherung der Vektoren. Sei m die Anzahl der belegten Attribute, dann wird bei dieser Speicherorganisation in einem m-großen Vektor die Indexe der belegten Attribute gespeichert. In einem weiteren m-großen Vektor werden die ursprünglichen Werte der unbelegten Attribute gespeichert. Enthält beispielsweise die erste Zelle im ersten Vektor die Zahl 11, so bedeutet dies, daß der Wert in der ersten Zelle des zweiten Vektors der Wert des elften Attributs ist und die Attribute von 1 bis 10 leer sind. So einen Vektor nennt man auch *sparse vector*.

Nehmen wir für die Speicherung der Indexe den gleichen Speicherbedarf wie für die Speicherung der Attribute an, so ist bereits ab $m < N/2$ eine Speicherung in dieser Form sinnvoll. Für die zusätzliche Laufzeit beim Aufsuchen ist zu beachten, daß eine sequentielle Auslesung der Attribute keinen signifikanten Mehraufwand bedeutet (durch geschickte Registerverwaltung kann sogar Zeit gespart werden), doch beim wahlfreien Zugriff die Laufzeit um den Faktor m zunimmt. Dies betrifft aber nicht den Perzeptron-Algorithmus, da kein wahlfreier Zugriff nötig ist. Im Übrigen ist eine Speicherung des Perzeptronvektors als Sparse-Vektor nicht sinnvoll, da mit sehr hoher Wahrscheinlichkeit $m > N/2$ ist.

Entscheidet man sich für die effektivere Speicherart (bei $M < N/2$), so ist der Speicherverbrauch für die Trainingsmenge $2MP$ Speichereinheiten, sonst NP. Diesen Betrag rechnet man im Normalfall nicht dem verwendetem Lerner zu, es sei denn, er benötigt für das Lernen oder für das Klassifizieren die gesamte Trainingsmenge im Speicher. Der Speicherverbrauch für die Hypothese des Perzeptrons und für die Berechnungen beträgt somit lediglich $O(N)$ Speichereinheiten sowohl für das Training wie für das Testen.

Beim Lernen führt der Algorithmus pro Instanz eine Klassifizierung durch und gegebenenfalls eine Aktualisierung. Eine Klassifizierung besteht aus der Berechnung eines Skalarproduktes und eines effektiven Null-Vergleichs. Beim Skalarprodukt werden N Multiplikationen und $N-1$ Additionen ausgeführt ($\sum_{i=1}^{N} a_i w_i$). Die Aktualisierung setzt sich im ungünstigsten Fall (Lernrate $\eta \neq 1$) aus

[8] engl.: sparse

N Multiplikationen und N Additionen zusammen (N-mal $w_i \leftarrow w_i + \eta a_i$). Die Aktualisierung unterscheidet sich in ihrer Komplexität nicht sehr von dem Skalarprodukt. Wir führen deshalb das Skalarprodukt Π als Maßeinheit für die Laufzeit ein. Es gilt demzufolge

$$O(\mathit{Klassifizierung}) = O(\mathit{Aktualisierung}) = O(1)\,\Pi \tag{3.9.1}$$

Eine Laufzeit beispielsweise von $n\,\Pi$ bedeutet demnach, daß das Verfahren n Skalarprodukt-Operationen benötigt. Wenn klar ist, daß Π-Operationen gemeint sind, wird die Einheit Π weggelassen.

Im günstigsten Fall, d. h. wenn das Perzeptron das Trainingsbeispiel richtig klassifiziert, beträgt der Rechenaufwand 1 Π. Im schlimmsten Fall kommt noch eine Aktualisierung hinzu, was summiert 2 Π ergibt. Sei $\hat{\delta}_{IsErr}(c_{\mathcal{S}}, \mathcal{S})$ der Trainingsfehler auf Basis der Loss-Funktion δ_{IsErr} (siehe Abschnitt 2.3.2), dann wurde das Perzeptron beim Training $\hat{\delta}_{IsErr}P$ Mal aktualisiert. Die Anzahl der Skalarprodukt-Operationen auf den Trainingsdaten beträgt somit $(1 + \hat{\delta}_{IsErr})P$. Wir gehen von einer linearen Laufzeit in der Anzahl der Trainingsbeispiele für inkrementelle Algorithmen aus und geben deshalb die Anzahl der Operationen je Trainingsbeispiel an: $(1 + \hat{\delta}_{IsErr})\,\Pi$. Im günstigsten Fall klassifiziert das Perzeptron die Daten perfekt und $\hat{\delta}_{isErr}$ ist somit Null. Wie in Abschnitt 3.5 und 3.7 untersucht, läßt sich für den Fehler des Perzeptrons eine obere Schranke angeben. In allen Fällen gilt jedoch $\hat{\delta}_{IsErr} \leq 1$. Für die Anzahl der Operationen für das Training des Perzeptrons gilt somit nach der O-Notation $\Omega(1) = (1 + \hat{\delta}_{IsErr}) = O(2)$. Wie man leicht einsehen kann, beträgt die Laufzeit für das Testen $O(1)$ je Beispiel.

Wie schon bei der Betrachtung des Speicherverbrauch, müssen wir uns mit der Verwendung von Sparse-Vektoren beschäftigen. Bei der Rechnung mit einem Sparse-Vektor reduziert sich die Anzahl der Operationen, da nur die Komponenten berücksichtigt werden müssen, die ungleich Null sind. Betrachten wir das Skalarprodukt eines Perzeptrons mit einem Sparse-Vektor. Wenn man davon ausgeht, daß ein Perzeptron voll belegt ist, so beträgt die Anzahl der Operationen lediglich m Multiplikationen und $m-1$ Additionen. Die erwartete Laufzeit eines Skalarprodukts beträgt somit $\Omega(0) = 2M = O(2N)$ Basisoperationen. An dieser Stelle ist allerdings zu berücksichtigen, daß auch bei der vollständigen Speicherung der Vektoren eine effiziente Berechnung möglich ist. Durch Testen auf Null vor der Multiplikation jeder Komponente kann die teure Multiplikation verhindert werden. Es fallen somit im Vergleich zum Sparse-Vektor zusätzlich N Vergleiche an. Besonders bei sehr unbelegten Instanzen (kleines Verhältnis M/N), wie es beispielsweise bei der Textklassifizierung anzutreffen ist, zahlt sich die Verwendung von Sparse-Vektoren hinsichtlich der Laufzeit aus. Für die Aktualisierung des Perzeptrons kann weiterhin die Laufzeit mit einem Π angegeben werden, da die Reduktion der Anzahl der Operationen im gleichen Maße anfällt.

Die Komplexität beträgt zusammengefaßt für das Training $O(N)$ im Speicherverbrauch und $O(1)$ Skalarprodukte oder $O(N)$ Basisoperationen in der Laufzeit. Für das Testen gilt $O(N)$ respektive $O(1)\,\Pi$. Die eingeführte Einheit Π bezeichnet eine Skalarprodukt-Operation und besitzt die Komplexität $O(2N)$. Der Einsatz von Sparse-Vektoren zahlt sich vom Standpunkt der Speicherung ab etwa einer durchschnittlichen Halbbelegung der Attibutvektoren aus. Eine Zusammenstellung und ein Vergleich finden sich auf Tafel 5.1 (S. 70).

3.10 Einsatz bei der Textklassifizierung

Mit dem Perzeptron wurde ein sehr einfacher und schneller Algorithmus vorgestellt. Im Hinblick auf die für diese Arbeit interessante Textklassifizierung wurde bereits in Abschnitt 3.5 bemerkt,

daß die Unabhängigkeit des Trainingsfehlers von der Dimensionalität auf eine gute Eignung für diese Problemklasse hindeuten. Ausgehend von den Betrachtungen zur VC-Dimension läßt die hohe Dimensionalität des Raumes bei Texten sogar auf eine allgemein gute lineare Separierbarkeit von Textdaten und damit auf eine gute Klassifizierbarkeit durch das Perzeptron-Verfahren schließen. In Bezug auf die Laufzeit sind die Untersuchungen im Zusammenhang mit Sparse-Vektoren vielversprechend.

Ergebnisse beispielsweise aus [Ng u. a., 1997] und [Dagan u. a., 1997] bestätigen zum Teil die Annahme der guten Tauglichkeit für Text. Bei diesen Versuchen kommen modifizierte Versionen oder auf dem Perzeptron basierte Verfahren zum Einsatz, die allesamt besser abschneiden als erprobte, gute Textlernalgorithmen wie dem *RIPPER*, *Sleeping Experts* [Cohen und Singer, 1996], *SWAP* und der beim Information Retrieval beliebte *Rocchio*-Algorithmus. In mehreren Untersuchungen hat sich zusätzlich gezeigt, daß die Verwendung von Kernel-Methoden bei der Textklassifizierung mit SV-Maschinen (siehe Abschnitt 3.8.2) keinen Vorteil bringen oder sogar schlechtere Ergebnisse liefern [Joachims, 1998, Yang und Liu, 1999]. Da der SVM-Algorithmus weitgehend gute Leistungen bei der Klassifizierung von Text erbringt, ist dies ein weiteres Indiz dafür, daß Textdaten gut linear separierbar sind.

Einen guten Vergleich der Ergebnisse verschiedener Algorithmen auf Textdaten bietet Sebastiani [2002, S. 38], allerdings ohne das Perzeptron miteinzubeziehen. Das liegt mitunter daran, daß das Perzeptron lange Zeit als *„Underperformer“* bei hoch-dimensionalen Daten galt und deshalb nicht oft untersucht wurde.

4 Multiclass Multilabel Perceptron

Der *Multiclass-Multilabel-Perceptron*-Algorithmus (*MMP*), stellt eine Adaptation des Perzeptron-Algorithmus auf das Multilabel-Problem dar. Er wurde 2002 von Crammer und Singer [2002, 2003] entwickelt. Inspiriert für die Verwendung des Perzeptrons als Basis wurden die Autoren durch die guten Resultate des Perzeptrons und seiner Varianten, beispielsweise des *Voted-Perzeptrons* von Freund und Schapire [1999]. Auch der MMP-Algorithmus konnte bei den Versuchen von Crammer und Singer gute Leistungen erbringen. Als besondere Merkmale des MMP sind zudem seine einfache Funktionsweise und seine Effizienz auszumachen.

Ein Perzeptron ist ein binärer Lerner und kann somit nur die Zuordnung zu zwei Klassen lernen. Das MMP wurde entwickelt, um Multilabel-Klassifikationen zu lernen. Das heißt, die Anzahl der möglichen Klassen ist nicht begrenzt und ein Objekt kann zu mehr als nur einer Klasse zugeordnet sein (vgl. Abschnitt 2.2). Die Vorhersage des MMP entspricht nicht direkt einer Menge von relevanten Klassen, sondern einem Klassenranking, der Algorithmus gehört somit zu den *Klassen-Ranking-Algorithmen* (siehe Abschnitt 2.4.1).

Es existieren Techniken, um ein Multiklassen-Problem auf mehrere binäre Probleme zurückzuführen, damit auch ein binärer Algorithmus Multiklassen-Klassifikationen lernen kann. Eine sehr simple und deshalb oft eingesetzte Methode ist die *One-against-all*-Methode. Man kann den MMP-Algorithmus als eine Variante des One-against-all-Verfahrens ansehen. Um dem Leser die Funktionsweise des MMP näher zu bringen, beschäftigt sich der nächste Abschnitt deshalb mit dem One-against-all-Verfahren unter Verwendung des Perzeptrons.

4.1 One-against-all mit Perzeptrons

Das One-against-all-Verfahren gehört zu den sogenannten *class binarization* oder *decomposing* Verfahren, die ein Multiklassen-Problem auf mehrere Zweiklassen-Probleme abbilden auf eine Weise, daß die Vorhersagen der auf den Zweiklassen-Problemen basierenden Klassifizierer sich zu einer Gesamtvorhersage für das Multiklassen-Problem zusammenfügen lassen. Diese Technik wird in der vorliegenden Arbeit auch *binäre Zurückführung* genannt. Die auf den binären Daten angewandten Lernverfahren heißen *Basislerner* [Fürnkranz, 2002].

Die One-against-all, *One-against-the-rest* oder auch *unordered class binarization* Methode zerteilt ein Multiklassenproblem mit K Klassen in K Zweiklassen-Probleme. Dabei deklariert es für das k-te Zweiklassen-Problem alle Beispiele, die der k-ten Klasse zugeordnet sind, als positiv und alle restlichen als negativ. Jeder Basislerner ist demnach einer Klasse zugeordnet, die es von den anderen zu unterscheiden gilt. Ein Basislerner wird aus diesem Grund auch Prototyp einer Klasse genannt. Die Definition kann ohne weitere Modifikationen auf den allgemeineren Fall eines Multilabel-Problems erweitert werden. Wir formalisieren nun das One-against-all-Verfahren. Für die

Eingabe: Trainingsmenge $\langle(\bar{x}_1, \mathcal{L}_1), \ldots, (\bar{x}_P, \mathcal{L}_P)\rangle$

initialisiere Perzeptrons $\bar{w}_1^1, \ldots, \bar{w}_1^K$
for $i = 1 \ldots P$ **do** ▷ Iteriere über alle $\bar{x}_i$ in Trainingsmenge
  for $k = 1 \ldots K$ **do** ▷ Iteriere über alle Klassen bzw. Basislerner
    if $y_k \in \mathcal{L}_i$ **then**
      $\mathcal{L}_i^k \leftarrow \{y_k\}$ ▷ $\bar{x}_i$ als positives Beispiel
    else
      $\mathcal{L}_i^k \leftarrow \{\overline{y_k}\}$ ▷ $\bar{x}_i$ als negatives Beispiel
    $\bar{w}_{i+1}^k \leftarrow$ TRAINIEREPERZEPTRON$(\bar{w}_i^k, (\bar{x}_i, \mathcal{L}_i^k))$ ▷ nach der Perzeptron-Lernregel
return $\bar{w}_{P+1}^1, \ldots, \bar{w}_{P+1}^K$

Abbildung 4.1: One-against-all am Beispiel des Perzeptron

fortan verwendete Schreibweise gelte, daß ein hochgestellter Index k die k-te Klasse bzw. den k-ten Basislerner referenziere.

Sei $\mathcal{S} = \langle(\bar{x}_1, \mathcal{L}_1), \ldots, (\bar{x}_P, \mathcal{L}_P)\rangle$ eine Multilabel-Trainingsfolge mit $\mathcal{L}_i \in \mathcal{Y}$ für die Menge von relevanten Klassen von $\bar{x}_i$. Sei K die Anzahl der Klassen $|\mathcal{Y}|$. Die binäre Zurückführung nach der One-against-all-Methode erzeugt aus der Trainingsmenge K binäre Trainingsmengen $\mathcal{S}^k = \langle(\bar{x}_1, \mathcal{L}_1^k), \ldots, (\bar{x}_P, \mathcal{L}_P^k)\rangle$ mit

$$\mathcal{L}_i^k = \begin{cases} \{y_k\} & , y_k \in \mathcal{L}_i \\ \{\overline{y_k}\} & , y_k \notin \mathcal{L}_i \end{cases} \tag{4.1.1}$$

Die Klasse $\overline{y_k}$ repräsentiert alle Klassen ungleich y_k, d. h. $\mathcal{L}_i \backslash \{y_k\}$. Die so erzeugten binären Beispielmengen können nun von den Basislernern zum Trainieren verwendet werden. Der Pseudocode in Abbildung 4.1 demonstriert eine mögliche Implementierung am Beispiel des Perzeptrons als Basislerner.

Bei der Definition des One-against-all-Verfahrens wurde offengelassen, wie die Vorhersagen der einzelnen Klassifizierer zu einer Gesamtaussage kombiniert werden. Es liegt nahe, die positive Vorhersage des k-ten Klassifizierers als Zuordnung zur k-ten Klasse zu interpretieren. Wir erhalten nach dieser Methode folgende Klassifizierfunktion für das One-against-all-Verfahren.

$$c_{\mathcal{S}}(\bar{x}) := \{y_k \mid c_{\mathcal{S}_k} = \{y_k\}\} \tag{4.1.2}$$

Ein Problem der One-against-all-Methode ist, daß die erzeugten Zweiklassen-Daten sehr unausbalanciert sind. Sie enthalten wenige positive Beispiele und sehr viele negative Beispielen. Der durchschnittliche Anteil der positiven Beispiele an den Gesamtbeispielen beträgt L/K, wobei L die durchschnittliche Anzahl Labels je Trainingsbeispiel angibt. Viele Basislerner haben damit Schwierigkeiten, u. a. auch das Perzeptron. Die trennende Hyperebene wird unter diesen Umständen zugunsten der negativen Klasse nahe an die positiven Beispiele herangeführt. Die Klassifizierfunktion des Perzeptron neigt dazu, eine negative Vorhersage abzugeben.

Um diesem Problem zu entgehen besteht die Möglichkeit, für die Beurteilung die Konfidenzwerte heranzuziehen. Für das Perzeptron nehmen wir dafür das bei der Klassifizierung berechnete Skalarprodukt aus Gewichtsvektor und Beispielvektor. Übertragen auf das Modell des künstlichen Neurons bedeutet dies, daß wir statt der Schwellwertfunktion die lineare Aktivierungsfunktion verwenden (siehe Abschnitt 3.2). Ein hoher Konfidenzwert steht für eine hohe Wahrscheinlichkeit, einer

bestimmten Klasse anzugehören. Bevor die Konfidenzwerte der verschiedenen Perzeptrons miteinander verglichen werden können, müssen sie normalisiert werden. Das Ergebnis ist der Konfidenzvektor

$$\bar{p} = (\frac{\bar{w}^1}{|\bar{w}^1|}\bar{x}, \ldots, \frac{\bar{w}^K}{|\bar{w}^K|}\bar{x}) \tag{4.1.3}$$

Der Konfidenzwert p_k ist der Abstand von Beispiel $\bar{x}$ zu der gerichteten Hyperebene $\bar{w}_k$. Der Vergleich der Konfidenzwerte führt zu einem Klassenranking, diese Variante des One-against-all ist demnach ein Klassen-Ranking-Verfahren. Aus dem Ranking kann anschließend wie in Abschnitt 2.4.1 beschrieben eine Vorhersage für das Multilabel-Problem reproduziert werden. Auch ohne diesen letzten Schritt kann die Klassifizierung anhand der in Abschnitt 2.4.2 beschriebenen Loss-Funktionen ermittelt werden.

4.2 Beschreibung des MMP

Der Multiclass-Multilabel-Perceptron-Algorithmus ist eine Erweiterung des One-against-all-Verfahrens. Beide Verfahren verwenden denselben Hypothesenraum, je Klasse ist demnach ein Vektor bzw. ein Perzeptron für die Beurteilung der Relevanz zuständig. Die Algorithmen unterscheiden sich jedoch grundlegend im Training. Während beim klassischen One-against-all jedes Perzeptron für sich unabhängig vom Rest trainiert wird, geschieht die Aktualisierung der Prototypen beim MMP-Algorithmus in einem globalen Zusammenhang. Die Aktualisierung hängt zudem maßgeblich von zwei Parametern ab: der verwendeten Kosten-Funktion und der verwendeten Aktualisierungsmethode. Bevor auf die unterschiedlichen Parameter eingegangen wird, folgt eine allgemeine Beschreibung der Funktionsweise des MMP.

Der erste Schritt in jeder Iteration, d. h. bei jedem neuen Trainingsbeispiel $\bar{x}$, ist das Testen der aktuellen Hypothese auf dem Beispiel. Dafür wird jeder Prototyp auf das Beispiel angewendet, das Ergebnis ist ein Konfidenzvektor $\bar{p}$. Die Konfidenzen in $\bar{p}$ werden sortiert und wir erhalten ein Klassenranking, das aufgrund einer Loss-Funktion bewertet wird (siehe Abschnitt 2.4.1 und 2.4.2). Wenn das Ranking perfekt ist, d. h. der Fehler $\delta(\bar{r}, \mathcal{L}) = 0$, ist das Training für dieses Beispiel abgeschlossen. Ist der Fehler größer Null, wird die Menge der falsch geordneten Klassenpaare ermittelt. Ein Paar einer positiven und einer negativen Klasse ist falsch klassifiziert, wenn die negative Klasse in der Rangordnung vor der relevanten steht.

$$E := \{(u,v) \mid r_u > r_v,\ y_u \in \mathcal{L},\ y_v \notin \mathcal{L}\} \tag{4.2.1}$$

Diese Fehlermenge spielt eine zentrale Rolle im Algorithmus, da nur die Prototypen von Klassen aktualisiert werden, die in dieser Menge vertreten sind. Eine Veranschaulichung findet sich in Abbildung 4.3. Die Pfeilbeziehungen zwischen den Klassen stellen die fehlerhaften Klassenpaare dar. Für jedes Klassenpaar (u,v) in E wird nun ein Aktualisierungsgewicht $\alpha^{u,v} \geq 0$ anhand der ausgewählten Aktualisierungsmethode berechnet. Dieses Gewicht kann sich beispielsweise danach richten, wie weit die Klassen u und v im Ranking voneinander entfernt sind. Nachfolgend werden die Gewichte normalisiert, so daß ihre Summe $\sum_{y_u \in \mathcal{L}} \sum_{y_v \notin \mathcal{L}} \alpha_{u,v} = 1$ ergibt. Für jeden Prototypen, dessen Klasse falsch geordnet wurde, wird abschließend der Faktor τ ermittelt, der die Aktualisierung $\bar{w}^k + \tau^k \bar{x}$ festlegt. Für jede Klasse werden dabei die α-Gewichte der Paare aufsummiert, an denen die Klasse beteiligt ist. Anschließend werden diese Werte mit dem durch die Loss-Funktion

Eingabe: Trainingsmenge $\langle(\bar{x}_1, \mathcal{L}_1), \ldots, (\bar{x}_P, \mathcal{L}_P)\rangle$

initialisiere Perzeptrons $\bar{w}_1^1, \ldots, \bar{w}_1^K$
for $i = 1 \ldots P$ **do** ▷ Iteriere über alle $\bar{x}_i$ in Trainingsmenge
  $\bar{p}_i \leftarrow (\bar{w}_i^1 \bar{x}_i, \ldots, \bar{w}_i^K \bar{x}_i)$ ▷ Berechne Konfidenzen
  $\bar{r}_i \leftarrow \text{SORT}(\bar{p}_i)$ ▷ Berechne Klassenranking
  $l_i \leftarrow \delta(\bar{r}_i, \mathcal{L}_i)$ ▷ Rankingfehler nach gewählter Loss-Funktion
  if $l_i > 0$ **then**
    $E_i \leftarrow \{(u,v) \mid r_{iu} > r_{iv},\ y_u \in \mathcal{L}_i,\ y_v \notin \mathcal{L}_i\}$ ▷ Falsch geordnete Klassenpaare
    for all $(u,v) \in E_i$ **do**
      $\alpha_i^{u,v} \leftarrow \text{UPDATEMETHODE}((u,v), \bar{r}_i)$ ▷ Nach gewählter Methode
    for all y_k vertreten in E **do** ▷ Für alle falsch geordneten Klassen y_k

$$\tau_i^k \leftarrow \begin{cases} l_i \sum_{y_v \notin \mathcal{L}} \alpha_i^{k,v} & y_k \in \mathcal{L}_i \\ -l_i \sum_{y_u \in \mathcal{L}} \alpha_i^{u,k} & y_k \notin \mathcal{L}_i \end{cases}$$

    $\bar{w}_{i+1}^k \leftarrow \bar{w}_i^k + \tau_i^k \bar{x}_i$ ▷ Aktualisiere Perzeptrons
return $\bar{w}_{P+1}^1, \ldots, \bar{w}_{P+1}^K$

Abbildung 4.2: MMP-Algorithmus

ermittelten Fehler multipliziert und bei irrelevanten Klassen das Vorzeichen negativ gemacht. Eine schematische Darstellung des Algorithmus findet sich in Abbildung 4.2.

4.2.1 Haupteigenschaften

Um die Haupteigenschaften des MMP-Algorithmus aufzuzeigen, werden die Hauptunterschiede zum zuvor vorgestellten One-against-all-Verfahren mit dem Perzeptron als Basislerner, das zuvor vorgestellt wurde, aufgezeigt.

Bei dem One-against-all-Verfahren wird jedes Perzeptron für sich allein betrachtet. Das Ziel ist es, jedes Perzeptron so zu modellieren, daß es bei Beispielen aus seiner positiven Klasse ein positives Skalarprodukt liefert und bei allen anderen Klassen ein negatives Skalarprodukt. Dabei wird nicht berücksichtigt, daß durch das Zusammenfügen der Einzelvorhersagen jedes Perzeptron in Konkurrenz zu den anderen steht, eine korrekte Vorhersage des Prototypen einer Klasse demnach nicht unbedingt ausreichend ist, um diese Klasse beim Klassenranking korrekt zu platzieren.

Beim MMP-Algorithmus werden die Basislerner als voneinander abhängige Komponenten eines Gesamtsystems betrachtet. Ziel des Gesamtsystems ist es, insgesamt eine korrekte Vorhersage zu treffen. Für eine korrekte Gesamtvorhersage müssen beim MMP-Algorithmus die Skalarprodukte der Perzeptrons der relevanten Klassen größer sein als die der Perzeptrons der irrelevanten Klassen. Dieser Unterschied zwischen den Konfidenzwerten der negativen und der positiven Klassen, von den Autoren als *margin* bezeichnet, ist folgendermaßen definiert.

$$\mu := \min_{y_u \in \mathcal{L}} \{\bar{w}^u \bar{x}\} - \max_{y_v \notin \mathcal{L}} \{\bar{w}^v \bar{x}\} \tag{4.2.2}$$

Das Klassenranking für ein Beispiel $\bar{x}$ ist perfekt, falls der Abstand μ positiv ist. Für eine gute Leistung gilt es demnach, $\min_i\{\mu_i\}$ für alle $\bar{x}_i$ zu maximieren. Dies versucht der MMP-Algorithmus

zu erreichen, indem bei einem falsch klassifizierten Beispiel die Perzeptrons aktualisiert werden, die an dem negativen Abstand beteiligt sind (dies geschieht durch Bestimmung der Fehlermenge E). Für Perzeptrons von positiven Klassen wird der Normalenvektor in Richtung des Beispiels verschoben und somit das Skalarprodukt erhöht, dementsprechend wird bei negativen Klassen in die Gegenrichtung verschoben um das Skalarprodukt zu verringern. Insgesamt wird dadurch der Abstand μ gesteigert. Um das Ziel des positiven Abstands möglichst schnell zu erreichen, ist der Grad der Translation eines Perzeptron zusätzlich davon abhängig, wie stark das Perzeptron für den schlechten Abstand verantwortlich ist.

Der Grad der Verschiebung wird beim MMP-Algorithmus auf drei Wegen beeinflußt. Erstens ist der Grad von der gewählten Aktualisierungsmethode abhängig, die beurteilt, wie schwerwiegend die falsche Anordnung von zwei Klassen zueinander ist. Zweitens von der Anzahl der Positionen, um die eine Klasse in der Rangordnung falsch platziert wurde, da dies der Anzahl der Paare in der Fehlermenge E entspricht, an der die Klasse beteiligt ist. Und drittens von der Schwere des Fehlers des Gesamtsystems, und damit von der Wahl der Loss-Funktion. Die Verwendung des Maßes μ für die Festlegung des Grades der Aktualisierung findet nur implizit statt.

Ein weiterer Unterschied zwischen beiden Verfahren ist ihr konservatives Verhalten. Die einzelnen Basislerner arbeiten beim One-against-all-Verfahrens zwar konservativ, da das Perzeptron selbst konservativ ist, doch das Gesamtsystem tut es nicht, da trotz perfektem Ranking die Perzeptrons verändert werden, falls sie selbst nicht konsistent auf dem zu klassifizierenden Beispiel sind. Der MMP-Algorithmus arbeitet konservativ, da die Basislerner nur aktualisiert werden, wenn es einen Rankingfehler gibt. Zusätzlich sind die einzelnen Basislerner konservativ, wenn man die Konsistenz derart definiert, das die Bedingung des positiven Abstands $\mu > 0$ durch die Hypothesen der Basislerner erfüllt werden muß. Die Eigenschaft, daß nur die Prototypen aus der Fehlermenge E verändert werden, nennen die Autoren *ultrakonservatives Verhalten*.

Wie bereits das Perzeptron sind das One-against-all-Verfahren auf Basis von Perzeptrons und der MMP-Algorithmus inkrementelle Algorithmen. Nach jedem Trainingsbeispiel kann ein Klassifizierer angegeben werden. Für die Evaluierung auf einer vorgegebenen Trainings- und Testmenge wird die Trainingsmenge als Folge von Beispielen dem Lernalgorithmus übergeben. Die Anzahl der Epochen gibt an, wie oft der Lerner die Trainingsfolge durchläuft. Für das Testen wird der Klassifizierer genommen, der nach dem letzten Beispiel entstanden ist (siehe Abschnitt 2.3.3 und 3.5).

4.2.2 Aktualisierungsmethoden

Summiert man die Verschiebungen bzw. Aktualisierung aller Perzeptrons zu einem gesamten Translationsvektor zusammen, so ergibt sich insgesamt für eine Iteration die Verschiebung $l_i \bar{\mathrm{x}}_\mathrm{i}$ für die falsch klassifizierten positiven Klassen und die Verschiebung $-l_i \bar{\mathrm{x}}_\mathrm{i}$ für die falsch klassifizierten negativen Klassen. Dies wird zum einen durch die Bedingung sicher gestellt, daß die Summe der α-Gewichte eins betragen muß, und zum anderen dadurch, daß jedes α-Gewicht zum Faktor τ einer relevanten und einer irrelevanten Klasse addiert wird. Dies veranschaulicht folgende Rechnung.

$$\sum_{y_u \in \mathcal{L}} \tau^u = \sum_{y_u \in \mathcal{L}} l \sum_{y_v \notin \mathcal{L}} \alpha^{u,v} = l \sum_{y_u \in \mathcal{L}} \sum_{y_v \notin \mathcal{L}} \alpha^{u,v} = l \tag{4.2.3}$$

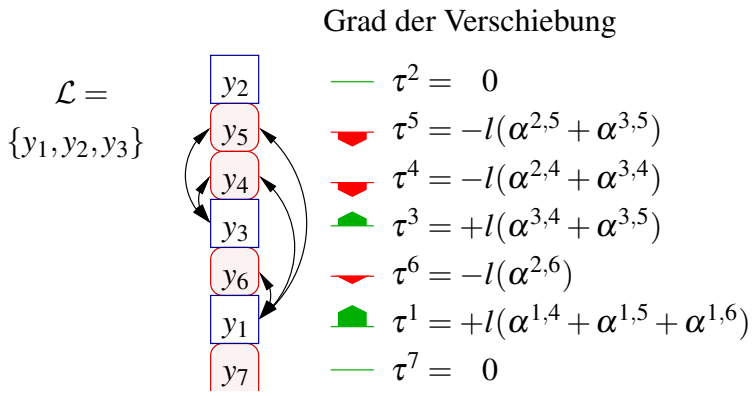


Abbildung 4.3: MMP-Aktualisierung am Beispiel eines fehlerhaften Rankings

Damit ist auch $\sum_{y_v \notin \mathcal{L}} \tau^v = -l$. Wie groß der Anteil der Aktualisierung eines einzelnen Prototypen an der gesamten Aktualisierung ist, wird im MMP-Verfahren durch die Wahl der Aktualisierungsmethode bestimmt. Als Bedingung für die Auswahl der Gewichte α gilt, daß nur für Paare aus der Fehlermenge E Gewichte bestimmt werden dürfen, die Gewichte positiv sein müssen und ihre Summe eins ergeben muß. Crammer und Singer führen vier mögliche Methoden auf, die im folgenden beschrieben werden.

Uniform-Update Bei dieser Methode erhält jedes fehlerhaft geordnete Klassenpaar dasselbe Gewicht, d. h. für jedes Paar $(u,v) \in E$ gilt $\alpha^{u,u} = 1/|E|$. Dies hat zur Folge, daß der Grad der Aktualisierung proportional zur Falschplatzierung im Ranking ist. Der Prototyp beispielsweise einer Klasse, die besonders schlecht platziert wurde, wird stark aktualisiert, da die Klasse oft in der Menge der Falschpaare E vertreten ist. Eine Veranschaulichung bietet Abbildung 4.3: das Perzeptron von Klasse y_1 wird am stärksten verschoben, da $\tau^1 = l\,(3\frac{1}{|E|}) = \frac{3}{5}\,l$.

Max-Update Bei der Max-Update-Methode wird nur das Paar aus E mit dem höchsten Rangunterschied gewichtet. Dies bedeutet, es wird die am niedrigsten geordnete positive Klasse und die am höchsten geordnete negative Klasse aktualisiert. Formal gilt

$$\alpha^{a,b} = \begin{cases} 1 & \bar{\mathrm{r}}_a = \min\limits_{y_u \in \mathcal{L}} \{\bar{\mathrm{r}}_u\}\,,\ \bar{\mathrm{r}}_b = \max\limits_{y_v \notin \mathcal{L}} \{\bar{\mathrm{r}}_v\} \\ 0 & \text{sonst} \end{cases} \tag{4.2.4}$$

Proportional-Update Zu jedem Klassenpaar wird die Differenz der Skalarprodukte ermittelt und das Aktualisierungsgewicht auf diesen Wert gesetzt.

$$\alpha^{u,v} = \bar{\mathrm{w}}^v \bar{\mathrm{x}} - \bar{\mathrm{w}}^u \bar{\mathrm{x}} \tag{4.2.5}$$

Anschließend werden die Gewichte normalisiert. Das Gewicht eines Klassenpaares ist beim Proportional-Update daher proportional zu ihrem Abstand $\mu^{u,u} = \bar{\mathrm{w}}^v \bar{\mathrm{x}} - \bar{\mathrm{w}}^u \bar{\mathrm{x}}$. Der erzielte Gesamteffekt

ähnelt dem des Uniform-Updates, bei dem die Perzeptrons proportional zum Abstand in Rängen aktualisiert werden. Diese Wirkung wird durch das Proportional-Update verstärkt.

Werden die Perzeptrons mit den Nullvektoren initialisiert, muß dies bei der ersten Iteration berücksichtigt werden und für $\alpha^{u,u}$ vernünftige Werte eingesetzt werden.

Random-Update Diese Methode ordnet jedem Paar aus E einen zufälligen, gleichverteilten Wert aus dem Intervall $[0,1]$ zu. Die so generierten $\alpha^{u,v}$ werden anschließend normalisiert.

4.2.3 Loss-Funktionen

Das Gesamtausmaß der Aktualisierung wird abgesehen vom unbeeinflußbaren Faktor des Trainingsbeispiels vom Fehler des Klassifizierers auf dem Beispiel bestimmt. Die Bestimmung des Fehlers erfolgt durch eine Loss-Funktion auf dem Ranking. Diese Funktionen wurden ausführlich in Abschnitt 2.4.2 behandelt. Wie an jener Stelle beschrieben, berücksichtigen unterschiedliche Loss-Funktionen unterschiedliche Aspekte bei der Beurteilung des Rankings. Durch Wahl der Loss-Funktion kann das Verhalten des MMP-Algorithmus an unterschiedliche Anforderungen an die Klassifizierung angepaßt werden. Die Anforderungen des MMP-Algorithmus selbst an die Loss-Funktion ist, daß bei einem perfekten Ranking der Funktionswert Null beträgt und daß der Wert beschränkt ist.

Crammer und Singer geben die bereits vorgestellten Maße IsError-Loss und ErrSetSize-Loss an (siehe Abschnitt 2.4.2). Zusätzlich benutzen sie drei weitere Loss-Funktionen aus dem *Information Retrieval*, die nachfolgend beschrieben werden. Auch der vorgestellte Margin-Loss erfüllt die Bedingungen des MMP an die Loss-Funktion und kann somit eingesetzt werden.

OneError-Loss Der *OneError-Loss* gibt an, ob die erst-platzierte Klasse eine relevante Klasse ist. Die Loss-Funktion kann folgendermaßen angegeben werden.

$$\begin{aligned} &\delta_{OneErr} : \pi^K \times 2^{\mathcal{Y}} \to \{0,1\} \\ &\delta_{OneErr}(\bar{r}, \mathcal{L}) := 1 - I[y_{r_1} \in \mathcal{L}] \end{aligned} \tag{4.2.6}$$

Average-Precision-Loss Der *Average-Precision-Loss* ermittelt den durchschnittlichen Precision-Wert für ein gegebenes Ranking. Bei der Berechnung werden nur die ersten $|\mathcal{L}|$ Positionen berücksichtigt. Angelehnt an die Notation zu Recall und Precision aus Abschnitt 2.4.2, definieren wir die durchschnittliche Precision folgendermaßen

$$AvgP := \frac{1}{|\mathcal{L}|} \sum_{y_k \in \mathcal{L}} precision_k = \frac{1}{|\mathcal{L}|} \sum_{y_k \in \mathcal{L}} \frac{TP_k}{TP_k + FP_k} \tag{4.2.7}$$

Da die Average-Precision für ein perfektes Ranking eins ist, verwenden wir für die Loss-Funktion $1 - AvgP$:

$$\begin{aligned} &\delta_{AvgP} : \pi^K \times 2^{\mathcal{Y}} \to [0, {}^{K-1}/_{K}] \\ &\delta_{AvgP}(\bar{r}, \mathcal{L}) := 1 - AvgP \end{aligned} \tag{4.2.8}$$

MaxF1-Loss Das F1-Maß ist ein beliebtes Maß, um Recall und Precision zu einem einzigen Maß zu verknüpfen. Der F1-Wert ist das harmonische Mittel von Recall und Precision.

$$F1_k := \frac{2\, recall_k \cdot precision_k}{recall_k + precision_k} \tag{4.2.9}$$

Der *MaxF1-Loss* wählt aus den möglichen F1-Werten den höchsten aus. Da auch beim F1-Maß der maximale Wert eins ist, muß dies bei der Gestaltung der Loss-Funktion berücksichtigt werden.

$$\begin{aligned} &\delta_{MaxF1} : \pi^K \times 2^{\mathcal{Y}} \to [0,1] \\ &\delta_{MaxF1}(\bar{\mathrm{r}}, \mathcal{L}) := 1 - \max_{0 \le k \le K} \{F1_k\} \end{aligned} \tag{4.2.10}$$

4.3 Fehlerabschätzung

Auch für das MMP läßt sich eine Schranke für die Anzahl der Fehler angeben. Es wird wieder angenommen, daß eine Hypothese existiert, die die Daten perfekt klassifiziert.

Sei $\langle(\bar{\mathrm{x}}_1, \mathcal{L}_1), (\bar{\mathrm{x}}_2, \mathcal{L}_2), \ldots, (\bar{\mathrm{x}}_P, \mathcal{L}_P)\rangle$ eine Trainingsfolge mit $R \ge |\bar{\mathrm{x}}_\mathrm{i}|$. Sei $\bar{\mathrm{v}}^1, \ldots, \bar{\mathrm{v}}^K$ eine Menge von Perzeptrons mit $\sum_{0 \le k \le K} (\bar{\mathrm{v}}^k)^2 = 1$, die für jedes Trainingsbeispiel ein perfektes Ranking vorhersagen mit

$$\mu = \min_{1 \le i \le \mathcal{X}} \left\{ \min_{y_u \in \mathcal{L}} \{\bar{\mathrm{w}}^u \bar{\mathrm{x}}\} - \max_{y_v \notin \mathcal{L}} \{\bar{\mathrm{w}}^v \bar{\mathrm{x}}\} \right\} > 0 \tag{4.3.1}$$

Sei A der maximale Wert der gewählten Loss-Funktion, so daß für alle Beispiele $l^i < A$ gelte. Dann ist der akkumulierte Fehler des MMP-Algorithmus beschränkt durch

$$\sum_{i=1}^{K} l^i \le 2A \left(\frac{R}{\mu}\right)^2 \tag{4.3.2}$$

Für den Beweis wird ähnlich verfahren wie beim Beweis für die Fehlerschranke des Perzeptrons in Abschnitt 3.5. Es wird eine obere und eine untere Schranke für $\sum_k |\bar{\mathrm{w}}_{i+1}^k|^2$ angegeben, bei der die untere schneller wächst als die obere. Dabei wird ausgenutzt, daß der Grad der Gesamtaktualisierung nur von l^i abhängt. Ein vollständiger Beweis findet sich in [Crammer und Singer, 2003, Kapitel 5].

Werden die Beispielvektoren und die Loss-Funktion normalisiert, so daß $R' = 1$ und $A' = 1$ gilt, so ergibt sich als Schranke für den akkumulierten Fehler

$$\sum_{i=1}^{K} l'^i \le 2\frac{1}{\mu'^2} \qquad , l'^i = l^i/A\,,\ \mu' = \mu'/R \tag{4.3.3}$$

Es wird deutlich, daß der akkumulierte Fehler antiproportional zum Quadrat des Abstands μ ist, Der Abstand μ stellt den maximal möglichen Margin zwischen relevanten und irrelevanten Klassen bei den Konfidenzwerten dar.

4.4 Komplexitätsanalyse

Für die Analyse der Komplexität des MMP-Algorithmus seien zur Erinnerung nochmals die wichtigsten Variablen genannt: K bezeichne die Anzahl der Klassen, N die Anzahl der Attribute. Die Angabe der Laufzeit erfolgt wie bereits bei der Komplexitätsanalyse des Perzeptrons in Anzahl der Operationen Π. Die Laufzeit eines Π hängt maßgeblich von der durchschnittlichen Anzahl M der verwendeten Attribute je Beispiel ab und entspricht der Laufzeit einer Skalarprodukt-Operation.

Da für jede Klasse ein Perzeptron verwendet wird, beträgt der Speicherverbrauch das K-fache des Speicherverbrauch eines Perzeptrons. Zusätzlich wird $O(2K)$ Speicher für die Konfidenzen und für das Ranking benötigt. Es ergibt sich ein Speicherverbrauch von $O(KN+2K) = O(KN)$ sowohl für das Training als auch für das Testen.

Für das Testen eines Beispiels werden K Skalarprodukte berechnet. Die Sortierung der Konfidenzvektoren benötigt $O(K \log K)$ Vergleichsoperationen. Wir konzentrieren uns jedoch bei dieser Untersuchung nur auf die Anzahl der Π-Operationen, da wir davon ausgehen , daß diese viel teurer als Vergleichsoperationen sind.

Jedes Beispiel wird beim Training als erstes klassifiziert, um festzustellen, ob es bereits korrekt klassifiziert wird. Die Laufzeit für ein perfekt klassifiziertes Beispiel beträgt somit K Π-Operationen.

Im Falle eines fehlerhaften Rankings erhöht sich die Anzahl der Π. Für jedes aktualisierte Perzeptron kommt ein Π hinzu. Betrachten wir zuerst den Fall des Uniform-, Proportional- und Random-Updates. Aktualisiert werden diejenigen Perzeptrons, deren Klassen sich im Ranking zwischen der am höchsten platzierten negativen Klasse und der am niedrigsten platzierten positiven Klasse befinden, inklusive dieser beiden Klassen. Dieser Wert entspricht den um eins erhöhten Margin-Loss. Die zusätzliche Anzahl Operationen für ein falsch klassifiziertes Trainingsbeispiel beträgt somit $\delta_{margin} + 1$. Den durchschnittlichen IsError-Fehler kann man als Anteil der Trainingsbeispiele verstehen, die falsch klassifiziert wurden. Das bedeutet, daß $\hat{\delta}_{IsErr}P$ Beispiele falsch klassifiziert wurden, und nur auf diesen fallen die zusätzlichen Π-Operationen an. Der akkumulierte Margin-Loss beträgt auf diesen Daten wie für den kompletten Trainingssatz $\hat{\delta}_{margin}P$. Die zusätzliche Anzahl von Operationen für die falsch klassifizierten Trainingsbeispiele beträgt somit

$$\hat{\delta}_{margin}P + \hat{\delta}_{IsErr}P \tag{4.4.1}$$

Da im schlimmsten Fall alle K Prototypen aktualisiert werden müssen, beträgt die Laufzeit je Trainingsbeispiel

$$\Omega(K) = K + \hat{\delta}_{margin} + \hat{\delta}_{IsErr} = O(2K) \tag{4.4.2}$$

Bei der Max-Update-Methode werden je falsch klassifiziertem Beispiel genau zwei Prototypen aktualisiert. Die Laufzeit je Beispiel beträgt somit $K + 2\hat{\delta}_{IsErr}$.

Für das One-against-all-Verfahren gelten gleiche Bedingungen für den Speicherverbrauch und der Laufzeit auf den Testdaten. Für das Training unterscheidet sich die Angabe der Laufzeit, die Schranken bleiben jedoch bestehen.

Der Fehler des k-ten Perzeptrons auf seinem Trainingssatz $\mathcal{S}^k$ beträgt $\hat{\delta}_{IsErr}(c_{\mathcal{S}^k}, \mathcal{S}^k)$. Wir definieren $\hat{\delta}_{per}$ als Durchschnitt dieser Werte. Die Gesamtanzahl der Perzeptronfehler auf dem Trainingssatz beim One-against-all-Verfahren ist somit $\hat{\delta}_{per}KP$ und damit die Laufzeit je Beispiel

$$\Omega(K) = K + \hat{\delta}_{per}K = K(1 + \hat{\delta}_{per}) = O(2K) \tag{4.4.3}$$

Da die einzige gesicherte Beziehung zwischen δ_{per}, δ_{IsErr} und δ_{margin} die ist, daß alle drei Funktionen gleichzeitig Null ergeben, läßt sich keine Aussage darüber machen, welches Verfahren effizienter ist.

Die Angaben zur Laufzeit und zum Speicherverbrauch sind nochmals in einer Gegenüberstellung in Tabelle 5.1 zu finden (S. 70).

4.5 Einsatz bei der Textklassifizierung

Der MMP-Algorithmus wurde von Crammer und Singer auf zwei Textsammlungen getestet: dem älteren Reuters-21578-Korpus und dem neueren, größeren Reuters-Korpus aus dem Jahr 2000, das in Abschnitt 6.1 ausführlich beschrieben wird. Wir konzentrieren uns auf die Ergebnisse auf dem letzten Datensatz.

Der große Reuters-Korpus besteht aus mehr als 800.000 Dokumenten. Diesen können beliebig viele der 102 Kategorien zugeordnet werden, im Durchschnitt besitzt jedes Dokument ca. 3 Klassenzuordnungen. Die Kategorien sind hierarchischen aufgebaut, diese Struktur wird jedoch in den Experimenten nicht ausgenutzt. Versuche wurden auf vier verschiedenen Trainingsmengen durchgeführt, die Anzahl der Beispiele reicht von 639 bis 521.439. Als Methode für die Wortgewichtung für die Transformation in den Vektorraum wurde die *Singhal-Methode* gewählt (näher beschrieben in Abschnitt 6.3.3). Mit einer Feature-Selection auf Basis des Rocchio-Algorithmus wurde das Vokabular auf 4.529 Wörter bei der kleinsten Trainingsmenge und 9.325 bei der größten reduziert. Es wurden alle möglichen Kombinationen der Parameter des MMP-Algorithmus verglichen (Aktualisierungsmethoden: Uniform-, Max-, Proportional- und Random-Update. Loss-Funktionen: IsError, ErrSetSize, OneError, AvgP und MaxF1). Zusätzlich wurde der *Rocchio-Algorithmus* und das One-against-all-Perzeptron-Verfahren getestet. Der Fehler auf der Testmenge wurde anhand aller verfügbaren Loss-Funktionen ermittelt, unabhängig von der beim Training eingesetzten Kosten-Funktion. Die Beobachtungen dieser Versuchsanordnung lassen sich folgendermaßen zusammenfassen:

Die Reihenfolge der besten Aktualisierungsmethoden ist folgende: Uniform-, Random-, Proportional- und Max-Update. Besonders mit dem Zuwachs der Trainingsmenge bildet sich diese Reihenfolge klar heraus. Das Ergebnis läßt sich folgendermaßen interpretieren. Die Random-Update-Methode vergibt im Durchschnitt dieselben Gewichte wie Uniform-Update und konvergiert aus diesem Grund in dem Verhalten zum Uniform-Update. Beim Proportional-Update kommt es offenbar zu einem geringfügigen *Overfitting*, die sehr schlechten Prototypen werden sehr stark aktualisiert und die restlichen fehlerhaften Prototypen vernachlässigt. Dies zeigt sich auch in der Leistung der Max-Update-Methode, die nur jeweils die zwei schlechtesten Prototypen aktualisiert. Diese letzte Methode schneidet mit Abstand am schlechtesten ab.

Als bestes Fehlermaß für die Aktualisierung ist die einfache IsError-Funktion auszumachen. Diese Beobachtung ist unabhängig von der Loss-Funktion, die für die Bewertung auf den Testdaten verwendet wird.

Weiterhin von Bedeutung ist die Beobachtung, daß es bei mehrmaligem Durchlaufen der Trainingsmenge zu einem starken Overfitting kommt.

Das One-against-all-Verfahren schlägt die verschiedenen Kombinationen, die sich beim MMP-Algorithmus ergeben, in etwa 20% der Fälle. Besonders beim Vergleich der OneError-Werte positioniert sich das One-against-all-Perzeptron sehr nah an den Spitzenleistungen des MMP. Beim Vergleich der ErrSetSize-Werte wiederum kann man einen klaren Unterschied zugunsten des MMP beobachten. Dies läßt sich auf die fehlende Berücksichtigung des Rankings beim One-against-all-Verfahrens zurückführen im Gegensatz zum MMP, das Ranking-sensitiv arbeitet.

Exemplarisch wird im folgenden das Ergebnis für den MMP-Algorithmus unter Verwendung der Uniform-Update-Methode und dem IsError-Loss auf der größten Trainingsmenge genannt. Der MMP-Algorithmus klassifiziert 30,68% der Testbeispiele perfekt, die durchschnittliche ErrorSetSize beträgt 2,87. Auf der kleineren Reuters-21578-Datenbank betragen die Werte 13,07% und 1,33. Obwohl diese Werte nicht direkt miteinander verglichen werden können, zeigen sie doch einen wesentlichen Unterschied insbesondere bei den IsError-Werten. Eine mögliche Erklärung ist die wesentlich geringere Anzahl an relevanten Klassen je Beispiel von 1,24 bei ähnlicher Gesamtanzahl Klassen bei dem Reuters-21578-Korpus. Erstens ist dadurch eine Überlappung der Kategorien unwahrscheinlicher, die Dokumente demnach eindeutiger einer Kategorie zuordenbar. Und zweitens ist zu erwarten, daß sich bei Zunahme der Anzahl der relevanten Klassen der Margin μ unabhängig von der zunehmenden Überlappung der Kategorien verringert und dadurch eine Klassifizierung durch die Perzeptrons erschwert wird.

Um diese Werte beurteilen zu können, wären Vergleichswerte mit anderen Verfahren notwendig. Doch obwohl beide Korpora weitläufig verwendet werden, sind die hier vorgestellten Bewertungsmethoden für Klassen-Rankings nicht sehr verbreitet. Der mitverwendete Rocchio-Algorithmus, der insgesamt eine schlechtere Leistungen als der One-against-all-Ansatz erbrachte, läßt keinen vernünftigen Vergleich zu, wie die Autoren bemerken, da ihre Implementierung womöglich nicht an die Aufgabe des Klassen-Rankings angepaßt war.

Wir versuchen einen Vergleich mit den Ergebnissen von Granitzer [2003, S. 69] auf dem großen Reuters-Datensatz. In diesen Untersuchungen wird mit einem Lernalgorithmus experimentiert, der die Daten unter Ausnutzung der Hierarchie in kleinere Probleme aufteilt, die von einem Basislerner gelöst werden. Für das Training wurden etwa 18.000 und für das Testen 26.000 Dokumente verwendet (ob eine Feature-Selection durchgeführt wurde ist nicht ersichtlich). Für die beste Variante basierend auf Support-Vektor-Maschinen wurden folgende Ergebnisse erzielt: 16,49% OneError-Loss und 0,840 AvgP-Loss[1]. Für die nachfolgenden Ergebnisse des MMP wurden Trainingsmengen von 5.139 und 50.139 Beispielen verwendet und auf 10.000 Trainingsbeispielen getestet (Uniform-Update, IsError-Loss): 8,03% respektive 0,854 für die kleinere Trainingsmenge und 5,16% respektive 0,906 für die größere. Auf den ersten Blich scheint selbst bei der kleineren Trainingsmenge der MMP-Algorithmus bessere Ergebnisse als der SVM-Algorithmus zu erzielen. Obwohl die Aussage dieses Vergleichs aufgrund der unterschiedlichen Ausgangslagen nur mit großer Vorsicht zu genießen ist, läßt sich intuitiv auf eine konkurrenzfähige Leistung des MMP-Algorithmus im Vergleich zu etablierten Textlernalgorithmen schließen.

Ein klarere Beurteilung erlaubt möglicherweise der kleinere Reuters-21578-Datensatz. Der MMP-Algorithmus erzielt in der Uniform-IsError-Kombination einen *Breakeven-Point* bei den Recall-

[1] Die Angabe des AvgP-Loss bezieht sich auf die Berechnung wie in Gleichung 4.2.7 angegeben, nicht auf die an die Bedingungen des MMP angepaßte Loss-Funktion δ_{AvgP}.

Precision-Werten von 0,85 [Crammer und Singer, 2003, Abb. 6]. Im Vergleich mit den Ergebnissen aus der ausführlichen Zusammenstellung in [Sebastiani, 2002, Tabelle 6] ist dieser Wert in der Rangfolge im vordersten Feld anzusiedeln.[2] Auch bei dieser Gegenüberstellung ist zu beachten, daß unterschiedliche Versuchsbedingungen galten.

Die Ergebnisse für das One-against-all-Verfahren sind 38,86% IsError-Loss, 6,04% OneError-Loss und 10,43 ErrorSetSize-Loss auf dem großen Reuters-Datensatz respektive 15,71%, 9,59% und 4,87 auf dem kleinen Satz. Es ist erwähnenswert, daß die Perzeptrons hierbei denselben Raum an über 80 Stellen teilen. Wenn man nun die Reuters-Korpora als ausreichend repräsentativ für Textdaten betrachtet, und davon ausgeht, daß ein geringer Fehler eines Perzeptrons auf eine gute lineare Separierbarkeit der Daten schließen läßt (großer Margin μ), bestätigen die akzeptablen Ergebnisse des One-against-all-Verfahrens die in Abschnitt 3.10 geäußerte Vermutung, daß Textdaten allgemein gut linear trennbar sind.

[2] Vergleichswerte z. B. C4.5 0,79, Naive Bayes 0,80, BalancedWinnow 0,82, RIPPER 0,82, k-NN 0,82-0,86, SVM 0,84-0,87, AdaBoost.MH 0,88

5 Multilabel-Pairwise-Coupling

Das *Pairwise-Coupling-Verfahren*, auch *one-against-one*, *round robin* oder *pairwise binarization* genannt, ist eine Class-Binarization-Methode. Dabei wird für jedes mögliche Klassenpaar ein binäres Problem konstruiert, an dem jeweils die positiven Beispiele beteiligt sind. Bei einem K-Klassen-Problem ergeben sich somit $\frac{K(K-1)}{2}$ binäre Probleme, die jeweils von einem Basislerner gelöst werden. Eine Möglichkeit für die Bildung einer Gesamtvorhersage besteht in der Durchführung einer Abstimmung, d. h. die Vorhersage jedes Basisklassifizierers wird als Stimme für eine von zwei Klassen interpretiert. Es gewinnt die Klasse mit den meisten Stimmen.

Dieses Verfahren läßt sich leicht an Multilabel-Daten anpassen. Die Erweiterung, die man *Multilabel-Pairwise-Coupling* nennen könnte, steht in Konkurrenz zu dem MMP-Algorithmus. Bestehende Überlegungen und Untersuchungen zu Class-Binarization-Methoden bestätigen dem paarweisen Ansatz Vorteile gegenüber der klassischen One-against-all-Methode. In einer Gegenüberstellung im weiteren Verlauf dieser Arbeit wird überprüft, ob sich diese Überlegungen auch auf einen Vergleich mit dem MMP-Algorithmus übertragen lassen.

Es folgt zunächst eine ausführlichere Beschreibung der paarweisen Zurückführung und der im Rahmen dieser Arbeit entwickelten Erweiterung auf den allgemeinen Fall einer Multilabel-Klassifikation.

5.1 Beschreibung der Funktionsweise

Wir konzentrieren uns bei der Beschreibung des Pairwise-Coupling-Verfahrens auf den Fall, daß Perzeptrons als Basislerner verwendet werden. Dies entspricht der in dieser Arbeit verwendeten Konfiguration und stellt darüber hinaus keine Einschränkung der Funktionsweise dar.

Die Pairwise-Coupling-Methode generiert aus einem initialen Multiklassen-Problem mit K Klassen $\binom{K}{2} = \frac{K(K-1)}{2}$ kleinere Zweiklassen-Probleme. Für jedes mögliche Klassenpaar (y_u, y_v), $y_u, y_v \in \mathcal{Y}, u < v$ wird ein Perzeptron $\bar{\mathrm{w}}^{uv}$ trainiert. Das Perzeptron $\bar{\mathrm{w}}^{uv}$ wird mit Beispielen $(\bar{\mathrm{x}}_i, \mathcal{L}_i)$ angelernt, die den Klassen y_u oder y_v zugeordnet sind. Hierzu wird ein Beispiel aus y_u als positives Beispiel trainiert während ein Beispiel aus y_v als negatives aufgefaßt wird. Das Ziel dabei besteht darin, daß der Klassifizierer zwischen beiden Klassen unterscheiden lernt. Ein Beispiel aus y_u wird somit zum Trainieren von $K-1$ Perzeptrons $\bar{\mathrm{w}}^{1u}, \bar{\mathrm{w}}^{2u}, \ldots, \bar{\mathrm{w}}^{u-1,u}, \bar{\mathrm{w}}^{u,u+1}, \ldots, \bar{\mathrm{w}}^{u,K}$ verwendet. Dies macht auch deutlich, daß die Pairwise-Coupling-Methode nicht mehr Schritte beim Training benötigt als die One-against-all-Methode, wie zunächst die hohe Anzahl an Basislernern befürchten läßt.

Der *Multilabel-Pairwise-Coupling-Algorithmus* (*MLPC*) berücksichtigt die Möglichkeit, daß ein Beispiel in einer Multilabel-Klassifikation mehreren Klassen zugeordnet sein kann. Hierbei wird ein Beispiel mit der Klassenzuordnung $\mathcal{L}$ zum Trainieren derjenigen Basislernern verwendet, die zwischen einer der relevanten Klassen aus $\mathcal{L}$ und einer der irrelevanten Klassen aus $\overline{\mathcal{L}} = \mathcal{Y} \backslash \mathcal{L}$

Eingabe: Trainingsmenge $\langle(\bar{x}_1, \mathcal{L}_1), \ldots, (\bar{x}_P, \mathcal{L}_P)\rangle$

1: initialisiere Perzeptrons $\{\bar{w}_1\}^{uv}$, $0 < u < v \leq K$
2: **for** $i = 1 \ldots P$ **do** ▷ Iteriere über alle $\bar{x}_i$ in Trainingsmenge
3: **for all** $(y_u, y_v) \in \mathcal{L}_i \times \mathcal{Y} \backslash \mathcal{L}_i$ **do**
4: **if** $u < v$ **then**
5: $\bar{w}_{i+1}^{uv} \leftarrow$ TRAINIEREPERZEPTRON$(\bar{w}_i^{uv}, (\bar{x}_i, \{y_u\}))$ ▷ $\bar{x}_i$ als positives Beispiel
6: **else**
7: $\bar{w}_{i+1}^{vu} \leftarrow$ TRAINIEREPERZEPTRON$(\bar{w}_i^{vu}, (\bar{x}_i, \{y_v\}))$ ▷ $\bar{x}_i$ als negatives Beispiel
8: **return** $\{\bar{w}_{P+1}\}^{uv}$, $0 < u < v \leq K$

Abbildung 5.1: Training des Multilabel-Pairwise-Coupling-Algorithmus

diskriminieren. Die Funktionsweise des MLPC-Algorithmus ist schematisch in Abbildung 5.1 dargestellt. Für ein Perzeptron $\bar{w}^{uv}$ bedeutet dies, daß ein Beispiel $\bar{x}$, zusätzlich zu dem im Pairwise-Coupling betrachteten Fall $u, v \notin \mathcal{L}$, ignoriert wird, falls $u, v \in \mathcal{L}$. Hier sind auch andere Vorgehensweisen denkbar, die beispielsweise versuchen, das Beispiel *in der Mitte* zu positionieren, oder den Datentypus und die Eigenarten des Basislerners zu berücksichtigen. [1]

Eine Folge der Tatsache, daß es sich um Multilabel-Daten handelt ist, daß mehr Basislerner pro Schritt trainiert werden müssen. Für ein Trainingsbeispiel müssen alle möglichen Paarungen von relevanten und irrelevanten Klassen aus $\mathcal{L} \times (\mathcal{Y} \backslash \mathcal{L})$ berücksichtigt werden, das ergibt somit $|\mathcal{L}| \cdot (K - |\mathcal{L}|)$ Kombinationen.

Das Multilabel-Pairwise-Coupling-Verfahren erbt wie schon die One-against-all-Methode die Eigenschaft des Basislerners, die Trainingsbeispiele inkrementell zu lernen. Zudem ist der paarweise Ansatz wie bereits die One-against-all-Binarization nach außen hin kein konservativer Algorithmus, da die Konsistenz mit dem aktuellen Trainingsbeispiel nicht überprüft wird.

Angewandt auf Multiklassen-Daten verhält sich der MLPC-Algorithmus identisch zur Pairwise-Coupling-Methode. Das MLPC-Verfahren, so wie es hier beschrieben wird, läßt sich auf alle Klassifikationstypen anwenden, die in Abschnitt 2.2 beschrieben wurden. Es gestattet somit allgemein für beliebige Klassifikationen eine Rückführung auf die Zweiklassen-Klassifikation mit einfacher Zuordnung.[2]

Für das Zusammenfügen der Einzelvorhersagen der Basisklassifizierer im MLPC-Algorithmus kann beispielsweise dasselbe einfache Verfahren aus der Einleitung dieses Kapitels verwendet werden. Dieses stellt das Standardverfahren dar und wird u. a. *Max-Wins*, *Plurality-Voting* oder einfach nur *Voting* genannt.

Ein Beispiel $\bar{x}$ wird von allen erzeugten $\frac{K(K-1)}{2}$ Basisklassifizierern klassifiziert, so daß man die Vorhersage als Stimme für eine Klasse interpretieren kann. Jedes Perzeptron $\bar{w}^{u,v}$ stimmt demnach für u oder für v. Die Auszählung ergibt einen Konfidenzvektor $\bar{p}$, wobei p_k der Anzahl der Stimmen für Klasse y_k entspricht. Das Klassenranking $\bar{r}$ gibt die Klassen geordnet nach absteigender Anzahl

[1] Z.B. kann berücksichtigt werden, ob sich die Klassen einer Klassifikation in ihrer Intension überschneiden wie beispielsweise bei einer hierarchischen Klassifikation (ein Gegenbeispiel zur Überschneidung der Intension ist die Multilabel-Zuordnung eines Textes zu einem Autor).

[2] Die Zweckmäßigkeit bei der Zweiklassen-Klassifikation darf bezweifelt werden.

von Stimmen wieder. Bei einem Unentschieden in der Anzahl der Stimmen kann beispielsweise aufgrund der A-priori-Klassenwahrscheinlichkeit über die Reihenfolge bestimmt werden [Fürnkranz, 2001, 2002]. Im Falle einer Multiklassen-Klassifikation, bzw. wenn die Anzahl der relevanten Klassen vorgegeben oder sogar bekannt ist, kann nun eine Menge von positiven Klassen als Vorhersage angegeben werden. Allgemein kann der MLPC-Algorithmus als Klassen-Ranking-Algorithmen betrachtet werden (siehe Abschnitt 2.4.2).

Die offensichtlichste Schwäche des Votings stellt die große Anzahl irrelevanter Abstimmungen dar. Obwohl nur $K-1$ Klassifizierer Wissen über eine positive Klasse besitzen können, dürfen weitere $\frac{(K-1)(K-2)}{2}$ *„unqualifizierte“* Klassifizierer eine Stimme abgeben. Im besten Fall schlagen sich diese Stimmen als ein Rauschen auf der Abstimmung nieder. Im schlimmsten Fall jedoch konzentrieren sich diese auf eine negative Klasse. Dies kann beispielsweise bei Abhängigkeiten zwischen den Klassen oder bei unausbalancierten Klassen (und dafür empfänglichen Lernern) verstärkt auftreten. Allerdings kann jede Klasse höchstens $p_k \leq K-1$ Stimmen erhalten, und wenn alle Basisklassifizierer der positiven Klasse korrekt abstimmen, fehlt jeder negativen Klasse im besten Fall eine Stimme, die sie an die positive Klasse verloren hat (siehe dazu auch die ausführlichere Untersuchung in Abschnitt 5.6. Eine weitere Auswirkung der vielen irrelevanten Stimmabgaben ist die erhöhte Laufzeit. Die Anzahl der Vorhersageoperationen wächst quadratisch mit der Anzahl der Klassen, hingegen verhält sich das One-against-all-Verfahren in hierbei linear. In Abschnitt 5.8 werden Ansätze beschrieben, die sich mit diesen beiden Schwierigkeiten auseinandersetzen.

5.2 Vergleich zum One-against-all-Verfahren

Beim One-against-all-Verfahren wird ein Multiklassen-Problem in binäre Probleme aufgeteilt, die jedoch denselben Umfang wie das ursprüngliche Problem besitzen. Es wurde lediglich die Multiklassen-Klassifikation der Beispiele auf eine binäre Zuordnung reduziert. Das Pairwise-Coupling-Verfahren stützt sich auf die Überlegungen, daß sich kleinere Probleme einfacher und besser lösen lassen [z. B. Fürnkranz, 2002]. Ausgehend von demselben begrenzten Instanzenraum, läßt sich für eine kleine Menge (binärer) Beispiele leichter ein Klassifizierer finden als für eine große Menge, da weniger Beispiele berücksichtigt werden müssen, und es kann eine bessere Generalisierung eines lösenden Klassifizierers erwartet werden, da die Beispiele insgesamt unähnlicher zueinander sind. Aus diesem Grund wird beim Pairwise Coupling das Hauptproblem in kleine Probleme aufgeteilt, die jeweils nur Beispiele aus zwei Klassen enthalten. Die Anzahl der Beispiele in den kleineren Problemen beträgt bei einem K-Klassen-Problem im Durchschnitt ein $\frac{2}{K}$-tel der Gesamtanzahl der Beispiele.

Ein weiterer Vorteil könnte für Lernalgorithmen bestehen, die sensibel gegenüber einem unausgeglichenen Verhältnis zwischen positiven und negativen Beispielen sind und folglich bei der Hypothesenbildung eine der Klassen bevorzugen könnten. Das Verhältnis beträgt beim One-against-all-Verfahren im Durchschnitt 1 zu K, beim Pairwise-Coupling-Verfahren ist es ausgeglichen.

Den Überlegungen zum vorteilhaften Training bei der paarweisen Aufteilung steht die Problematik der hohen Anzahl an störenden Einzelvorhersagen bei der Klassifizierung gegenüber. Um zu einer abschließenden Bewertung der verschiedenen Argumente zu kommen, werfen wir einen Blick auf empirische Ergebnisse zum Vergleich zwischen der Pairwise-Coupling- und One-against-all-Methode.

Bei den meisten im folgenden aufgezählten Untersuchungen wurden Support-Vektor-Maschinen als Basislerner verwendet. Weston und Watkins [1999] beispielsweise präsentieren eine Variante der Support-Vektor-Maschine, die direkt ein Multiklassen-Problem lernt. Zum Vergleich verwenden sie das klassische One-against-all und das Pairwise-Coupling-Verfahren. Die Versuche finden auf sehr kleinen Datensätzen statt und es kann kein Unterschied zwischen den Ansätzen festgestellt werden. Platt u. a. [1999] stellen eine Alternative zur üblichen Abstimmung bei der Kombinierung der Vorhersagen vor, die sie die *Directed Acyclic Graphs* Variante (*DAG*) nennen (siehe dazu die Beschreibung in Abschnitt 5.8). In diesem Zusammenhang vergleichen sie ihre Methode mit den üblichen One-against-all und Pairwise-Coupling. Das letztere Verfahren kann in einem Ergebnis eine signifikante Verbesserung aufweisen, doch die Untersuchung läßt keine allgemeine Aussage zu. Hsu und Lin [2002] vergleichen One-against-all, Pairwise-Coupling, DAG und ein Multiklassen-SVM-Algorithmus. Es sind durchweg bessere Werte für den Pairwise-Coupling-Ansatz im Vergleich zu den restlichen Verfahren zu beobachten. Yao u. a. [2001] vergleichen die Leistungen mehrerer Class-Binarization-Verfahren bei der Klassifizierung von Fingerabdrücken. Als Basislerner setzen sie Support-Vektor-Maschinen ein und erreichen eine leichte Verbesserung durch den Einsatz von Pairwise-Coupling gegenüber dem One-against-all. Fürnkranz [2002] vergleicht den Pairwise-Coupling-Algorithmus direkt mit dem One-against-all-Verfahren und der *ordered class binarization*, einer Variante des One-against-all für den Multiklassen-Fall. Dabei kommt der Regellerner *RIPPER* [Cohen und Singer, 1996] als Basislerner zum Einsatz. Die Versuche finden auf mehreren Datensätzen statt. Der paarweise Ansatz erzielt in den meisten Fällen eine wesentliche Verbesserung, in keinem Fall kann eine signifikante Verschlechterung festgestellt werden. Besonders durch das erfolgversprechende letzte Resultat motiviert, stellt sich die Frage, ob sich die positiven Ergebnisse auch auf andere Problemstellungen übertragen lassen.

5.3 Motivation für den MLPC

Der MMP-Algorithmus ist eine Adaptation der One-against-all-Methode, die auf den Multilabel-Fall ausgerichtet ist. Der Einsatz von Perzeptrons und die einfache Funktionsweise erlauben es dem Algorithmus, große Datenmengen inkrementell und sehr effizient zu verarbeiten, an denen andere etablierte Lernalgorithmen aufgrund von Laufzeit- und Speicheranforderungen scheitern. Gleichzeitig sorgt die Adaptation der Lernregel für eine bedeutende Leistungssteigerung der Klassifizierung im Vergleich zum One-against-all-Ansatz. Zudem scheint der Algorithmus damit an die Leistungen der anerkannten Textklassifizierer anschließen zu können (siehe Abschnitt 4.5). Der MMP-Algorithmus zeigt außerdem, daß der Perzeptron-Algorithmus trotz oder gerade wegen seiner Einfachheit eine gute Grundlage für leistungsstarke Verfahren bietet.

Weiterhin wurde mit dem Pairwise-Coupling-Verfahren ein Class-Binarization-Verfahren vorgestellt, das mit der Aufteilung in viele kleine Teilprobleme einen alternativen Ansatz verfolgt. Es gibt eindeutige Zeichen dafür, daß das Pairwise-Coupling-Verfahren klare Vorteile gegenüber der klassischen One-against-all-Methode besitzt. Mit dem neu vorgestellten MLPC-Algorithmus wurde das Pairwise-Coupling-Verfahren auf den allgemeinen Fall der Multilabel-Klassifikation erweitert.

Aus der nahen Verwandtschaft von MMP und One-against-all ergibt sich nun folgende zentrale Fragestellung

- Sind die Vorteile des paarweisen Ansatzes gegenüber der One-against-all-Methode ausreichend, um an den Leistungsschub des MMP-Algorithmus heranzureichen.

Um einen Vergleich unter gleichen Voraussetzungen zu erlauben, wird hierfür der im Rahmen dieser Arbeit entwickelte MLPC-Algorithmus auf Basis des Perzeptron-Algorithmus verwendet. Diese Kombination wird fortan MLPC$_p$genannt. Bevor jedoch ein direkte Gegenüberstellung stattfinden kann, sind zunächst folgende Fragen zu beantworten:

- Sind für den Einsatz von Perzeptrons als Basislerner auch Vorteile für das Pairwise-Coupling-Verfahren gegenüber dem One-against-all-Verfahren zu erwarten?
- Lassen sich die Überlegungen zur One-against-all-Methode in Beziehung zum Pairwise-Coupling-Verfahren trotz der Modifikationen auf den MMP-Algorithmus übertragen.
- Ist der MLPC-Algorithmus im Stande, die Vorteile des paarweisen Ansatzes auf den Multilabel-Fall zu übertragen?

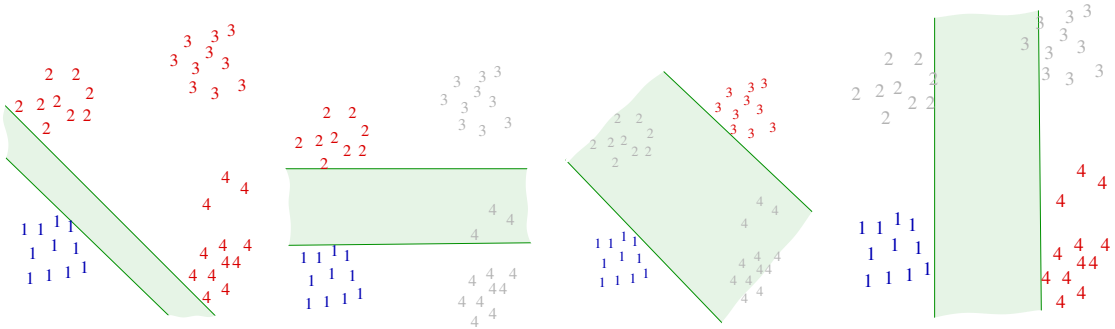
Diese Punkte werden im weiteren Verlauf dieser Arbeit untersucht. In Kapitel 6 wird schließlich die zentrale Fragestellung durch einen direkten Vergleich zwischen MMP und MLPC$_p$-Algorithmus beantwortet. Als Grundlage hierfür bietet sich das Reuters-Korpus an, da es bereits bei der Evaluierung des MMP-Algorithmus verwendet wurde und durch die großen Datenmengen und die hohe Komplexität der Klassifikation hohe Anforderungen an einen Multilabel-Lernalgorithmus stellt. Zudem besitzen Ergebnisse auf diesen Daten hohe Aussagekraft, da der Datensatz unter realen Bedingungen erstellt wurde und hohen Ansprüchen genügt. Durch Verwendung der Reuters-Datenbank resultieren folgende zusätzliche Fragestellungen:

- Ergeben sich durch den Einsatz von Perzeptrons als Basislerner ähnliche positive Eigenschaften für die Laufzeit- und Speicheranforderungen wie für den MMP-Algorithmus?
- Ist damit der MLPC-Algorithmus auch für die Verwendung als Textklassifizierer geeignet?

5.4 Pairwise-Coupling bei linearen Problemen

Es wird nun untersucht, ob sich die Überlegungen zu den Vorteilen für die Lernbarkeit von kleinen gegenüber großen Problemen auch auf das Lösen von linearen Problemen übertragen lassen. Wie wir in Abschnitt 3.5 beobachten konnten, hängt die Schwierigkeit des Perzeptron-Algorithmus, eine trennende Hyperebene zu finden, maßgeblich vom Margin einer möglichen Hyperebene ab. Als Maß für die Schwierigkeit betrachten wir hierbei die Anzahl der Trainingsfehler und damit der Iterationen, die die Perzeptron-Lernregel für das Finden der Hyperebene benötigt. Auch die Güte des daraufhin gefundenen Perzeptrons, d. h. die verallgemeinernde Fähigkeit des erzeugten Modells, hängt von dem Freiraum zwischen den binären Punktmengen ab, da bei einem großen Zwischenraum der Perzeptron-Algorithmus eine größere Chance hat, eine Hyperebene mit großem Margin zu finden. Zusammenfassend läßt sich feststellen, daß die Lernfähigkeit eines linearen Lerners und die Lösbarkeit eines binären Problems proportional zu einem wachsenden Freiraum zwischen den positiven und negativen Beispielen des Problems zunimmt.

Nun ist es noch nötig, eine Beziehung zwischen Problemgröße und Margin nachzuweisen. Abbildung 5.2 zeigt die Aufspaltung eines Vier-Klassen-Problems nach der One-against-all und nach der Pairwise-Coupling-Methode. Es zeigt sich intuitiv, daß die Punktzwischenräume mit einer Reduzierung der Anzahl der Punkte zunehmen.



(a) One-against-all: Klasse $\{1\}$ gegen $\{2,3,4\}$

(b) Pairwise-Coupling: Klasse $\{1\}$ vs. $\{2\}$

(c) Pairwise-Coupling: Klasse $\{1\}$ gegen $\{3\}$

(d) Pairwise-Coupling: Klasse $\{1\}$ gegen $\{4\}$

Abbildung 5.2: Gegenüberstellung der One-against-all- und Pairwise-Coupling-Methode im zweidimensionalen Raum. Die Abbildungen zeigen den erhöhten Margin beim Pairwise-Coupling-Verfahren, indem das initiale Problem, Klasse $\{1\}$ von den restlichen zu unterscheiden, paarweise in mehrere Teilprobleme aufgeteilt wird.

Diese Beziehung wird durch folgendes Szenario deutlicher aufgezeigt. Wir betrachten zwei initiale Punkte unterschiedlicher Klassen in einem Vektorraum, die wir als Mittelpunkte der Klassen betrachten. Der Abstand zwischen den Klassen ist zu diesem Zeitpunkt maximal. Nun fügen wir allmählich einer Klasse Punkte hinzu. Diese positionieren wir um den initialen Punkt nach einer willkürlichen Verteilung. Wenn wir die Beziehung $\varphi(n)$ zwischen Anzahl der eingefügten Punkte und Abstand untersuchen, so stellen wir fest, daß der Abstand mit wachsender Anzahl der Punkte monoton fällt. Dieser Abstand ist bei $\varphi(2)$ maximal und erreicht bei $n \to \infty$ ein Minimum, das bei entsprechender Verteilung $-\infty$ sein kann.[3] In dem Intervall $2 \to \infty$ findet somit mit Sicherheit eine Veränderung des Abstands statt (es sei denn, es gilt $\varphi(2) = \varphi(\infty)$). Die Wahrscheinlichkeit einer Reduktion des Abstands hängt somit von der Anzahl der Punkte ab, die eingefügt werden. Diese Beziehung ist bei Entnahme von Punkten genau umgekehrt. Es läßt sich somit generell behaupten, daß der Margin bei Reduktion der Problemgröße größer wird.

Aufgrund des größeren Margins bei kleineren Problemen ist demnach auch bei linear trennenden Algorithmen eine Verbesserung durch die paarweise Aufteilung gegenüber dem One-against-all-Verfahrens zu erwarten.

5.5 Gültigkeit für den MMP-Algorithmus

Wir untersuchen als nächstes, ob die Ähnlichkeit zwischen dem One-against-all-Verfahren und dem MMP-Algorithmus dazu ausreicht, um die theoretischen Überlegung zu der Abhängigkeit der Leistung von der Problemgröße auf den MMP-Algorithmus zu übertragen.

Beim MMP-Algorithmus wird bei den Einzelproblemen der Raum nicht explizit in zwei Hälften für positive und negative Beispiele aufgeteilt. Eine Hyperebene muß lediglich möglichst so ge-

[3] Ein negativer Abstand ist als die Ausbreitung (in Richtung der direkten Verbindung zwischen den Klassen) des Raumes zu verstehen, in dem beide Punkte gemeinsam vorkommen. Vgl. dazu auch Abschnitt 3.4 und 3.7

wählt werden, daß die Differenz zwischen dem kleinsten Skalarprodukt des Normalenvektors mit den positiven Beispielen und dem größten Skalarprodukt mit den negativen Beispielen maximiert wird, das Vorzeichen der Skalarprodukte ist dabei unerheblich. Zudem müssen insgesamt die Prototypen der Teilprobleme so gewählt werden, daß bei der Gegenüberstellung der Vorhersagen der einzelnen Prototypen die Differenz zwischen den Skalarprodukten für relevante Klassen und den Skalarprodukten der irrelevanten Klassen maximiert wird (um eine perfekte globale Klassifizierung zu erreichen, siehe Abschnitt 4).

Die erste Bedingung wird von der optimalen trennenden Hyperebene erfüllt. Durch Parallelverschiebung der Ebene und einer Streckung des Normalenvektors kann das Minimum bzw. Maximum der Skalarprodukte für positive bzw. negative Beispiele auf einen beliebigen Wert gesetzt werden. Die zweite Bedingung kann demzufolge durch eine geschickte Wahl der Translation und Skalierung erreicht werden. Die optimale Hypothese des MMP-Algorithmus ist folglich wie bei dem One-against-all-Verfahren eine Menge von optimalen Perzeptrons mit zwei zusätzlichen Einstellgrößen je Prototyp, die die Einzelvorhersagen zu einem perfekten Ranking zusammenfügen.

Da die Leistungen der Einzelprototypen wie bei dem One-against-all-Verfahren von dem Margin der binären Teilprobleme abhängt, ist auch für den MMP-Algorithmus von einer Abhängigkeit der Lernfähigkeit von der Problemgröße auszugehen.

5.6 Analyse des Votings

In diesem Abschnitt wird das Voting-Verfahren untersucht, das die Einzelvorhersagen der Basisklassifizierer zu einer Gesamtvorhersage durch eine Abstimmung zusammenfügt. Dabei ist von besonderem Interesse, welche Auswirkungen die Erweiterung des Pairwise-Coupling-Verfahrens auf den Multilabel-Fall zur Folge hat. Auf den ersten Blick ist dabei mit einer größeren Anzahl relevanter Klassen je Beispiel eine Verschlechterung der Qualität der Abstimmung verbunden, da mehr Klassen in dem Klassenranking auf die oberen Plätze gebracht werden müssen. Um dies zu untersuchen, wird zunächst eine allgemeingülige Modellierung der Abstimmung vorgestellt.

Beim Voting werden insgesamt $\frac{K(K-1)}{2}$ Stimmen vergeben. Jeder Basisklassifizierer besitzt für die Abstimmung eine Stimme. Seien y_u und y_v die Klassen, zwischen denen sich ein Basisklassifizierer $c^{u,v}$ entscheiden kann, dann bezeichne $s^{u,v}$ die Stimme dieses Klassifizierers. Es gilt $s^{u,v} = 1$, wenn sich der Basisklassifizierer für Klasse y_u entscheidet und $s^{u,v} = 0$, wenn er für y_v wählt. Es existieren nur für $u < v$ Basisklassifizierer. Zur Vereinfachung der Schreibweise und ohne Beschränkung der Allgemeinheit definieren wir eine Symmetrie auf den Stimmen. Somit gilt für die Stimme $s^{u,v}$ folgendes

$$s^{u,v} := \begin{cases} I[c^{u,v}(\bar{\mathrm{x}}) = \{y_u\}] & u < v \\ 0 & u = v \\ 1 - s^{v,u} & u > v \end{cases} \tag{5.6.1}$$

Die Gesamtanzahl der Stimmen für eine Klasse y_k sei somit gegeben durch

$$p^k = \sum_{\substack{1 \le n \le K \\ n \neq k}} s^{k,n} \; < K - 1 \tag{5.6.2}$$

Bei der Klassifizierung eines Beispiels mit der Menge von relevanten Klassen $\mathcal{L}$ lassen sich die Basisklassifizierer in zwei Gruppen aufteilen. In der ersten Gruppe befinden sich die Klassifizierer, die Informationen für die Vorhersage besitzen. Sie wurden jeweils mit Beispielen aus einer positiven Klasse und einer negativen Klasse trainiert. Die Mitglieder der zweiten Gruppe sind diejenigen Klassifizierer, die keinerlei Wissen für die Klassifizierung besitzen, da sie angelernt wurden, um entweder zwischen zwei positiven oder zwei negativen Klassen zu unterscheiden. Diese letzte Gruppe ist für die in Abschnitt 5.1 als Rauschen bezeichnete Beeinflussung der Abstimmung verantwortlich. Bei jeder Wahl existieren $|\mathcal{L}| \cdot (K - |\mathcal{L}|)$ Mitglieder in der ersten und $\binom{|\mathcal{L}|}{2} + \binom{K-|\mathcal{L}|}{2}$ in der zweiten Gruppe. Die Aufteilung der Klassifizierer in beide Gruppen ist bei jedem Beispiel unterschiedlich, je nach der Menge der relevanten Klassen des Beispiels. Bei der Angabe des Abstimmergebnisses läßt sich die Aufteilung der Stimme in zwei Gruppen ausnutzen, indem vom Ursprung der Stimme unterschieden wird. Für eine positive Klasse y_i lautet demnach die Auszählung folgendermaßen

$$p^i = \sum_{y_v \notin \mathcal{L}} s^{i,v} + \sum_{\substack{y_u \in \mathcal{L} \\ u \neq i}} s^{i,u} \tag{5.6.3}$$

Es können $K - |\mathcal{L}|$ der wissenden und $|\mathcal{L}| - 1$ der unwissenden Klassifizierer eine Stimme für eine positive Klasse abgeben. Die Auszählung für eine negative Klasse y_j läßt sich folgendermaßen angeben

$$p^j = \sum_{y_u \in \mathcal{L}} s^{j,u} + \sum_{\substack{y_v \notin \mathcal{L} \\ v \neq j}} s^{j,v} \tag{5.6.4}$$

Hierbei ist das Verhältnis zwischen den Basisklassifizierern $|\mathcal{L}|$ zu $K - |\mathcal{L}| - 1$. Die Angabe der Bedingung $u \neq i$ bzw. $v \neq j$ bei der zweiten Summe ist nicht notwendig, da $s^{k,k} = 0$ gilt, und wird deshalb aus Gründen der Übersichtlichkeit fortan ausgelassen.

Eine Wahl ist erfolgreich, wenn jede relevante Klassen mehr Stimmen erhält als jede irrelevante Klasse, d. h. wenn $p^i < p^j$ für alle $y_i \in \mathcal{L}, y_j \notin \mathcal{L}$ gilt. Aus den Berechnungen von p^i und p^j wird deutlich, daß der Erfolg der Wahl somit an erster Stelle vom Anteil der wissenden Klassifizierer abhängt, die korrekt abstimmen, und an zweiter Stelle vom Grad der Abweichung vom Rauschen der unwissenden Klassifizierer, d. h. von der Gleichverteilung auf alle Klassen ihrer Für- und Gegenstimmen. Wir untersuchen deshalb die erwartete Differenz zwischen den Stimmen der relevanten Klassen und den Stimmen der irrelevanten Klassen $p^i - p^j$, da dieser entscheidend für die Wahl ist. Dazu benötigen wir zunächst den durchschnittlichen Fehler der Klassifizierer. Dieser unterscheidet sich je nach Gruppierung des Klassifizierers. In der nachfolgenden Gleichung sind die Berechnung des Fehlers e_α der wissenden Klassifizierer und der Fehler e_β der unwissenden angegeben

$$e_\alpha := 1 - \frac{1}{|\mathcal{L}|(K - |\mathcal{L}|)} \sum_{y_u \in \mathcal{L}} \sum_{y_v \notin \mathcal{L}} s^{u,v} \quad \in [0,1] \tag{5.6.5}$$

$$e_\beta := \frac{1}{\binom{|\mathcal{L}|}{2} + \binom{K-|\mathcal{L}|}{2}} \left(\sum_{y_u \in \mathcal{L}} \sum_{\substack{y_{u'} \in \mathcal{L} \\ u' > u}} s^{u,u'} + \sum_{y_v \notin \mathcal{L}} \sum_{\substack{y_{v'} \notin \mathcal{L} \\ v' > v}} s^{v,v'} \right) = \tfrac{1}{2} \tag{5.6.6}$$

Der Wert e_α entspricht dem Fehler des verwendetem Basislerner. Die Erwartungswerte für p^i und

p^j lassen sich nun folgendermaßen angeben

$$\begin{aligned}
\mathrm{E}\left[p^i \mid y_i \in \mathcal{L}\right] &= \mathrm{E}\left[\sum_{y_v \notin \mathcal{L}} s^{i,v}\right] + \mathrm{E}\left[\sum_{y_u \in \mathcal{L}} s^{i,u}\right] \\
&= \sum_{y_v \notin \mathcal{L}} (1 - e_\alpha) + \sum_{y_u \in \mathcal{L}} e_\beta \\
&= (K - |\mathcal{L}|)(1 - e_\alpha) + (|\mathcal{L}| - 1)\tfrac{1}{2}
\end{aligned} \tag{5.6.7}$$

$$\begin{aligned}
\mathrm{E}\left[p^j \mid y_j \notin \mathcal{L}\right] &= \mathrm{E}\left[\sum_{y_u \in \mathcal{L}} s^{j,u}\right] + \mathrm{E}\left[\sum_{y_v \notin \mathcal{L}} s^{j,v}\right] \\
&= \sum_{y_u \in \mathcal{L}} e_\alpha + \sum_{y_v \notin \mathcal{L}} e_\beta \\
&= |\mathcal{L}| e_\alpha + (K - |\mathcal{L}| - 1)\tfrac{1}{2}
\end{aligned} \tag{5.6.8}$$

Durch Ermittlung der durchschnittlichen Differenz μ zwischen der Anzahl der Stimmen für positive Klassen und für negative Klassen läßt sich feststellen, von welchen Größen eine erfolgreiche Wahl in erster Linie abhängt.

$$\begin{aligned}
\mathrm{E}\left[\mu^{i,j}\right] =& \mathrm{E}[p^i - p^j \mid y_i \in \mathcal{L}, y_j \notin \mathcal{L}] \\
=& \mathrm{E}\left[p^i \mid y_i \in \mathcal{L}\right] - \mathrm{E}\left[p^j \mid y_j \notin \mathcal{L}\right] \\
=& K\left(1 - \tfrac{1}{2} - e_\alpha\right) + |\mathcal{L}|\left(-1 + \tfrac{1}{2} + \tfrac{1}{2} + e_\alpha - e_\alpha\right) + \left(-\tfrac{1}{2} + \tfrac{1}{2}\right) \\
=& K(\tfrac{1}{2} - e_\alpha)
\end{aligned} \tag{5.6.9}$$

Erstaunlicherweise hängt dieser Wert nicht von der Anzahl der Labels eines Beispiels ab. Im Durchschnitt ist somit die Distanz zwischen den Stimmen der positiven und den Stimmen der negativen Klassen nicht von der Anzahl der relevanten Klassen eines Beispiels abhängig, sondern nur von der Anzahl der Klassen und von dem Fehler des Basisklassifizierers.

Dies ist plausibel, da durch die erhöhte Anzahl der Labels zwar die möglichen *sicheren* Stimmen der wissenden Klassifizierer für eine positive Klasse vermindert werden, im Gegensatz jedoch erhält eine negative Klasse weniger von den für sie vorteilhaften Stimmen der unwissenden Klassifizierer, da es mehr direkte Auseinandersetzungen mit relevanten Klassen gibt.

Allerdings ergibt sich daraus auch, daß durch ein höhere Anzahl von Labels die Anzahl der von e_α abhängigen Stimmen abnimmt und das Rauschen, d. h. die von $e_\alpha = \frac{1}{2}$ abhängigen Stimmen, zunimmt. Auf den Erwartungswert von $\mu^{i,j}$ hat dies wie gezeigt keinen Einfluß. Doch es ist zu erwarten, daß die Verteilung von $\mu^{i,j}$ um $K(\frac{1}{2} - e_\alpha)$ flacher wird, d. h. mit größerem $|\mathcal{L}|$ eine höhere Varianz besitzt. Die Wahrscheinlichkeit, daß $\mu^{i,j} < 0$ gilt, wird dadurch erhöht. Dies über den Rahmen der Arbeit hinaus weiter zu untersuchen, erscheint lohnenswert.

5.7 Komplexitätsanalyse

Für die Komplexitätsanalyse werden die bereits bekannten Variablen K für die Anzahl der Klassen und N für die Anzahl der Attribute benötigt. Zusätzlich sei L die durchschnittliche Anzahl relevanter

Klassen je Beispiel. Wir beginnen mit der bei dem MLPC$_p$-Algorithmus unkomplizierten Analyse der Speicheranforderungen.

Da für K Klassen $\frac{K(K-1)}{2}$ Basisklassifizierer gebraucht werden und je Perzeptron $O(N)$ Speicher verbraucht wird, beträgt der Speicherverbrauch beim MLPC$_p$-Algorithmus sowohl für das Training als auch für das Klassifizieren $O(\frac{K(K-1)}{2}N) = O(K^2N)$.

Für die Klassifizierung eines Beispiels führt der MLPC$_p$-Algorithmus für jedes Perzeptron eine Skalarproduktoperation durch, die Laufzeit beträgt somit

$$\frac{K(K-1)}{2} = O(K^2) \tag{5.7.1}$$

Die Laufzeit für die Sortierung ist, wie bereits bei der Komplexitätsanalyse des MMP-Algorithmus erwähnt, vernachlässigbar.

Beim Lernen übergibt der MLPC-Algorithmus das Beispiel $\bar{x}$ allen Basisklassifizierern zum Trainieren, die zwischen den positiven Klassen in $\mathcal{L}$ und $\mathcal{Y}\backslash\mathcal{L}$ unterscheiden. Ein Perzeptron führt je Beispiel eine Π-Operation aus um festzustellen, ob es die Instanz bereits korrekt klassifiziert. Es ergeben sich insgesamt $|\mathcal{L}|(K-|\mathcal{L}|)$ Operationen. Je falsch klassifiziertem Beispiel wird zusätzlich für das Aktualisieren der Hypothese eine Π-Operation ausgeführt. Wir verfahren wie bei der Laufzeit-Analyse des One-against-all-Verfahrens in Abschnitt 4.4 und geben mit $\hat{\delta}_{per}$ den Durchschnittsfehler der Perzeptrons auf der Trainingsmenge an. Die Gesamtanzahl der Perzeptronfehler auf der Trainingsmenge beträgt somit $\hat{\delta}_{per} \cdot L(K-L) \cdot P$. Dies ergibt folgende durchschnittliche Laufzeit je Trainingsbeispiel für den MLPC$_p$-Algorithmus

$$\Omega(LK) = L(K-L)(1+\hat{\delta}_{per}) = O(2LK) \tag{5.7.2}$$

Im schlimmsten Fall, wenn $|\mathcal{L}| = \frac{K}{2}$ gilt und jedes Perzeptron seine Hypothese aktualisieren muß, beträgt die Laufzeit $\frac{1}{2}K^2 = O(K^2)$. Es ist interessant zu beobachten, daß der Anstieg der durchschnittlichen Anzahl von relevanten Klassen je Beispiel die Komplexität für das Training im starken Maße ansteigen läßt bis zu dem Grad, bei dem die Laufzeit quadratisch in der Anzahl der Klassen wird. Dies zeigt sich auch im direkten Vergleich mit dem MMP-Algorithmus.

Der MMP-Algorithmus benötigt für das Lernen eines Beispiels $K + \hat{\delta}_{margin} + \hat{\delta}_{IsErr}$ Operationen, der MLPC$_p$-Algorithmus im Vergleich dazu $L(K-L)(1+\hat{\delta}_{per})$. Für das Verhältnis MLPC$_p$ zu MMP gelten folgende Schranken

$$\begin{aligned} \frac{1}{4}L &= L\frac{\frac{1}{2}K}{2K} \leq L\frac{K-L}{2K} \\ &\leq \frac{L(K-L)(1+\hat{\delta}_{per})}{K+\hat{\delta}_{margin}+\hat{\delta}_{IsErr}} \\ &\leq L\frac{2K-L)}{K} < L\frac{2K}{K} < 2L \end{aligned} \tag{5.7.3}$$

Im Durchschnitt dürfte der MLPC$_p$-Algorithmus die L-fache Zeit für das Trainieren benötigen. Beim Klassifizieren ist der paarweise Ansatz klar im Nachteil, da jedes der Basisklassifizierer eine Vorhersage berechnet. Das Verhältnis beträgt hierbei $\frac{K-1}{2}$. Dies zeigt auf, daß es sich bei dem MLPC$_p$-Algorithmus um ein symmetrisches Verfahren handelt, das längere Zeit für das Klassifizieren als für das Trainieren benötigt (ähnlichen den *Lazy-Algorithmen*). Für das Verhältnis der

	Laufzeit		Speicher
	Training	Testen	Training & Testen
Perzeptron	$1+\hat{\delta}_{per}$	1	N
MMP	$K+\hat{\delta}_{margin}+\hat{\delta}_{IsErr}$	K	$O(KN)$
MLPC$_p$	$L(K-L)(1+\hat{\delta}_{per})$	$\frac{K(K-1)}{2}$	$O(K^2N)$
MLPC$_p$/MMP	$O(L)$	$\frac{K-1}{2}$	$O(K)$

Tabelle 5.1: Vergleich der Laufzeit- und Speicherkomplexität. Die Laufzeiten sind angegeben in durchschnittlicher Anzahl der Skalarprodukt-Operationen je Beispiel. Weitere wichtige Größen sind *K Anzahl der Klassen*, *L durchschnittliche Anzahl Labels je Beispiel*, *N Anzahl der Attribute*, $\hat{\delta}_{per}, \hat{\delta}_{IsErr} \leq 1, \hat{\delta}_{margin} < K$.

Speicheranforderungen gilt ebenfalls $\frac{K-1}{2}$. Eine Gegenüberstellung der Laufzeiten und Speicheranforderungen der Perzeptron-, MMP- und MLPC$_p$-Algorithmen ist in Tabelle 5.1 dargestellt.

Die Komplexität des MLPC$_p$-Algorithmus hängen somit im größeren Umfang von der Anzahl der Klassen ab als die des MMP-Algorithmus. Insbesondere bei den Speicheranforderungen gelangt der paarweise Ansatz bei einer großen Anzahl an Klassen und Attributen früh an seine Grenzen. Dies ist beispielsweise bei der Textklassifizierung der Fall. Die Verwendung des relativ sparsamen Perzeptron-Algorithmus erlaubt hierbei dennoch eine gute Einsatzfähigkeit. Der MLPC$_p$benötigt beispielsweise für die Hypothese bei einem Datensatz mit 100 Kategorien und 50.000 Attributen je Beispiel und 4 Byte je Dezimalzahl ca. 1 Gigabyte an Speicher, eine derzeit übliche Hauptspeichergröße. Für die Berechnungen hingegen ist der Textdatentyp von Vorteil. Wie bei der Komplexitätsanalyse des Perzeptrons erläutert, ist von der hohen Anzahl der Attribute bei Text nur ein Bruchteil belegt. Dies begünstigt die Berechnungen des Skalarprodukts, so daß die hohe Anzahl der Berechnungen beim Voting nicht so schwer ins Gewicht fällt.

Es stellt sich die Frage, ob der Austausch des Basislerners einen Vorteil insbesondere bei der Textklassifizierung mit sich bringen könnte. Interessant in diesem Zusammenhang ist die Erkenntnis, daß das Laufzeitverhältnis des Pairwise-Coupling-Verfahren zur One-against-all-Methode mit Erhöhung der Komplexität des Basislerner zugunsten des paarweisen Ansatzes ansteigt [Fürnkranz, 2002]. Der Grad der Komplexität bezieht sich dabei auf die Beziehung von Laufzeit und Anzahl der Beispiele. Die Verbesserung der Laufzeitkomplexität des paarweisen Ansatzes gegenüber der One-against-all-Methode ist intuitiv anhand eines Problems mit ausgeglichenen Klassen zu begreifen: ein Teilproblem besteht bei dem One-against-all-Ansatz aus P Beispielen, während das Pairwise-Coupling-Verfahren kleinere Probleme mit $\frac{2}{K}P$ erstellt. Insgesamt über alle Teilprobleme hinweg ist die Anzahl der Trainingsbeispiele in etwa gleich. Benötigt ein Basisklassifizierer nun eine zur Anzahl der Beispiele superlineare Laufzeit, so dauert das Training von P Instanzen beim One-against-all-Verfahren P^c, $c > 1$. Der paarweise Ansatz hingegen benötigt nur $\frac{K}{2}(\frac{2}{K}P)^c < P^c$, $K > 2$. Eine ausführlichere Betrachtung dieser Beziehung findet sich in [Fürnkranz, 2002].

Beim Pairwise-Coupling-Verfahren sollten allerdings keine asymmetrischen Algorithmen verwendet werden, die die Rechenzeit auf das Klassifizieren verlagern (z. B. Lazy-Algorithmen). Zudem sind superlineare Anforderungen an den Speicherplatz zu verhindern.

5.8 Varianten und alternative Ansätze

Die existierenden Varianten des Pairwise-Coupling-Verfahrens konzentrieren sich auf die Zusammenfügung der Einzelvorhersage. Es werden dabei zwei unterschiedliche Ziele verfolgt.

Das erste Ziel ist es, eine mögliche Verbesserung der Vorhersage durch Ausnutzen der Konfidenzen der Basisklassifizierer zu erreichen. Ein sehr simpler Ansatz hierfür stellt beispielsweise das Verfahren dar, das einen Basisklassifizierer nur zur Abstimmung zuläßt, wenn die Konfidenz seiner Vorhersage ein Mindestmaß erreicht [Fürnkranz, 2002]. Damit soll verhindert wissen, daß die erwähnten *unwissenden* Klassifizierer die Wahl zu stark beeinflussen.

Das zweite verfolgte Ziel ist die Reduktion der Anzahl der nötigen Vorhersagen, um eine schnellere Klassifizierung zu ermöglichen. Zunächst wird jedoch eine Modifikation vorgestellt, die bei asymmetrischen Lerner angebracht ist.

Double Pairwise-Coupling Beim *Double-Pairwise-Coupling* wird zusätzlich zu einem Klassifizierer $c^{u,v}$ sein Gegenpart $c^{v,u}$ angelernt [Fürnkranz, 2002]. Dies ist notwendig, wenn der Lernalgorithmus nicht symmetrisch ist, d. h. wenn die Invertierung der (binären) Klassifikation beim Training nicht die Invertierung der Vorhersage zur Folge hat. Dies ist beispielsweise bei *Covering-Regellernern* der Fall. Durch das doppelte Anlernen wird dieser Ungleichbehandlung entgegengewirkt, denn die Stimmen der gepaarten Klassifizierer neutralisieren sich, wenn sie die Klassifikation eines Beispiels unterschiedlich vorhersagen. Bei gleicher Abstimmung verdoppeln sich die Stimmen für eine Klasse, so daß diese Modifikation für symmetrische Verfahren wie beispielsweise dem Perzeptron-Algorithmus (siehe Abschnitt 3.3) bis auf die Verdopplung der Stimmen beim Voting keine Auswirkungen hat.

Voting-Against Cutzu [2003] stellt in seiner Untersuchung des Votings fest, daß die Vorhersage eines Basisklassifizierers korrekterweise (immer) nur als Gegenstimme gezählt werden müßte. Er erkennt, daß wenn ein Klassifizierer $c^{u,v}$ die Klasse y_u als Gewinner ausmacht, so kann man aus dieser Entscheidung im besten Fall davon ausgehen, daß y_v eine irrelevante Klasse darstellt. Folgendes Beispiel verdeutlicht intuitiv diesen Zusammenhang.

Wir betrachten wiederum den Klassifizierer $c^{u,v}$. Im einem ersten Fall gehört ein Beispiel der Klasse y_u an. Wenn die Vorhersage des Klassifizierers $\{y_u\}$ als Für-Stimme bewertet wird, verursacht dies beim Voting keinen Fehler. Wird die Stimme als Gegenstimme für y_u betrachtet, ist die Auswertung ebenfalls korrekt. In einem zweiten Fall gehört ein Beispiel der Klasse y_v an und der Klassifizierer sagt die falsche Klasse vorher. Beide Auszählungsstrategien scheitern, da der Basisklassifizierer selbst eine falsche Vorhersage abgegeben hat. In einem dritten Fall gehört ein Beispiel einer dritten Klasse y_k an. Unabhängig von dem Ausgang der Vorhersage, wird bei der Interpretation als Für-Stimme ein Fehler begangen. Die Auszählung als Gegenstimme für y_u oder y_v stellt hingegen keinen Fehler dar.

Diese Strategie liefert dasselbe Ergebnis wie das übliche Voting, wenn alle Basisklassifizierer eine Stimme abgeben. Stehen jedoch nicht alle möglichen Basisklassifizierer für die Vorhersagebildung zur Verfügung, findet dieses Verfahren eine korrektere Lösung. Für alle Varianten, die eine Reduktion der Anzahl der Vorhersagen anstreben, ist dieser Ansatz deshalb zu bevorzugen.

PKPD Die *PKPD-Methode*, benannt nach ihren Entwicklern Price, Knerr, Personnaz und Dreyfus [1995], nutzt im Gegensatz zum Voting die Konfidenzen der Basisklassifizierer aus, um eine genauere Gesamtvorhersage zu berechnen. Dieses Verfahren faßt dabei die Einzelkonfidenzen als probabilistische Vorhersage der Klassifizierer auf und kombiniert diese zu einer globalen Wahrscheinlichkeitsverteilung. Die Ausgabe ist somit für jedes Beispiel und jeder Klasse die Wahrscheinlichkeit der Klassenzugehörigkeit (siehe Abschnitt 2.4.1). In einem Vergleich von mehreren probabilistischen Verfahren für das Multiklassen-Pairwise-Coupling von Wu u. a. [2004] konnte die PKPD-Methode trotz ihrer Einfachheit überzeugen.

Das Prinzip der Methode ist wie soeben erwähnt, daß die Konfidenz $s^{u,v}$ der Vorhersage eines Klassifizierers als Wahrscheinlichkeitsschätzung aufgefaßt wird, d. h. $\Pr(y_u) = s^{u,v}$. Die Wahrscheinlichkeit für eine Klasse k wird durch Aufsummierung der Kehrwerte der Konfidenzen für k berechnet. Anschließend wird der Konfidenzenvektor normalisiert. Das Prinzip ist in der nächsten Gleichung dargestellt.

$$\bar{\mathrm{p}}^k = \sum_v \frac{1}{1 - s^{k,v}} \tag{5.8.1}$$

Klassen, die schlechte Konfidenzwerte erhalten, werden somit stärker bestraft. Dieses Verhalten besitzt deshalb Ähnlichkeit zu der Voting-Against-Strategie, die ebenfalls auf Bestrafungen aufbaut.

In den Vergleichen von Wu u. a. schnitt das Voting bei der Verwendung der Anzahl der Stimmen als Wahrscheinlichkeitsschätzung im Vergleich zu den probabilistischen Verfahren erwartungsgemäß schlecht ab. Dies wird klar, wenn man bedenkt, daß eine Klasse höchstens $K-1$ Stimmen erhalten kann und somit höchstens eine Wahrscheinlichkeit von $2/K$ für eine Klasse vorhergesagt werden kann. Es konnte jedoch beobachtet werden, daß keines der probabilistischen Ansätze entscheidende Vorteil gegenüber dem Voting bei der Vorhersage der positiven Klasse erzielen konnte. Dieses Ergebnis läßt vermuten, daß ein probabilistisches Verfahren auch bei der Vorhersage einer Multilabel-Klassifikation nicht entscheidend überlegen ist. Diese Vermutung wird näher in 6.5.1 untersucht.

Directed Acyclic Graphs Die Bewertung mithilfe von *Directed Acyclic Graphs* (*DAG*) stellt eine Methode für das Pairwise-Coupling dar, um die Anzahl der für die Bildung der Gesamtvorhersage verwendeten Basisklassifizierer zu reduzieren [Platt u. a., 1999]. Dabei wird ein Baum konstruiert, bei dem jeder innere Knoten einen Basisklassifizierer und jedes Blatt eine Klassenvorhersage darstellt. Das Verfahren läßt sich jedoch am einfachsten anhand der Analogie zur Intervallschachtelung erklären.

Wir setzen ohne Beschränkung der Allgemeinheit die initiale Klassenpaarung auf $[i, j] = [1, K]$. Nun wird die Vorhersage des Basisklassifizierers $c^{i,j}$ für ein Beispiel berechnet. Lautet die Klassifizierung y_i, so wird j um eins reduziert, d. h. die nächste Paarung wäre $[i, j-1]$. Bei einer Vorhersage von y_j wird umgekehrt i erhöht. Nach $K-2$ Vergleichen entscheidet die letzte Paarung $[i, i+1]$ über die Zuordnung.

Dieses Verfahren konnte gute Resultate im Vergleich zur klassischen One-against-all-Methode und zum Voting erbringen. Zusätzlich läßt sich eine Schranke für den Generalisierungsfehler angeben. Diese Schranke ist von der Anzahl der Klassen K und von dem Margin der Zweiklassen-Probleme abhängig, was unsere bisherigen Vermutungen nochmals bestätigt. Leider läßt sich dieses Verfahren nicht auf den Multilabel-Fall übertragen, da die Reihenfolge der ausgeschiedenen Klassen und

damit ein mögliches Ranking zu sehr von dem Aufbau des Baumes abhängt. Es sind keine Anpassungen bekannt, die dieses Problem lösen.

Andere Methoden zur Reduktion der Anzahl der nötigen Vorhersagen Eine vielversprechende Methode zur Reduktion der nötigen Klassifizierungen ist die Verwendung des Schweizer Turniersystems [Fürnkranz, 2002]. Dieses System wird besonders bei Schachturnieren verwendet, um aufgrund der großen Anzahl Mitspieler die Anzahl der Spielrunden zu beschränken. Hierbei trifft in jeder Runde ein Mitspieler auf einen anderen Mitspieler, der im Turnier bislang gleich erfolgreich gewesen ist (für Sieg, Patt und Niederlage gibt es jeweils $1, \frac{1}{2}$ und 0 Punkte). Es gewinnt derjenige Spieler, der nach der letzten Runde die meisten Punkte gewinnen konnte. Übertragen auf das Pairwise-Coupling-Verfahren entsprächen Klassen den Spielern und die Stimmabgaben den Schachspielen. Für die Auswertung der Stimmen würde sich hierbei das erwähnte Voting-Against anbieten. Bei einer Begrenzung der Runden auf beispielsweise $\log(K)$ ließe sich somit die Anzahl der Klassifizierungen auf $\frac{1}{2}\log(K)K$ reduzieren.

Eine andere Methode sieht vor, bei jedem Voting zufällig die Menge der Basisklassifizierer zu bestimmen, die abstimmen darf. Die Auswahl der Jury könnte beispielsweise anhand der A-priori-Klassenwahrscheinlichkeiten geschehen. Dabei ließe sich die Beeinflussung des Lerners durch unausgeglichene Klassenverhältnisse berücksichtigen. Es konnten keine Referenzen zu diesem Ansatz gefunden werden.

Für die Reduktion der Anzahl der benötigten Klassifizierer könnte auch ein möglicher hierarchischer Aufbau der Klassen genutzt werden. Eine Idee sieht vor, den Klassenbaum in gleich große Teilbäume aufzuteilen und nur die Klassifizierer für die Vorhersage zu verwenden, die innerhalb dieser Teilbäume diskriminieren. Der Grundgedanke dabei ist, daß die Stimme eines Klassifizierers, der zwischen zwei Klassen aus unterschiedlichen Teilbäumen unterscheidet, höchstens als Gegenstimme für einen der Teilbäume interpretiert werden kann, da die nahe Verwandschaft der Klassen eines Teilbaumes keine nähere Aussage zuläßt. Klassifizierer innerhalb der Teilbäume ziehen hingegen viel engere Grenzen zwischen den Klassen darin und eignen sich deshalb besser für die Vorhersage dieser Klassen. Es kann zudem erwartet werden, daß sich die Klassifizierer eines Teilbaumes, in dem sich eine relevante Klasse befindet, besser auf diese Klasse als Gewinner einigen können als die Klassifizierer eines anderen Teilbaumes auf einen Konkurrenten innerhalb dieses Baumes. Es bleibt zu untersuchen, ob diese Gedanken zutreffend sind. Es konnten keine ähnlichen Ansätze in der Literatur gefunden werden.

6 Versuchsergebnisse

Im folgenden Kapitel werden die durchgeführten Versuche zum MMP- und zum MLPC$_p$-Algorithmus. Es wird mit einer Beschreibung der verwendete Datensatzes begonnen. Anschließend wird beschrieben, wie die Grundeinstellung für die Versuche festgelegt wurden. In Abschnitt 6.4 werden beide Algorithmen miteinander verglichen und ihre Eigenschaften analysiert. Schließlich werden in einem letzten Abschnitt einige zusätzliche Versuche von Interesse vorgestellt.

6.1 Reuters-Datenbank

Das *Reuters Corpus Volume 1* (*RCV1*) ist ein großes Textarchiv, das im Jahre 2000 von der Nachrichtenagentur *Reuters* für die Forschung und Entwicklung zur Verfügung gestellt wurde.[1] Das Archiv stellt die Nachfolge des gut untersuchten Reuters-21578-Korpus dar.[2]

Es enthält insgesamt 806.791 journalistische Nachrichten in englischer Sprache, die im Zeitraum vom 20. August 1996 zum 19. August 1997 erstellt wurden. Eine Nachricht besteht meist aus etwa 1000 Wörtern, die Spanne reicht jedoch von wenigen 100 bis zu etwa 13.000 Wörtern.[3]

Im Jahre 2004 wurde eine überarbeitete Fassung von Lewis u. a. [2004] zur Verfügung gestellt, die Fehler in der Originalfassung behebt und bereits eine Vorverarbeitung der Texte enthält. Die sogenannte *RCV1v2* dient dieser Arbeit als Datengrundlage. Wenn nicht anders angegeben, wird bei der Erwähnung der Reuters-Datenbank Bezug auf die Version von Lewis u. a. genommen.

6.1.1 Klassifikation

Die Dokumentensammlung sind nach drei Gesichtspunkten unterteilt: nach Thema (*topic code*), nach der Beziehung zu einer Wirtschafts- oder Industriesparte (*industry code*) und nach dem geographischem Bezug (*region code*). Die zweite Zuordnung ist stark durch das Geschäftsmodell der Reuters-Nachrichtenagentur bestimmt, die ihre Nachrichten größtenteils an Kunden aus der Wirtschaft verkauft. Die dritte Zuordnung stellt eine einfache Multiklassen-Zuordnung dar. In der vorliegenden Arbeit beschränkt sich die Untersuchung deshalb auf die Zuordnung nach Themen. Dies entspricht einer natürlichen Kategorisierung von Texten.

Diese Zuordnung besteht aus 103 Kategorien, die in einer Hierarchie angeordnet sind. Die Richtlinien von Reuters sehen einerseits vor, daß ein Dokument mindestens einer Kategorie zugeordnet

[1] http://about.reuters.com/researchandstandards/corpus/,
http://trec.nist.gov/data/reuters/reuters.html
[2] http://www.daviddlewis.com/resources/testcollections/reuters21578/
[3] http://about.reuters.com/researchandstandards/corpus/statistics/index.asp

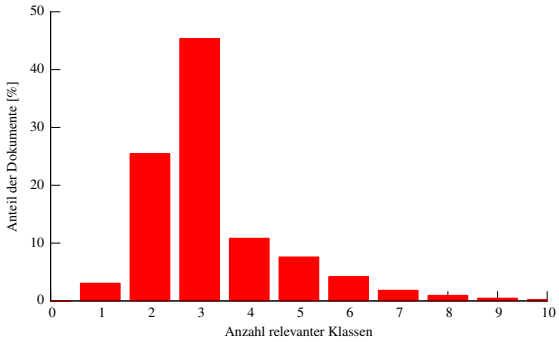


Abbildung 6.1: Verteilung der Anzahl der relevanten Klassen

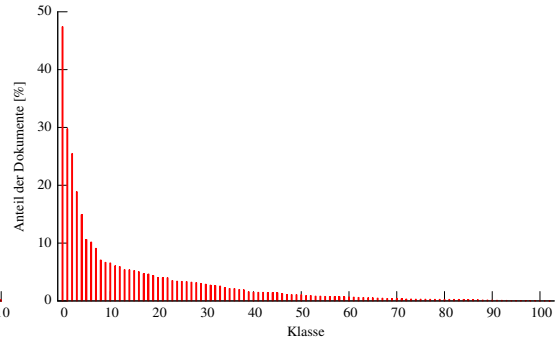


Abbildung 6.2: Größe der Kategorien. Die Klassen sind absteigend nach ihrer Größe numeriert.

wird, und andererseits, daß dem Dokument die treffendsten Kategorien zugeordnet werden (*Minimum Code Policy*), sowie alle Überkategorien in der Hierarchie (*Hierarchy Policy*). Es existiert keine Beschränkung in der Anzahl der Zuordnungen. Die Zuordnungen wurden entweder maschinell oder manuell durchgeführt und mindestens von einem Redakteur geprüft, so daß von einer weitgehend fehlerfreien und konsistenten Klassifikation ausgegangen werden kann. Im Durchschnitt ist ein Dokument 3,24 Kategorien zugeordnet, die Spanne reicht von einer relevanten Klasse bis zu dem Extrem von 17 Labels. Die Verteilung ist in Abbildung 6.1 dargestellt. Die Klasse mit den meisten Zuordnungen stellt die Kategorie *„Corporate/Industrial“* (*CCAT*) mit 381.327 Dokumenten dar. Sie gehört zusammen mit *„Economics“* (*ECAT*), *„Governement/Social“* (*GCAT*) und *„Markets“* (*MCAT*) zu den vier Hauptkategorien der Hierarchie. Die kleinste Kategorie stellt *„Millennium Issues“* (*GMIL*) mit fünf Dokumenten dar. Abbildung 6.2 gibt die unausgeglichene Verteilung der Klassen wieder. Die Kategorien sind in einer dreistufigen Hierarchie organisiert.

Bei der Zuordnung nach Themen handelt es sich demnach um eine Multiklassen-Klassifikation mit mehrfacher Zuordnung mit zusätzlichem hierarchischem Aufbau der Klassen (siehe Kapitel 2.2). Eine besondere Berücksichtigung der hierarchischen Organisation findet in dieser Arbeit nicht statt. Eine Möglichkeit dazu wurde in Abschnitt 5.8 genannt. In Abschnitt 6.5.2 wird jedoch ein Versuch unternommen, die Hierarchy-Policy-Richtlinie auszunutzen.

Die hohe Anzahl an Klassen und die hohe durchschnittliche Anzahl an Labels der Reuters-Datenbank stellt hohe Ansprüche an einen Multilabel-Lernalgorithmus. Zudem erlaubt die hohe Anzahl an Dokumenten eine Überprüfung der Laufzeit- und Speicheranforderungen hinsichtlich einer realen Einsatzfähigkeit. Die große Datenmenge ermöglicht überdies einen repräsentativen Vergleich weitgehend unabhängig von zufälligen Erscheinungen. Aus diesen Gründen wurde dieser Datensatz als Grundlage für die Versuche ausgewählt.

6.1.2 Datenvorverarbeitung

Bevor die von Reuters ausgehändigte Version der Datenbank zu Versuchszwecken genutzt werden kann, muß sie in einer bestimmten Form vorliegen. Hierfür wird auf die Vorarbeit von Lewis u. a. zurückgegriffen. Die einzelnen Schritte werden nun im Einzelnen genannt.

Korrekturen an RCV1 Von den ursprünglichen 806.791 Dokumenten wurden 2377 entfernt, da sie der Minimum Code Policy nicht entsprachen. Damit wurde die Anzahl an Dokumenten auf 804.414 reduziert. Zusätzlich wurden 25.402 Dokumenten zusätzlichen Kategorien zugeordnet, damit sie der Hierarchical Policy entsprachen.

Parsen Jeder Nachrichtentext liegt im ursprünglichen RCV1 in Form einer XML-Datei vor, die neben dem eigentlichen Text weitere Informationen wie Erstellungsdatum, eindeutige Identifikationsnummer, Sprache, Titeltext und Code-Zuweisungen enthält. Die Originalformatierung der Nachricht mit der Identifikationsnummer 477551 vom 31.03.97 ist exemplarisch in Abbildung 6.3 dargestellt. Für die Erstellung der RCV1v2-Version wurde der Text im Textelement (`<text>`) und in der Titelzeile (`<headline>`) verwendet. Auf diesem extrahierten Text wurde anschließend eine Tokenization (siehe Abschnitt 2.5.2) durchgeführt. Das so erhaltene Vokabular wurde durch eine Stoppwortreduktion und eine Grundformreduktion nach dem Porter-Stemmer-Algorithmus weiter reduziert. Eine ausführliche Beschreibung der einzelnen Schritte ist in [Lewis u. a., 2004, Abschnitt 7] zu finden. Das Resultat dieser Verarbeitung stellen die Autoren als sogenannte Token-Files zur Verfügung.[4] Exemplarisch ist in Abbildung 6.4 der Text aus der Nachricht in Abbildung 6.3 dargestellt.

Die Aufteilung in Trainings- und Testmenge von Lewis u. a. wurde rückgängig gemacht, so daß sich eine Folge von Dokumenten ergibt, die wie in der ursprünglichen Fassung chronologisch sortiert ist (die wegen Mißachtung der Minimum Code Policy entfernten Dokumente wurden nicht berücksichtigt). In dieser Version beträgt die Anzahl der in einem Dokument verwendeten unterschiedlichen Wörter im Durchschnitt 77,48. Im Vergleich dazu arbeiten Crammer und Singer [2003] mit einer Version mit 137 Wörtern je Dokument, die nicht einer Grundformenreduktion unterzogen wurde. Allerdings beträgt die Größe des Vokabulars 288.062, während die Version von Crammer und Singer nur 225.329 Wörter enthält. Dies ist wahrscheinlich auf die unterschiedliche Parse-Methode zurückzuführen. In der RCV1v2-Version befinden sich beispielsweise Zeichenketten wie *„hh9“*, *„v57“*, *„yasouk“* mit nur einem Vorkommnis. Etwa 118.062 Wörter besitzen eine Dokumenthäufigkeit von 1.

6.2 Implementierung

Für die Implementierung der Algorithmen wurde die *WEKA*-Umgebung als Grundlage verwendet [Witten und Frank, 1999]. Allerdings mußten aufgrund von Unzulänglichkeiten zahlreiche Anpassungen und Modifikationen durchgeführt werden. Die wichtigste Einschränkung von Weka für den in dieser Arbeit angestrebten Vergleich besteht darin, daß Multilabel-Daten nicht unterstützt werden. Eine weitere Schwäche der Weka-Umgebung ist die an einigen Stellen ineffektive Resourcennutzung. Aus diesen Gründen wurde die Umgebung um die Multilabel-Unterstützung und weitere Funktionalitäten in Form eines Pakets namens `weka.classifiers.multilabel` erweitert. Zusätzlich wurde eine effiziente Variante des Perzeptron nachimplementiert.

Für die Implementierung der MMP- und Multilabel-Pairwise-Coupling-Algorithmen ist zu beachten, daß bei den nötigen Sortierungen der effiziente Standard-Algorithmus von Weka verwendet

[4] [Lewis u. a., 2004, Online Anhang 12], `http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm`

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<newsitem itemid="477551" id="root" date="1997-03-31" xml:lang="en">
<title>SPAIN: Spain's Banesto issue $150 mln in subordinated loaN.</title>
<headline>Spain's Banesto issue $150 mln in subordinated loaN.</headline>
<dateline>MADRID 1997-03-31</dateline>
<text>
<p>Banco Espanol de Credito Banesto said on Monday it issued $150 million in subordinated 10-year 7.5 percent
debt. Lead manager is Lehman Brothers.</p>
<p>The statement added that this is the first international issue Banesto has launched since 1993.</p>
<p>Banco Santander has a 50 percent stake in Banesto.</p>
<p>– Madrid Newsroom, + 341 585 8340</p>
</text>
<copyright>(c) Reuters Limited 1997</copyright>
<metadata>
<codes class="bip:countries:1.0">
  <code code="SPAIN">
    <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1997-03-31"/>
  </code>
</codes>
<codes class="bip:industries:1.0">
  <code code="I81402">
    <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1997-03-31"/>
  </code>
</codes>
<codes class="bip:topics:1.0">
  <code code="C17">
    <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1997-03-31"/>
  </code>
  <code code="C172">
    <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1997-03-31"/>
  </code>
  <code code="CCAT">
    <editdetail attribution="Reuters BIP Coding Group" action="confirmed" date="1997-03-31"/>
  </code>
</codes>
<dc element="dc.date.created" value="1997-03-31"/>
<dc element="dc.publisher" value="Reuters Holdings Plc"/>
<dc element="dc.date.published" value="1997-03-31"/>
<dc element="dc.source" value="Reuters"/>
<dc element="dc.creator.location" value="MADRID"/>
<dc element="dc.creator.location.country.name" value="SPAIN"/>
<dc element="dc.source" value="Reuters"/>
</metadata>
</newsitem>
```

Abbildung 6.3: Nachricht im XML-Format: `477551newsML.xml`. Der extrahierte Text und die Topiccodes der Nachricht sind kursiv dargestellt.

```
brother issu issu issu subordin subordin stat million stak credit intern year debt manag percent
percent lead banc banc launch spain mln lehm espanol loan banest banest banest banest monday de
add madrid santand newsroom
```

Abbildung 6.4: Token-File. Nachricht aus Abbildung 6.3 nach der Tokenization, Stoppwortreduktion und Grundformreduktion.

wurde (`weka.core.Utils.sort()`, eine Quick-Sort-Implementierung). Dieser arbeitet instabil, d. h. bei Gleichheit bleiben die Elemente (d. h. die Klassen) nicht notwendigerweise in ihrer ursprünglichen Reihenfolge. Da insbesondere beim Voting des MLPC der Fall eines Unentschiedens nicht gesondert berücksichtigt wurde, ist das Verhalten in diesem Punkt nicht deterministisch. Allerdings dürfte dieser Fall bei der hohen Anzahl der Klassen für das Gesamtergebnis einen vernachlässigbaren Unterschied ausmachen.[5] Dies beeinflußt zudem nicht das Training und damit die Hypothesenbildung, da eine Sortierung nur für die Bewertung der Klassifizierung notwendig ist.

Im folgenden wird kurz der Aspekt der Initialisierung besprochen. Bei der Implementierung des MMP-Algorithmus werden die Perzeptrons mit dem Nullvektor initialisiert. Die dadurch entstehende Problematik bei dem Proportional-Update (siehe Abschnitt 4.2.2) wird dadurch gelöst, daß für die Berechnung nicht die Skalarprodukte verwendet werden, sondern die normalisierten Konfidenzwerte (siehe Gleichung 2.4.1). Das Ergebnis ist bei der ersten Aktualisierung ein Verhalten wie bei der Uniform-Update-Methode. Für die sonstige Funktionsweise stellt die Verwendung der normalisierten Konfidenzwerte keine Änderung dar. Die Perzeptrons beim MLPC_p-Algorithmus werden mit gleichverteilten Zufallszahlen aus dem kleinen Intervall $[-10^{-4}, 10^{-4}]$ initialisiert (vgl. Abschnitt 3.4).

6.3 Versuchsvorbereitung

Der erste Schritt bestand darin, aus den in Abschnitt 6.1.2 beschriebenen Daten in Form von Tokens einen Datensatz zu erstellen, der als Grundlage für die durchzuführenden Versuche dienen sollte. Zunächst wurden hierfür die Daten in die Vektorraumdarstellung nach dem *bag-of-words*-Modell überführt (siehe Abschnitt 2.5.3.2).

Der zweite Schritt bestand darin, die Anzahl der verwendeten Attribute mithilfe einer Feature-Selection zu beschränken. Für diesen Prozess wurden die Attribute nach der Dokumenthäufigkeit geordnet, wie in Abschnitt 2.5.6 beschrieben, da dies zunächst die einfachste Methodik darstellte. Für die Gewichtung der Wörter wurde die einfache und oft eingesetzte Salton-Methode verwendet (siehe Abschnitt 2.5.3.2).

An dieser Stelle sei vorweggenommen, daß für die Ermittlung der Dokumenthäufigkeit und aller darüber hinausgehenden statistischen Werte, die für die verschiedenen Methoden der Feature-Selection und der Wortgewichtung notwendig sind, der komplette Datensatz betrachtet wurde. Dies widerspricht der Forderung, daß diese Werte nur aus der Trainingsmenge ermittelt werden dürfen, da sonst Informationen über die noch unbekannte Testmenge in die Bildung der Hypothese einfließen würde. Auf diese Forderung wurde aus zwei Gründen verzichtet. Erstens war es an dieser Stelle des Prozesses unmöglich, eine definitive Entscheidung über die Aufteilung in Trainings- und Testmenge zu treffen, da Erkenntnisse über die realen Laufzeit- und Speicheranforderung der Algorithmen fehlten. Und da jeder Konvertierungsprozess, d. h. die Umwandlung in die Vektorraumdarstellung mit der damit verbundenen notwendigen Festlegung der Reihenfolge der Attribute, erhebliche Zeit in Anspruch nahm (siehe Abschnitt 6.3.4), ermöglichte die globale Ermittlung

[5] Die Verwendung des stabilen Sortier-Algorithmus hätte die Klasse mit der höheren A-priori-Wahrscheinlichkeit begünstigt, da die Klassen intern nach ihrer Größe geordnet sind

der Werte eine effiziente und dynamische Anpassung an noch unbekannte Anforderungen. Zweitens kann auf die Bedingung verzichtet werden, wenn nur ein Vergleich zwischen verschiedenen Algorithmen angestrebt wird, wie es im Rahmen dieser Arbeit der Fall ist.

Für die initiale Bewertung des gewählten Datensatzes, d. h. der Kombination aus Feature-Selection-Methode, Anzahl der Attribute und Wortgewichtung, wurde der MMP-Algorithmus in der Variante des Uniform-Updates und der IsError-Lossfunktion gewählt. Der Grund hierfür ist, daß diese Parameter bei den Untersuchungen von Crammer und Singer [2003] auf der großen Reuters-Datenbank die besten Resultate erzielten (siehe Abschnitt 4.5). Zusätzlich wurde der MLPC$_p$-Algorithmus mit dem Perzeptron ohne Schwellwert als Basislerner verwendet. Diese Einstellung wurde gewählt, da es der Variante des Perzeptrons entspricht, die im MMP verwendet wird. Ein entscheidender Unterschied ist bei den Dimensionen der vorliegendem Daten nicht zu erwarten. In Abschnitt 6.3.6 findet eine gesonderte Betrachtung des Perzeptrons ohne Schwellwert statt.

Für den Vergleich der Einstellung wird vorwiegend der durchschnittliche IsError-Fehler auf den Testdaten verwendet. Dieser Wert hat den Vorteil, daß er intuitiv gut interpretierbar ist, da er direkt den Anteil der nicht perfekt klassifizierten Testbeispiele repräsentiert (siehe auch Abschnitt 2.4.2). Aus diesem Grund wird der IsError-Fehler im nachfolgenden für die Darstellung mit Hundert multipliziert und gibt somit den Anteil der Beispiele in Prozent an, die nicht perfekt klassifiziert wurden. Für die initialen Untersuchungen auf dem Datensatz ist IsError zudem von Bedeutung, da sich die für die Beurteilung verwendete MMP-Variante für die Aktualisierung nach diesem Fehler richtet und deshalb der Algorithmus nach diesem Wert beurteilt werden sollte. Weitere wichtige Werte für die initiale Bewertung stellen der ErrorSetSize-Loss und der OneError-Loss dar. Die auf Recall und Precision aufbauenden Losses AvgP und MaxF1 werden für die Ergebnisse der Anfangsphase nicht angegeben, da sie sich durch die verwendeten Berechnungstechniken nur geringe Unterschiede aufweisen und eine Beurteilung dadurch erschweren. Für die drei zuletzt genannten Werte gilt im übrigen auch die Konvention, daß ihre Beträge zum besseren Verständnis mit Hundert multipliziert werden. Der in dieser Arbeit vorgestellte Margin-Loss spielte in der initialen Phase keine Rolle, da er erst später implementiert wurde. Die Verwandschaft zum ErrSetSize-Loss läßt jedoch eine parallele Entwicklung vermuten.

6.3.1 Aufteilung in Trainings- und Testmenge

Als Aufteilung in Trainings- und Testmenge wurde auf Grundlage der Aufspaltung in [Crammer und Singer, 2003] für die Trainingsmenge die chronologisch ersten 521.439 Dokumente verwendet und für die Testmenge die restlichen 282.975 Dokumente. Dies entspricht etwa der üblichen $\frac{2}{3}$-$\frac{1}{3}$-Spaltung (siehe Abschnitt 2.3.2). Das letzte Trainingsdokument besitzt die Idendifikationsnummer 527.201 und wurde am 22.04.1997 erstellt. Die chronologische Ordnung der Dokumente entspricht den Bedingungen in einer realen Umgebung und wurde deshalb beibehalten.

6.3.2 Anzahl der Attribute

Für die Ermittlung der günstigsten Einstellung für die Anzahl der Attribute wurden die nach Dokumenthäufigkeit geordneten Wörter verwendet. Die in Abschnitt 6.2 vorgestellte Erweiterung der Weka-Umgebung erlaubt eine effiziente Auswahl der ersten N Attribute, was die Untersuchungen in diesem Bereich erleichterten. Für die Wortgewichtung wurde wie bereits erwähnt die Salton-

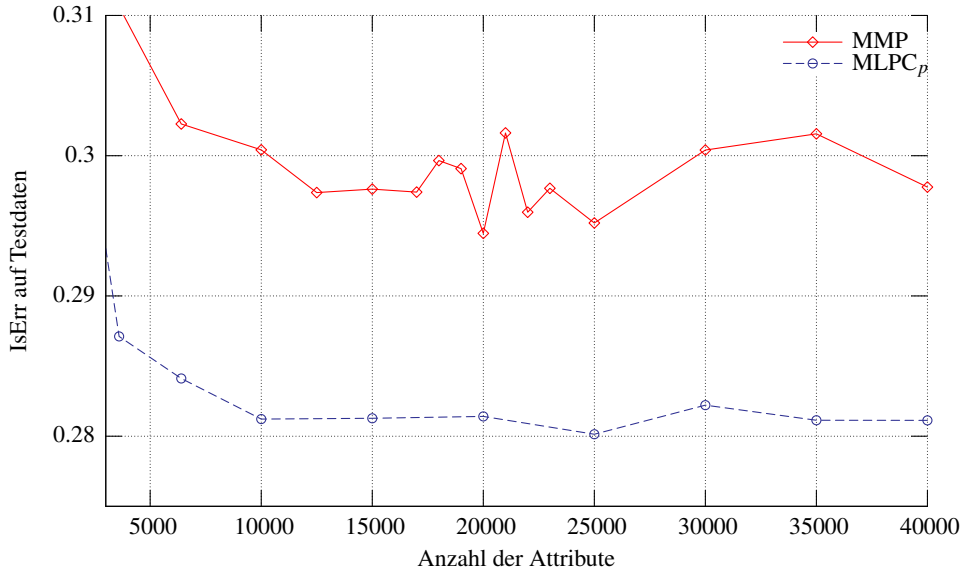


Abbildung 6.5: Feature-Selection

Methode verwendet. Die Versuche decken den gesamten Bereich von 100 bis zu den kompletten ca. 288.000 Attributen ab. Leider war ab 40.000 Attributen eine Durchführung mit dem MLPC$_p$-Algorithmus aufgrund von Speicherbeschränkungen nicht mehr möglich. Der IsError-Fehler abhängig von der Anzahl der verwendeten Attribute ist in Abbildung 6.5 dargestellt.

Eine interessante Beobachtung ist die relative Unempfindlichkeit beider Algorithmen gegenüber der ansteigenden Anzahl Attribute. Dies unterstreicht die vollständige Darstellung der Kurve des MMP-Algorithmus in Abbildung 6.9. Dadurch wird die in Kapitel 3.5 und 3.10 geäußerte Vermutung bestätigt, daß der Perzeptron-Algorithmus für hochdimensionale Daten geeignet ist.

Die Schwankungen des MMP-Algorithmus lassen eine allgemeingültige, optimale Einstellung der Anzahl der Attribute für diesen nicht zu. Da jedoch beide Algorithmen bei 25.000 Attributen ein lokales Minimum für den IsError-Fehler besitzen, wurde dieser Wert als Grundeinstellung festgelegt.

Ausgehend von dieser Entscheidung konnten nun die anfangs getroffenen Festlegungen der Wortgewichtungsmethode und der Feature-Selection-Methode überprüft werden.

6.3.3 Wortgewichtung

Für die Wahl der Wortgewichtungsmethode wurde neben der in Abschnitt 2.5.3.2 vorgestellten Salton-Methode die Singhal-Methode getestet, die als eine der effektivsten Wortgewichtungsmethoden gilt [nach Crammer und Singer, 2003, Abschnitt 9.2]. Als Referenz wurde zusätzlich die simple Worthäufigkeit eines Wortes als Wortgewichtung verwendet. Im folgenden werden einige Definitionen wiederholt bzw. für die Berechnung der Singhal-Methode neu eingeführt.

Sei $n(t_i, w_j)$ die Anzahl der Vorkommnisse von Wort w_j in Text t_i. Die Dokumenthäufigkeit sei gegeben durch es gilt $\mathrm{df}(w_j) = |\{t_i |\ w_j \in t_i\}|/P$. Die inverse Dokumenthäufigkeit sei folglich

		$MLPC_p$	Δ [%]	MMP	Δ [%]
IsError	Salton	27,96	0,07	29,24	–
	Singhal	27,94	–	29,55	1,05
	TF	29,92	7,11	32,02	9,49
ErrSetSize	Salton	1,92	0,21	2,87	1,65
	Singhal	1,91	–	2,82	–
	TF	2,15	12,58	3,04	7,63
OneError	Salton	2,95	–	3,82	2,86
	Singhal	3,01	1,72	3,71	–
	TF	3,41	15,38	4,36	17,42

Tabelle 6.1: Vergleich der Term-Weighting-Methoden. TF steht für die Verwendung der Worthäufigkeit. Ein Strich kennzeichnet die beste Methode.

$\text{idf}(w_j) = \log(1/\text{df}(w))$. Sei m_i die Anzahl der unterschiedlichen Wörter im Dokument t_i, d. h. $m_i = |\{w_j \mid n(t_i, w_j) > 0\}|$. Die durchschnittliche Anzahl der unterschiedlichen Wörter in allen P Texten sei gegeben durch $\hat{m} = \frac{1}{P}\sum_i m_i$. Zudem bezeichne $\hat{n}_i$ die durchschnittliche Worthäufigkeit in einem Dokument t_i, d. h. $\hat{n}_i = |t_i|/m_i$. Dann lassen sich die Berechnungen der Wortgewichte nach den genannten Methoden folgendermaßen angeben.

$$\text{tw}_{tf}(t_i, w_j) := n(t_i, w_j) \tag{6.3.1}$$

$$\text{tw}_{salton}(t_i, w_j) := \text{idf}(w_j) \cdot \frac{n(t_i, w_j)}{|t_i|} \tag{6.3.2}$$

$$\text{tw}_{singhal}(t_i, w_j) := \text{idf}(w_j) \cdot \frac{(1+\log(n(t_i, w_j)))/(1+\log(\hat{n}_i))}{1.0 - slope + slope \cdot (m_i/\hat{m})} \tag{6.3.3}$$

Die Variable *slope* ist eine Einstellgröße zwischen 0 und 1. Es wurde wie bei den Untersuchungen des MMP von Crammer und Singer ein Wert von 0,3 gewählt. Wie zuvor erwähnt, wurde für die Berechnung der für die Methoden notwendigen globalen Konstanten die komplette Datenmenge verwendet.

In Tabelle 6.1 sind die Ergebnisse der Untersuchung aufgezeigt. Der Fehler ist bezogen auf die Testmenge. Wie erwartet fallen die Ergebnisse für die Verwendung der Worthäufigkeit am schlechtesten aus. Für die Verwendung der aufwendigeren Singhal-Methode zeigt sich kein klarer Vorteil. Es wurde aus diesem Grund die einfache Salton-Methode als Grundeinstellung für zukünftige Versuche gewählt.

6.3.4 Feature-Selection-Methode

Neben der bereits durchgeführten Feature-Selection nach der Dokumenthäufigkeit, wurden die Ansätze des Informationsgewinns eines Attributs und des χ^2-Maßes umgesetzt, die in Abschnitt 2.5.6 vorgestellt wurden.

Für die Ordnung der Attribute nach dem Informationsgewinn (IG) wurde ein Algorithmus aus [Dinnesen und Andersen, 2005] verwendet, der wiederum eine Erweiterung eines Algorithmus in [Soica und Hearst, 2004] darstellt. Bei diesem Verfahren wird in jeder Runde das Wort mit der höchsten

		MLPC_p	Δ [%]	MMP	Δ [%]
IsError	IG	27,88	–	29,17	–
	DF	27,96	0,26	29,24	0,27
	χ^2	28,29	1,45	29,44	0,95
ErrSetSize	IG	1,90	–	2,86	–
	DF	1,92	0,58	2,87	0,17
	χ^2	1,96	2,66	2,92	2,09
OneError	IG	3,00	1,70	3,76	–
	DF	2,95	–	3,82	1,49
	χ^2	3,10	4,80	3,85	2,36

Tabelle 6.2: Vergleich der Methoden der Feature-Selection. Die Δ-Spalte gibt den prozentualen Unterschied zur Methode mit dem niedrigsten Loss an. Ein Strich kennzeichnet die beste Methode.

Dokumenthäufigkeit $\text{df}(w_i)$ aus dem Vokabular $\mathcal{W}$ entfernt. Anschließend wird der Datensatz um all jene Dokumente reduziert, die das entfernte Wort enthalten. Die Dokumenthäufigkeiten werden anhand der reduzierten Datenmenge erneut berechnet und der Prozess fortgeführt, bis der ursprüngliche Datensatz leer ist. Da hierdurch das Verfahren bereits nach einer geringen Anzahl Attributen beendet ist, schlagen Dinnesen und Andersen vor, den Prozess an diesem Punkt mit dem vollständigen Datensatz fortzusetzen, bis alle Wörter entfernt wurden. Die neue Ordnung der Attribute, d. h. der Wörter, entspricht der Reihenfolge, in der die Wörter entnommen wurden.

Für die Ordnung der Attribute nach dem χ^2-Maß wurde die χ^2*_max score*-Methode nach Yang und Pedersen [1997], Rogati und Yang [2002] verwendet, die schon bei den Untersuchungen des RCVv2 von Lewis u. a. zum Einsatz kommt. Hierfür wird für jede Attribut-Klasse-Kombination der χ^2-Wert nach folgender Formel berechnet.

$$\chi^2_{ij} = \frac{n(ad-bc)}{(a+b)(a+c)(b+d)(c+d)} \tag{6.3.4}$$

Dabei stellt n die Anzahl der für die Ermittlung der Daten verwendeten Dokumente dar. Der Wert a ist die Anzahl der Dokumente, in denen das Wort w_i auftaucht und die der Klasse y_j zugeordnet sind, b ist die Anzahl der Dokumente mit dem Wort und nicht in der Klasse, c umgekehrt die Anzahl der Beispiele ohne w_i aber in y_j, und schließlich ist d die Anzahl der Dokumente, die weder w_i enthalten noch y_j zugeordnet sind. Für jedes Attribut wird nun der höchste χ^2-Wert über alle Klassen ermittelt und absteigend nach diesem sortiert, d. h. nach $\chi^2_i\_max = \max_j \chi^2_{ij}$.

Die in Tabelle 6.2 dargestellten Werte spiegeln den Fehler auf der Testmenge bei Verwendung der ersten 40.000 Attributen wieder.

Die Ordnung nach dem χ^2-Maß erzielt in allen Fällen die schlechteste Bewertung. Der Unterschied zwischen der Ordnung nach dem Informationsgewinn und der Dokumenthäufigkeit ist hingegen minimal. Dies bestätigt die Ergebnisse aus den Untersuchungen zur Feature-Selection von Yang und Pedersen [1997].

Da der vorgestellte Algorithmus für die Ordnung nach dem Informationsgewinn sehr ineffizient arbeitet und zudem keine Verbesserung gegenüber der einfachen Sortierung nach der Dokumenthäu-

Parameter		Methode der Fehlermessung					
Update	Loss-Funktion	IsErr	ErrSetSize	Margin	OneErr	AvgP	MaxF1
Uniform	IsError	**29,24**	2,87	2,16	3,82	**92,80**	**93,03**
	ErrSetSize	31,91	**2,65**	1,92	4,30	92,14	92,35
	Margin	30,99	2,68	**1,90**	4,14	92,33	92,54
	OneError	40,57	5,16	4,02	4,09	89,71	89,94
	AvgP	29,96	2,95	2,10	**3,67**	92,63	92,79
	MaxF1	29,95	2,89	2,08	3,77	92,68	92,86
Random	IsError	29,43	3,08	2,23	3,84	92,65	92,85
	ErrSetSize	32,65	2,78	1,96	4,46	91,92	92,15
	Margin	31,13	2,70	1,91	4,00	92,37	92,53
	OneError	40,79	5,14	3,81	4,11	89,62	89,76
	AvgP	30,55	3,05	2,16	3,69	92,53	92,64
	MaxF1	30,12	2,95	2,11	3,76	92,66	92,82
Prop	IsError	30,79	3,44	2,61	4,14	92,21	92,52
	ErrSetSize	33,43	3,10	2,20	4,75	91,43	91,72
	Margin	32,25	3,09	2,18	4,70	91,72	92,09
	OneError	41,31	6,41	4,64	4,68	88,93	89,23
	AvgP	32,03	3,55	2,55	4,18	91,85	92,12
	MaxF1	31,69	3,49	2,55	4,22	91,94	92,25
Max	IsError	33,23	6,36	4,78	4,58	90,81	91,39
	ErrSetSize	47,16	9,59	7,41	10,98	85,44	87,14
	Margin	44,77	10,29	7,90	9,59	86,16	87,75
	OneError	49,26	16,41	11,95	6,32	84,53	85,92
	AvgP	37,24	8,58	6,50	5,41	89,19	90,01
	MaxF1	36,21	7,94	6,05	5,40	89,59	90,39

Tabelle 6.3: Vergleich der MMP-Parameter. IsErr, OneError, AvgP und MaxF1 wurden jeweils mit 100 multipliziert. Werte in Fettschrift geben das beste Ergebnis für die jeweilige Fehlermessung an.

figkeit darstellt, wurde als Grundeinstellung für alle weiteren Versuche die Ordnung der Attribute nach der Dokumenthäufigkeit gewählt.[6]

6.3.5 Einstellungen des MMP-Algorithmus

In den vorangegangenen Versuchen wurde der MMP-Algorithmus mit der Uniform-Update-Methode und der IsError-Funktion verwendet. Diese Entscheidung wurde wie erwähnt auf Grundlage der Versuche von Crammer und Singer [2003] getroffen, bei denen für diese Kombination die

[6] Die Berechnung wurde auf einem Computer mit zwei AMD Opteron 250/850 Prozessoren mit jeweils 2,4 GHz und 2 GB Arbeitsspeicher durchgeführt. Hierfür benötigte der Algorithmus 65 Minuten CPU-Zeit und 1,9 GB Speicher. Der hohe Speicherverbrauch (bedingt durch die notwendige Verfügbarkeit aller Beispielvektoren im Speicher) hatte zudem zur Folge, daß der Arbeitsspeicher oft auf die Festplatte ausgelagert werden mußte, so daß die Realzeit 110 Minuten betrug.

Variante	IsErr	Δ [%]	ErrSet	Δ [%]	OneErr	Δ [%]	AvgP	Δ [%]	MaxF1	Δ [%]
P_0	27,94	–	1,91	0,30	2,95	–	93,71	–	93,65	–
P_ϑ	28,06	0,44	1,91	–	3,00	1,55	93,71	–	93,65	–

Tabelle 6.4: Vergleich der Varianten des Perzeptron-Algorithmus für die Nutzung durch den MLPC$_p$-Algorithmus. P_0: Perzeptron ohne Schwellwert, P_ϑ: Perzeptron mit Schwellwert

besten Ergebnisse beobachtet werden konnten. Da der hier verwendete Datensatz aus der Reuters-Datenbank unterschiedlich ist und damit die Versuchsbedingungen abweichen, wurden die Versuche wiederholt. Zusätzlich wurde das im Rahmen dieser Arbeit vorgestellte Maß Margin-Loss sowohl für die Aktualisierung als auch für die Bewertung verwendet. Es ergeben sich somit 24 mögliche Parameter-Kombinationen. In Tabelle 6.3 sind die Ergebnisse für die Evaluierung auf der Testmenge angegeben.

Es zeigt sich insgesamt ein ähnliches Bild wie bei den Beobachtungen von Crammer und Singer. Die Uniform-Update-Methode überragt, jedoch bildet sich die allgemeine Überlegenheit der IsError-Lossfunktion nicht dermaßen deutlich heraus. Insgesamt ist die Beziehung zwischen der bei der Aktualisierung verwendeten Loss-Funktion und der für die Beurteilung verwendeten Loss-Funktion ausgeprägter. Es zeigt sich zudem die enge Verwandtschaft zwischen ErrorSetSize und Margin-Loss. Dabei scheint der Margin-Loss für die Beurteilung nach dem IsError-Fehler (und im geringeren Ausmaß nach dem OneError-Fehler) eine klare Verbesserung gegenüber dem ErrSetSize-Loss darzustellen.

Da trotz der nicht allgemeingültigen Aussage zur günstigsten Wahl der Loss-Funktion das IsError-Maß eine intuitive Bewertung zuläßt, wird als Grundeinstellung des MMP-Algorithmus für die weiteren Versuche die Uniform-Update-Methode zusammen mit der IsError-Lossfunktion gewählt.

6.3.6 Einstellungen des MLPC$_p$

Bei dem MLPC$_p$-Algorithmus sind nur zwei mögliche Varianten möglich, die sich aus den möglichen Perzeptrons ergeben. Es können entweder Perzeptrons mit oder ohne Schwellwert verwendet werden (siehe Abschnitt 3.8.1). Ein Vergleich findet sich in Tabelle 6.4.

Ein signifikanter Unterschied zwischen den Varianten ist, wie zu erwarten war, nicht registrierbar. Allerdings ist bei einer anderen Problemstellung als der Textklassifizierung ein Unterschied möglich. Für niedrig-dimensionale Daten, wie Punkte in der euklidischen Ebene oder im dreidimensionalen Raum, ergeben sich erhebliche Nachteile für die Variante ohne Schwellwert (vgl. Abschnitt 3.6).

Für den hier angestrebten Vergleich jedoch ist die Verwendung eines Schwellwerts nicht relevant. Da der MMP-Algorithmus (als Repräsentant der One-against-all-Methode) nach der ursprünglichen Beschreibung in [Crammer und Singer, 2003] und in der hier verwendeten Implementierung Perzeptrons ohne Schwellwert verwendet, wurde für den MLPC$_p$-Algorithmus (als Repräsentant des Pairwise-Coupling-Verfahrens) ebenfalls diese Variante des Perzeptron-Algorithmus als Grundeinstellung verwendet.

Einstellung	Wert
Größe der Trainingsmenge	521.439 Dokumente
Größe der Testmenge	282.975 Dokumente
Feature-Selection-Methode	Dokumenthäufigkeit
Anzahl der verwendeten Attribute	25.000
Wortgewichtungsmethode	Salton
Aktualisierungsmethode des MMP	Uniform-Update
Loss-Funktion des MMP	IsError-Loss
Basislerner des MLPC$_p$	Perzeptron ohne Schwellwert
Bewertung der Einzelvorhersagen im MLPC$_p$	Voting

Tabelle 6.5: Grundeinstellungen für die Versuche

6.3.7 Grundeinstellungen

Aus den durchgeführten Tests ergeben sich für weitere Versuche die folgenden Grundeinstellungen: für die Feature-Selection werden die Wörter anhand ihrer Dokumenthäufigkeit geordnet und nur die ersten 25.000 genutzt. Die Gewichtung der Wörter in der Vektorrepräsentation wird mithilfe der Salton-Methode berechnet. Die Feature-Selection und die Wortgewichtung ist unabhängig von der Wahl der Aufteilung der Trainings- und Testmenge, da der gesamte Datensatz als Grundlage für die Berechnungen verwendet wurde. Für die Aufteilung gilt, daß die chronologisch ersten 521.439 Dokumente aus der RCVv2-Datenbank als Trainingsmenge genutzt werden und die restlichen 282.975 Dokumente zur Bewertung. Für den MMP-Algorithmus wird die Uniform-Update-Methode kombiniert mit der IsError-Lossfunktion als Parameter ausgewählt. Der MLPC$_p$-Algorithmus verwendet das Perzeptron ohne Schwellwert als Basislerner. Eine Zusammenstellung der Grundkonfiguration findet sich in Tabelle 6.5.

6.4 Untersuchungen zum MMP und MLPC$_p$

In diesem Abschnitt werden die unterschiedlichen Eigenschaften des MMP-Algorithmus und des MLPC$_p$-Algorithmus untersucht. Es wird mit einem direkten Vergleich beider Algorithmen mit der zuvor vorgestellten Grundeinstellung begonnen. In einem weiteren Versuch wird die Lernrate beider Algorithmen vorgestellt. Anschließend wird das Konvergenzverhalten und ein mögliches Overfitting untersucht, indem die Algorithmen in mehreren Iterationen die Trainingsdaten lernen.

6.4.1 Direkter Vergleich

Für den direkten Vergleich wurden die in Tabelle 6.5 dargestellten Einstellungen verwendet. Die Resultate der Evaluierung auf den Testdaten befinden sich in Tabelle 6.6. Die mittlere Spalte gibt den prozentualen Unterschied der Werte des MLPC$_p$-Algorithmus zu den Werten des MMP-Algorithmus an. Dieser wurde wie folgt berechnet: $\Delta = (Loss_{MLPC_p}/Loss_{MMP} - 1) \cdot 100$. Für IsError, ErrorSetSize, Margin und OneError stellen demnach negative Werte einen Vorteil für den MLPC$_p$-Algorithmus dar, für AvgP und MaxF1 umgekehrt positive Werte (vgl. Abschnitt 2.4.2).

	MMP	Δ [%]	MLPC$_p$
IsError	29,24	-4,40	27,96
ErrSetSize	2,87	-33,19	1,92
Margin	2,16	-33,05	1,45
OneErr	3,82	-22,69	2,95
AvgP	92,80	0,98	93,71
MaxF1	93,03	0,67	93,65

Tabelle 6.6: Direkter Vergleich zwischen MMP und MLPC$_p$auf den Testdaten. Die Spalte in der Mitte gibt die prozentuale Verbesserung für den MLPC$_p$-Algorithmus zum MMP-Algorithmus wieder.

	MMP	MLPC$_p$
ErrSetSize	2,87	1,92
ErrSetSize*	9,81	6,85
Margin	2,16	1,45
Margin*	7,40	5,18
Margin[%]	2,12	1,42
Margin*[%]	7,25	5,08

Tabelle 6.7: Vergleich der hergeleiteten Fehler. ErrSetSize* bzw. Margin* geben den Fehler auf den nicht perfekt klassifizierten Beispielen wieder (vgl. Gleichung (2.4.9)). Margin[%] gibt zusätzlich den Anteil am Maximalfehler (K-1) an .

Es zeigt sich, daß der im Rahmen dieser Arbeit entwickelte MLPC$_p$-Algorithmus den MMP-Algorithmus für alle Fehlermaße übertrifft. Für den wichtigen IsError-Loss kann der MLPC$_p$-Algorithmus einen Vorsprung von 4,4% verzeichnen. Es zeigt sich jedoch besonders bei den Fehlermaßen ErrorSetSize- und Margin-Loss eine deutliche Überlegenheit. Diese messen den Grad der Schwere eines Rankingfehlers, d. h. die Anzahl der Fehler in der Reihenfolge der Klassen. Hierbei scheint sich die Architektur des paarweisen Ansatzes besonders auszuzahlen. Dies kann folgendermaßen interpretiert werden.

Mit höherer Anzahl an relevanten Klassen eines Beispiels steigt die Schwierigkeit, ein perfektes Ranking zu finden. Deshalb ist der akkumulierte IsError-Fehler der MMP- und MLPC$_p$-Algorithmen vorwiegend Beispielen mit einer hohen Anzahl an Labels zuzurechnen. Aus diesem Grund ähneln sich beide Verfahren bei diesem Fehlermaß am ehesten. Da es sehr unwahrscheinlich ist, für Beispiele mit einer hohen Anzahl Labels ein perfektes Ranking vorherzusagen, ist es an dieser Stelle von Bedeutung, die Anzahl der Mißplatzierungen zu minimieren. Dies hängt beim MMP-Algorithmus entscheidend von den Abständen zwischen den Konfidenzen (d. h. den Skalarprodukten) ab. Um diese zu beeinflussen steht dem Verfahren jedoch nur in dem Maß Freiraum zur Verfügung, wie es den einzelnen Prototypen für das Positionieren der trennenden Hyperebene zur Verfügung steht. D. h. letztendlich, daß eine geringe Abweichung von der nötigen Konfidenz große Auswirkungen auf das Ranking hat, da die Konfidenzen nah beieinander sind. Bei dem paarweisen Ansatz ist jedoch ein einzelner Fehler eines Basisklassifizierers besser zu verkraften, da insgesamt $K-1$ Klassifizierer je Klasse für die Bewertung zuständig sind. Die Abstände in den Konfidenzen sind demnach größer.

Dies veranschaulicht auch Abbildung 6.6, die den durchschnittlichen IsError- und Margin-Fehler in Abhängigkeit von der Anzahl der Labels der Beispiele zeigt (vgl. dazu die Verteilung der Anzahl der Labels in Abbildung 6.1). Tabelle 6.7 gibt in diesem Zusammenhang den durchschnittlichen ErrorSetSize- und Margin-Fehler auf den nicht perfekt klassifizierten Daten an. D. h. der akkumulierte Fehler geteilt durch die Anzahl der nicht perfekt klassifizierten Beispiele, da der Fehler auf den restlichen Beispielen Null ist. Diese Maße sind mit einem Sternchen gekennzeichnet und werden von dem IsError-Fehler hergeleitet (vgl. Gleichung (2.4.9) in Abschnitt 2.4.2)

Für den AvgP- und MaxF1-Fehler kann kein großer Unterschied festgestellt werden. Dies muß allerdings im Zusammenhang mit der geringen Varianz dieser Maße gesehen werden, die sich

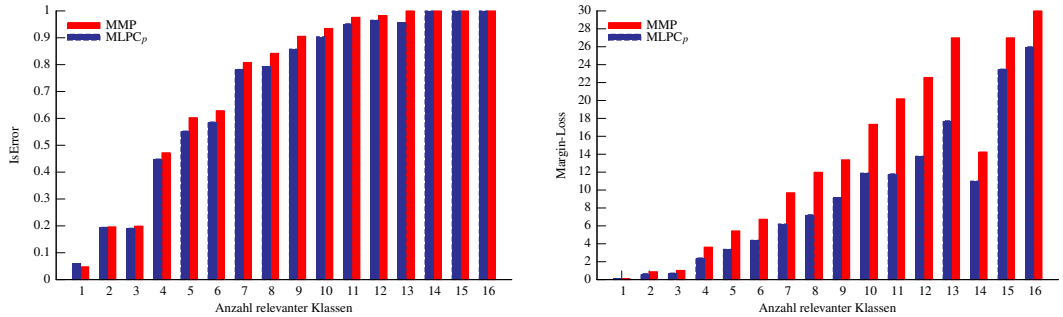


Abbildung 6.6: Fehler in Abhängigkeit der Anzahl der Labels eines Beispiels. Der Margin-Fehler des MMP-Algorithmus bei 16 Labels je Beispiele wurde abgeschnitten. Er beträgt 56 (bei jedoch nur einem Beispiel)

	*max*MMP	Δ [%]	*max*MLPC$_p$
IsError	29,24$^{(1)}$	-4,40	27,96$^{(5)}$
ErrSetSize	2,65$^{(2)}$	-28,15	1,92$^{(6)}$
Margin	1,90$^{(3)}$	-23,77	1,45$^{(5)}$
OneError	3,67$^{(4)}$	-19,58	2,95$^{(5)}$
AvgP	92,80$^{(1)}$	0,98	93,71$^{(5)}$
MaxF1	93,03$^{(1)}$	0,67	93,65$^{(5)}$

Tabelle 6.8: Vergleich der besten Einstellungen für MMP und MLPC$_p$auf den Testdaten. Für den MMP-Algorithmus wurde die Uniform-Update-Methode verwendet und folgende Loss-Funktionen: $^{(1)}$ IsError, $^{(2)}$ ErrSetSize, $^{(3)}$ Margin, $^{(4)}$ AvgP. Für MLPC$_p$das Perzeptron $^{(5)}$ ohne Schwellwert, $^{(6)}$ mit Schwellwert.

	MMP	Δ [%]	MLPC$_p$
IsError	26,48	-7,94	24,38
ErrSetSize	1,97	-45,49	1,07
Margin	1,50	-42,03	0,87
OneError	3,06	-28,24	2,20
AvgP	94,18	1,33	95,43
MaxF1	94,19	1,01	95,14

Tabelle 6.9: Direkter Vergleich zwischen MMP und MLPC$_p$auf den Trainingdaten. Die Spalte in der Mitte gibt die prozentuale Verbesserung für den MLPC$_p$-Algorithmus wieder.

aus ihrer Berechnungsvorschrift ergibt. Verwendet man beispielsweise für die Differenz die Werte $100 - AvgP$ und $100 - MaxF1$, so daß der beste zu erreichende Wert wie bei den anderen Maßen Null beträgt, so ergibt sich ein Vorteil für den MLPC$_p$-Algorithmus von -12,64% respektive -9,00% (vgl. Abschnitt 2.4.2). Beim Information Retrieval ist jedoch die Darstellung mit dem Wert 100 als Maximum üblich (für vollständiges Retrieval).

Da der MMP-Algorithmus je nach Anforderungen den Einsatz verschiedener Loss-Funktionen zuläßt, die die Leistung dementsprechend optimieren, ist in Tabelle 6.8 ein Vergleich zwischen den jeweils besten Ergebnissen des MMP-Algorithmus und des MLPC$_p$-Algorithmus dargestellt. Insbesondere für den Margin-Loss kann hierdurch eine relevante Verbesserung festgestellt werden, die jedoch nicht an die Leistung des MLPC$_p$ heranreicht. Zur Vollständigkeit ist in Tabelle 6.8 der Fehler auf der Trainingsmenge angegeben. Eine ausführlichere Betrachtung des Trainingsfehler in Zusammenhang mit der Anzahl der Epochen findet in Abschnitt 6.4.4 statt.

6.4.2 Laufzeit und Speicherverbrauch

Für den im vorangegangenen Abschnitt durchgeführten Versuch wurden die Laufzeit und der Speicherverbrauch ermittelt. Die Berechnungen fanden auf einem AMD Opteron Computer mit zwei 250/850 Prozessoren á 2,4 GHz und 2 GB Arbeitsspeicher statt. Zeiten wurden in CPU-Sekunden gemessen.

Der MMP benötigte für das Trainieren der 521.439 Instanzen 597 Sekunden, für das Testen der 282.975 Instanzen 376 Sekunden. Der MLPC$_p$-Algorithmus verwendete 1.174 respektive 9.012 Sekunden. Dies ergibt ein Verhältnis von 1,9 für das Training und 23,9 für das Testen.

Da jedoch neben der eigentlichen Berechnung des Algorithmus Operationen wie Protokollierung, Datenlesevorgänge u.ä. durchgeführt werden und nicht ausgeschlossen werden kann, daß dies innerhalb der CPU-Zeit-Messung geschieht, wurde zusätzlich die Anzahl der Skalarprodukt-Operationen erhoben (siehe 3.9).

Der MMP-Algorithmus benötigte demnach für das Training insgesamt 55.168.406 Π-Operationen, dies entspricht 105,80 Operationen je Beispiel. Das Ergebnis deckt sich mit der Komplexitätsanalyse in Abschnitt 4.4. Für den MLPC$_p$-Algorithmus wurden 169.424.707 Π-Operationen für das Training benötigt, d. h. 324,92 je Beispiel. Das Verhältnis beträgt somit 3,07, was in etwa der durchschnittlichen Anzahl der relevanten Klassen je Beispiel entspricht (die durchschnittliche Anzahl Labels auf der Trainingsmenge wurde nicht ermittelt). Für das Testen benötigte der MMP-Algorithmus 29.146.425 und der MLPC-Algorithmus 1.486.467.675 Operationen, was einem Verhältnis von 51 entspricht.

Die ermittelten Laufzeiten bestätigen die Berechnungen zur Komplexität aus Tabelle 5.1: für das Lernen entspricht das Verhältnis MMP- zu MLPC-Algorithmus in etwa der Anzahl der relevanten Klassen je Beispiel L. Für das Klassifizieren beträgt das Verhältnis das berechnete Vielfache $\frac{K-1}{2}$.

Die Messungen des Speicherverbrauchs ergaben 393 MB Speicherverbrauch für den MMP und 882 MB für den MLPC$_p$. Der Overhead ergibt sich aus der verwendeten Laufzeitumgebung.

6.4.3 Lernkurve

Die Lernkurve eines Algorithmus zeigt auf, wie schnell sich ein Algorithmus an die vorliegenden Daten anpaßt, d. h. wie schnell er in der Lage ist, seine Hypothese an neue Daten anzupassen. Ein Algorithmus mit einer steilen Lernkurve bedeutet zudem, daß das Verfahren bereits mit wenigen Beispielen eine allgemeingültige Hypothese aufstellen kann.

In Abbildung 6.7 werden die Lernkurven des MMP- und des MLPC$_p$-Algorithmus gegenübergestellt. Hierfür wurde der aktuelle Klassifizierer jeweils auf das nächste Beispiel angewandt, bevor es für das Training verwendet wurde (diese Form der Lernkurve ist effizient nur für inkrementelle Lerner anzugeben). Die y-Achse gibt somit den durchschnittlichen akkumulierten Fehler wieder, d. h. den akkumulierten Fehler geteilt durch die Anzahl der Trainingsbeispiele.

Es zeigt sich, daß der MMP-Algorithmus für den akkumulierten IsError-Fehler eine anfangs steilere Lernkurve besitzt als der MLPC$_p$-Algorithmus. Doch bereits bei etwa 7.500 Trainingsbeispielen wird der Vorsprung ausgeglichen. Bei der Betrachtung des Margin-Fehler ist im Gegensatz zum IsError-Loss kein anfänglicher Vorteil des MMP-Algorithmus zu beobachten, der Margin-Fehler

des MLPC$_p$-Algorithmus ist zu jedem Zeitpunkt niedriger als der des MMP-Algorithmus. Dies kann einerseits daran liegen, daß der MMP-Algorithmus in der für diesen Versuch verwendeten Grundeinstellung anhand des IsError-Loss trainiert wurde, und andererseits beim Margin-Loss ein allgemein größerer Vorsprung des Multilabel-Pairwise-Coupling-Algorithmus zu verzeichnen ist (siehe Tabelle 6.6). Die Kurven der ErrorSetSize-, OneError-, OneAvgP- und MaxF1-Losses zeigen ein ähnliches Bild wie beim Margin-Loss.

Eine weitere interessante Beobachtung ist der parallele Verlauf der Kurven. Anscheinend reagieren beide Verfahren ähnlich empfindlich auf die Folge der Beispiele. Daraus ist auf die Abhängigkeit beider Algorithmen von der Leistung des Perzeptrons auf den Daten zu schließen. Dies zeigt wiederum eine allgemeine Überlegenheit des paarweisen Ansatzes gegenüber der One-against-all-Methode auf.

Die Entstehung der Lernkurve entspricht einem realen Szenario: ein maschineller Klassifizierer schlägt für ein gegebenes Textdokument, beispielsweise eine Nachricht bei einer Nachrichtenagentur, eine Menge von relevanten Kategorien vor. Diese Zuordnung wird von einem Menschen überprüft, beispielsweise einem Redakteur, und notfalls korrigiert. Bei einer Korrektur erhält der Lernalgorithmus Rückmeldung und kann den Klassifizierer anpassen. Für die Darstellung der Lernkurve wurde der IsError- und Margin-Loss gewählt, da diese eine intuitive Beurteilung zulassen. Der MLPC$_p$-Algorithmus hätte nach einem halben Jahr Training (entspricht in etwa 400.000 Dokumenten) 70% der Dokument perfekt klassifiziert und im Durchschnitt zwei relevante Klassen unter der am besten platzierten irrelevanten Klasse platziert oder zwei irrelevante über der am schlechtesten platzierten relevanten Klasse gestellt (beides entspricht einem Margin-Fehler von zwei). Hierbei ist allerdings zu bemerken, daß für falsch klassifizierte Beispiele der Ranking-Fehler im Durchschnitt 6,7 beträgt (vgl. Margin*-Loss in Abschnitt 6.4.1).

6.4.4 Konvergenz und Overfitting

In diesem Abschnitt wird untersucht, wie sich die Algorithmen bei wiederholtem Lernen der Trainingsdaten verhalten. Im Zusammenhang mit dem MMP-Algorithmus konnten Crammer und Singer hierbei bereits ein klares Overfitting feststellen. Dies konnte durch den hier durchgeführten Versuch bestätigt werden.

Für die Untersuchung wurde das Training bis zu dreißig Mal wiederholt. Die Entwicklung des Fehlers auf der Trainings- und Testmenge ist in Abbildung 6.8 dargestellt. Es fällt auf, daß beide Algorithmen den Fehler auf den Trainingsdaten vermindern können. Für den Multilabel-Pairwise-Coupling-Algorithmus ist sogar eine Konvergenz erkennbar. Dies bestätigt die Vorteile des paarweisen Ansatzes wegen der Vergrößerung des Freiraums zwischen den Klassen (siehe Kapitel 5).

Allerdings läßt sich die bessere Anpassung der Hypothesen an die Trainingsdaten nicht auf die Leistung auf den Testdaten übertragen. Beide Algorithmen können keine Verbesserung des Fehlers auf den Testdaten bei Wiederholen des Trainings erreichen. Beim MMP-Algorithmus ergibt sich sogar eine bedeutsame Verschlechterung, die klar auf ein Overfitting hindeutet. Dies ist möglicherweise auf die unausgeglichene Klassenverteilung zurückzuführen, für die das Perzeptron empfänglich ist. Ein Zeichen hierfür ist das Verhalten der *unwissenden* Basisklassifizierer beim MLPC$_p$-Algorithmus (siehe Abschnitt 5.6): 70% dieser Basisklassifizierer entscheiden sich auf den Testdaten für die größere Klasse (bei einer Epoche und den Grundeinstellungen). Für den MLPC$_p$-Algorithmus sind

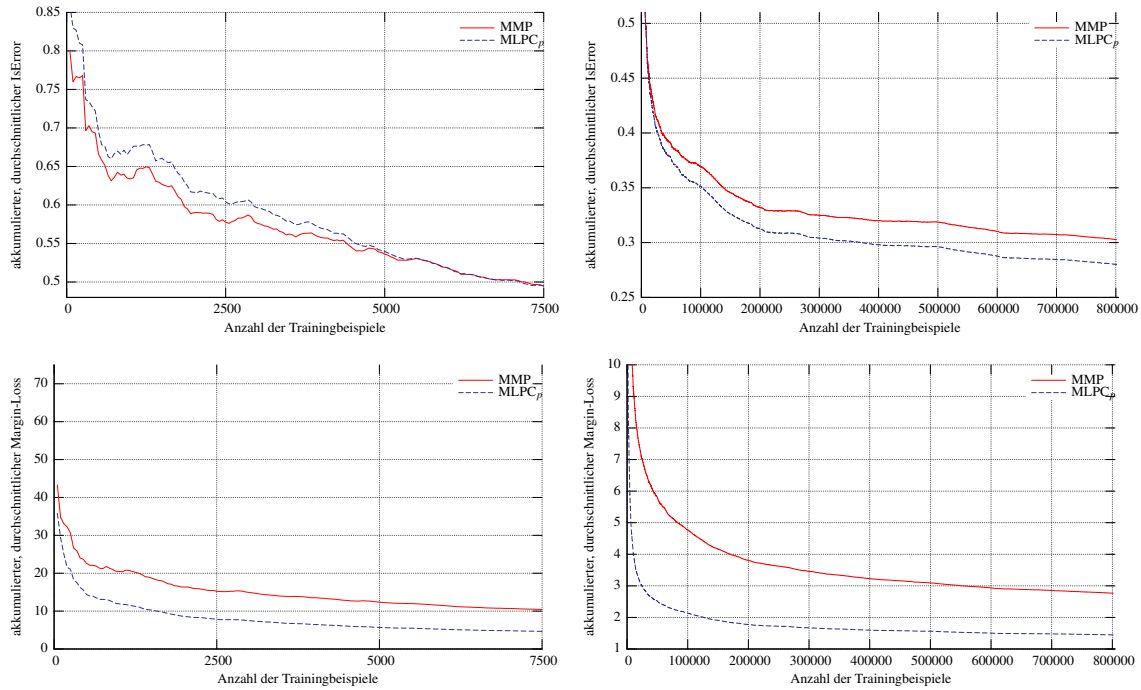


Abbildung 6.7: Lernkurven für den MMP-Algorithmus und für den Multilabel-Pairwise-Coupling-Algorithmus

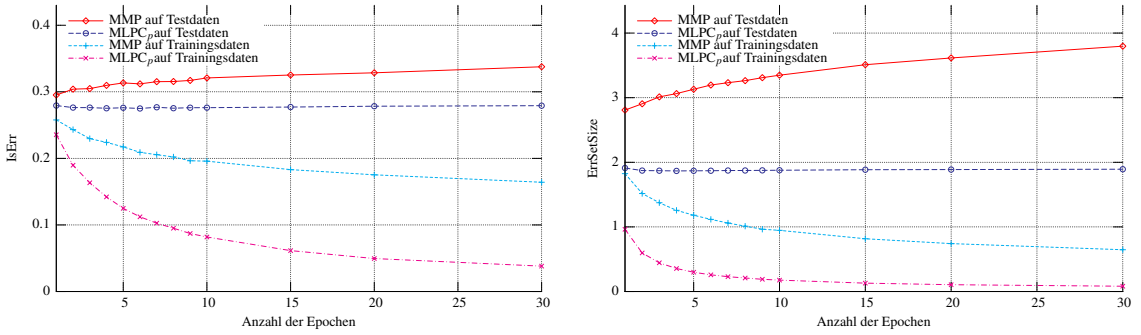


Abbildung 6.8: Trainings- und Testfehler abhängig von der Anzahl der Epochen. Es ist der Fehler auf den Trainingsdaten und auf den Testdaten angegeben.

Variante	IsErr	Δ [%]	ErrSet	Δ [%]	OneErr	Δ [%]	AvgP	Δ [%]	MaxF1	Δ [%]
Voting_0	27,94	1,37	1,91	0,96	2,95	0,60	93,71	0,08	93,66	0,10
PKPD_0	27,55	–	1,89	–	2,94	–	93,78	–	93,75	–

Tabelle 6.10: Vergleich der PKPD-Methode mit dem Voting. Es wurde jeweils das Perzeptron ohne Schwellwert verwendet.

für den Fehler auf den Testdaten abhängig von der Anzahl der Iterationen trotzdem keine signifikanten Unterschiede meßbar. Dies ist möglicherweise auf das Voting-Verfahren zurückzuführen, das ausgleichend wirkt. Dieser Umstand zeigt sich beispielsweise auch in der stetigeren Entwicklung des MLPC_pbei der Feature-Selection in Abbildung 6.5.

Es kann zusammengefaßt werden, daß der MLPC_p-Algorithmus trotz starker Anzeichen für ein Overfitting des Basislerners selbst hiervon nicht betroffen zu sein scheint. Dies läßt zudem vermuten, daß die Verwendung des in Abschnitt 3.8.2 vorgestellten Ballseptrons eine weitere Verbesserung möglich machen könnte, da dieser einen Minimalabstand zu den binären Trainingsbeispielen vorsieht und deshalb eine allgemeingültigere Hyperebene finden könnte [Shalev-Shwartz und Singer, 2005]. Weitere Untersuchungen in dieser Richtung wären von Interesse.

6.5 Zusätzliche Untersuchungen

Es wurden einige zusätzliche Untersuchungen unternommen, die nun kurz vorgestellt werden.

6.5.1 PKPD

Der in Abschnitt 5.8 erläuterte Algorithmus PKPD nach Price, Knerr, Personnaz und Dreyfus [1995] wurde implementiert und mit dem Multilabel-Pairwise-Coupling-Algorithmus verglichen. Die Resultate befinden sich in Tabelle 6.10. Es ist insbesondere bei dem IsError-Maß eine Verbesserung feststellbar. Übertragen auf den Vergleich mit dem MMP-Algorithmus erreicht die PKPD-Variante eine Verbesserung von 5,78% bei dem IsError-Fehler. Das Ergebnis läßt auch für weitere in Abschnitt 5.8 erwähnte Methoden zur Kombinierung der Einzelvorhersagen eine Leistungssteigerung versprechen.

6.5.2 Ausnutzung der Hierarchical Policy

Die in Abschnitt 6.1.1 erläuterte Hierarchical Policy von Reuters läßt sich ausnutzen, um die Menge der relevanten Klassen eines Beispiels zu reduzieren. Die Menge der relevanten Klassen stellt bedingt durch die Klassifikationsrichtlinie einen Teilbaum der Klassenhierarchie dar. Aus der initialen Menge der Labels werden nun nur diese behalten, die in dem Teilbaum ein Blatt darstellten, da sich die restlichen Klassen anhand der Richtlinie rekonstruieren lassen [vgl. Lewis u. a., 2004, Abschnitt 3.3.2]. Der so erhaltene Datensatz enthält nunmehr 1,46 relevante Klassen je Beispiel. Die Ergebnisse des MMP- und MLPC_p-Algorithmus sind in Tabelle 6.11 dargestellt. Es ist eine

	MMP	Δ [%]	MLPC$_p$
IsErr	27,30	-6,49	25,53
ErrSetSize	1,78	-34,07	1,18
Margin	1,60	-34,47	1,05
OneErr	15,15	-8,57	13,85
AvgP	87,77	1,47	89,06
MaxF1	90,06	1,14	91,09

Tabelle 6.11: Vergleich des MMP- und MLPC$_p$-Algorithmus bei Ausnutzung der Hierarchie.

allgemeine Verbesserung des Fehlers auf den Testdaten zu verzeichnen, die zu erwarten war. Allerdings fällt auf, daß der OneError-, AvgP- und MaxF1-Loss schlechtere Werte aufweisen. Dies ist auf die Reduzierung der durchschnittlichen Anzahl relevanter Klassen zurückzuführen.

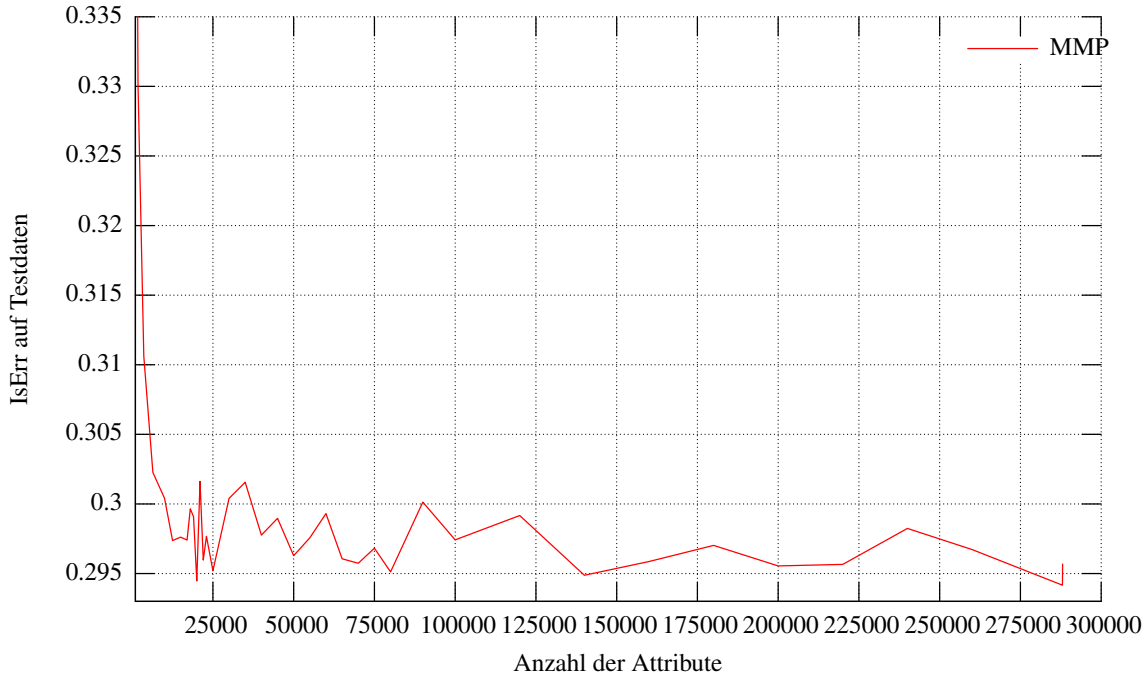


Abbildung 6.9: Komplette Übersicht der Feature-Selection für den MMP-Algorithmus

7 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein neuartiger Algorithmus vorgestellt, der das Pairwise-Coupling-Verfahren für den Multilabel-Fall verallgemeinert. Die theoretischen Überlegungen zum MMP und zum MLPC$_p$ konnten in der experimentellen Evaluation bestätigt werden. Es zeigte sich eine klare Verbesserung in der Klassifizierung im Vergleich zum konkurrierenden MMP-Algorithmus. Insbesondere bei der Qualität der erzeugten Klassenrankings konnte der MLPC$_p$ überragen. Dabei konnten die positiven Eigenschaften des effizienten und inkrementellen Lernens des MMP-Algorithmus übernommen werden. Angesichts der hohen Anforderung der Reuters-Datenbank konnte sich der MLPC$_p$-Algorithmus zudem für den Einsatz bei der Textklassifizierung bewähren.

Eine weitere interessante Eigenschaft ist die offensichtliche Resistenz gegen Overfitting. Diese Resistenz könnte durch Austausch des Basislerners sogar zu einer weiteren Leistungssteigerung genutzt werden. Beispielsweise stellen Ballseptrons eine leichte und effiziente Möglichkeit dar, die Fähigkeit zur Generalisierung des Perzeptron-Algorithmus weiter zu vergrößern und das Problem der unausgeglichenen Klassenverteilungen zu lösen (Abschnitte 3.8.2, 6.4.4). Die Berücksichtigung der Konfidenzen der Einzelvorhersagen kann eine zusätzliche Verbesserung erwirken, wie anhand des PKPD-Algorithmus demonstriert wurde (Abschnitt 6.5.1). Durch den Einsatz von Kernel-Methoden läßt sich zudem der MLPC$_p$ auch für Problemklassen erweitern, die nicht linear separierbar sind [Freund und Schapire, 1999].

Auch andere Basisklassifizierer sind denkbar. Allerdings muß hierbei der durch die besonderen Gegebenheiten des paarweisen Ansatzes bedingte hohe Speicherverbrauch der Hypothese und die allgemein längere Laufzeit für die Klassifizierung beachtet werden. Beide Größen entwickeln sich quadratisch in der Anzahl der möglichen Klassen. In Abschnitt 5.8 wurden Lösungsansätze vorgestellt, wie andere Techniken zur Auswertung der Einzelvorhersagen oder zur Ausnutzung der Hierarchie verwendet werden könnten, um die Laufzeit und sogar die Anzahl der nötigen Klassifizierer zu reduzieren.

Zusätzlich zu den aufgezeigten Möglichkeiten der Modifikationen am Algorithmus und ihren Auswirkungen bleiben zwei offene Fragen für zukünftige Arbeiten. Die erste Frage betrifft die Abhängigkeit des Rankingfehlers von dem Fehler des unterliegenden Basisklassifizierers: läßt sich eine allgemeingültige Beziehung zwischen den zwei Größen angeben? Erste Versuche wurden in dieser Arbeit unternommen (siehe Abschnitt 5.6). Die zweite Frage betrifft den Vergleich mit anderen Lernalgorithmen als den MMP. Da die verwendeten Fehlermaße trotz der an dieser Stelle unterstellten großen Aussagekraft bisher weitgehend unbeachtet blieben, erschwert dies einen direkten Vergleich zu etablierten Algorithmen. Bisher konnte eine vergleichsweise gute Leistung nur vermutet werden.

Literaturverzeichnis

[Baldi u. a., 2003]
Pierre BALDI, Paolo FRASCONI, Padhraic SMYTH: *Modeling the Internet and the Web: Probabilistic Methods and Algorithms*. John Wiley & Sons, 2003. ISBN 0470849061 25, 26, 38

[Bishop, 1995]
Christopher M. BISHOP: *Neural Networks for Pattern Recognition.* Oxford University Press, 1995. ISBN 0198538642 39

[Block, 1962]
H. D. BLOCK: *The perceptron: A model for brain functioning. I.* Reviews of Modern Physics, Band 34, S. 123–135, 1962. URL prola.aps.org/abstract/RMP/v34/i1/p123_1 39

[Burges, 1998]
Christopher J. C. BURGES: *A Tutorial on Support Vector Machines for Pattern Recognition*. Data Mining and Knowledge Discovery, Band 2, Nr. 2, S. 121–167, 1998. URL citeseer.ist.psu.edu/article/burges98tutorial.html 41, 44

[Cohen und Singer, 1996]
William W. COHEN, Yoram SINGER: *Context-sensitive learning methods for text categorization*. In: *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval*, S. 307–315. ACM Press, 1996. URL citeseer.ist.psu.edu/cohen96contextsensitive.html 47, 63

[Cortes und Vapnik, 1995]
Corinna CORTES, Vladimir VAPNIK: *Support-Vector Networks*. Machine Learning, Band 20, Nr. 3, S. 273–297, 1995. URL citeseer.ist.psu.edu/cortes95supportvector.html 38, 44

[Crammer und Singer, 2003]
K. CRAMMER, Y. SINGER: *A Family of Additive Online Algorithms for Category Ranking*. Journal of Machine Learning Research, Band 3, Nr. 6, S. 1025–1058, 2003. URL www.jmlr.org/papers/v3/crammer03b.html 9, 21, 43, 48, 53, 54, 55, 57, 59, 76, 79, 80, 81, 83, 84, 89

[Crammer und Singer, 2002]
Koby CRAMMER, Yoram SINGER: *A new family of online algorithms for category ranking*. In: *Proceedings of the 25th Annual International Conference on Research and Development in Information Retrieval (SIGIR 2002)*, S. 151–158. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-561-0. URL www.cs.huji.ac.il/~singer/papers/mcml.ps.gz 48

[Cutzu, 2003]
Florin CUTZU: *Polychotomous Classification with Pairwise Classifiers: A New Voting Principle*. In: *Proceedings of the 4th International Workshop on Multiple Classifier Systems (LNCS 2709)*, S. 115–124. Springer, 2003. ISBN 3-540-40369-8. URL citeseer.ist.psu.edu/595379.html 71

[Dagan u. a., 1997]
Ido DAGAN, Yael KAROV, Dan ROTH: *Mistake-driven learning in text categorization*. In: *Proceedings of 2nd Conference on Empirical Methods in Natural Language Processing (EMNLP 97)*, S. 55–63. Association for Computational Linguistics, Morristown, US, 1997. URL citeseer.ist.psu.edu/dagan97mistakedriven.html 47

[Dinnesen und Andersen, 2005]
Timo Paukku DINNESEN, Alex ANDERSEN: *Article classification*. Datamining Project Report, University of Aarhus, Denmark, 2005 81, 82

[Elisseeff und Weston, 2001]
André ELISSEEFF, Jason WESTON: *A Kernel Method for Multi-Labelled Classification*. In: *Advances in Neural Information Processing Systems*, Band 14, S. 681–687. MIT Press, 2001. URL www.kyb.tuebingen.mpg.de/bs/people/weston/publications.html 20

[Freund und Schapire, 1999]
Yoav FREUND, Robert E. SCHAPIRE: *Large Margin Classification Using the Perceptron Algorithm*. Machine Learning, Band 37, Nr. 3, S. 277–296, 1999. URL www.cse.ucsd.edu/~yfreund/papers/LargeMarginsUsingPerceptron.pdf 8, 39, 40, 42, 43, 44, 48, 94

[Frieß u. a., 1998]
Thilo-Thomas FRIESS, Nello CRISTIANINI, Colin CAMPBELL: *The Kernel-Adatron Algorithm: A Fast and Simple Learning Procedure for Support Vector Machines*. In: *Proceedings of the 15th International Conference on Machine Learning (ICML 1998)*, S. 188–196. Morgan Kaufmann, 1998. ISBN 1-55860-556-8. URL citeseer.ist.psu.edu/friess98kerneladatron.html 44

[Fürnkranz, 2001]
Johannes FÜRNKRANZ: *Round Robin Rule Learning*. In: *Proceedings of the 18th International Conference on Machine Learning (ICML 01)*, S. 146–153. Morgan Kaufmann Publishers, Williamstown, MA, 2001. URL citeseer.ifi.unizh.ch/frnkranz01round.html 62

[Fürnkranz, 2002]
Johannes FÜRNKRANZ: *Round Robin Classification.* Journal of Machine Learning Research, Band 2, S. 721–747, 2002. URL www.jmlr.org/papers/v2/fuernkranz02a.html 9, 48, 62, 63, 70, 71, 73

[Granitzer, 2003]
Michael GRANITZER: *Hierarchical Text Classification using Methods from Machine Learning*. Diplomarbeit, Technische Universität Graz, 2003. URL en.know-center.at/forschung/wissenserschliessung/dissertationen_diplomarbeiten 58

[Hsu und Lin, 2002]
Chih-Wei HSU, Chih-Jen LIN: *A Comparison of Methods for Multi-class Support Vector Machines*. IEEE Transactions on Neural Networks, Band 13, Nr. 2, S. 415–425, 2002. URL www.csie.ntu.edu.tw/~cjlin/papers/multisvm.pdf 8, 9, 63

[Joachims, 1998]
Thorsten JOACHIMS: *Text Categorization with Suport Vector Machines: Learning with Many Relevant Features.* In: *Machine Learning: ECML-98, 10th European Conference on Machine Learning (LNCS 1398)*, S. 137–142. Springer, 1998. ISBN 3-540-64417-2. URL hdl.handle.net/2003/2595 47

[Kuncheva, 2004]
Ludmila I. KUNCHEVA: *Combining Pattern Classifiers : Methods and Algorithms*. Wiley-Interscience, 2004. ISBN 0471210781 38

[Lewis u. a., 2004]
David D. LEWIS, Yiming YANG, Tony G. ROSE, Fan LI: *RCV1: A New Benchmark Collection for Text Categorization Research*. Journal of Machine Learning Research, Band 5, S. 361–397, 2004. URL jmlr.csail.mit.edu/papers/v5/lewis04a.html 74, 75, 76, 82, 91

[MacKay, 2005]
David MACKAY: *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 7.2 Auflage, 2005. ISBN 0521642981. URL www.inference.phy.cam.ac.uk/mackay/itprnn/book.html 30

[Minsky und Papert, 1969]
Marvin L. MINSKY, Seymour PAPERT: *Perceptrons*. MIT Press, Cambridge, MA, 1969 8, 38

[Mitchell, 1997]
Tom M. MITCHELL: *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997. ISBN 0070428077 16, 25, 28, 43

[Ng u. a., 1997]
Hwee T. NG, Wei B. GOH, Kok L. LOW: *Feature selection, perceptron learning, and a usability case study for text categorization*. In: *Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval*, S. 67–73. ACM Press, New York, USA, 1997. URL citeseer.ist.psu.edu/ng97feature.html 8, 47

[Novikov, 1962]
A.B.J. NOVIKOV: *On convergence proofs on Perceptrons*. In: *Proceedings of the Symposium of the Mathematical Theory of Automata*, Band XII, S. 615–622. 1962 39

[Platt u. a., 1999]
J. PLATT, N. CRISTIANINI, J. SHAWE-TAYLOR: *Large Margin DAGs for Multiclass Classification*. In: *Advances in Neural Information Processing Systems 12*, S. 547–553. The MIT Press, 1999. URL citeseer.ist.psu.edu/platt00large.html 63, 72

[Price u. a., 1995]
David PRICE, Stefan KNERR, Leon PERSONNAZ, Gerard DREYFUS: *Pairwise Neural Network Classifiers with Probabilistic Outputs*. In: *Advances in Neural Information Processing*

Systems, Band 7, S. 1109–1116. The MIT Press, 1995. URL citeseer.ist.psu.edu/price94pairwise.html 72, 91

[Reif, 2000]
Gerald REIF: *Moderne Aspekte des Wissensverarbeitung - Ein interaktiver Lernbehelf für das Web Based Training*. Diplomarbeit, Technische Universität Graz, 2000. URL www.iicm.edu/greif/thesis.html 30

[Rogati und Yang, 2002]
Monica ROGATI, Yiming YANG: *High-performing feature selection for text classification*. In: *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, S. 659–661. ACM Press, New York, NY, USA, 2002. ISBN 1-58113-492-4. URL www.cs.cmu.edu/~yiming/publications.html 82

[Rosenblatt, 1958]
F. ROSENBLATT: *The perceptron: a probabilistic model for information storage and organization in the brain*. Psychological Review, Band 65, Nr. 6, S. 386–408, 1958 8, 30

[Sebastiani, 2002]
Fabrizio SEBASTIANI: *Machine learning in automated text categorization*. ACM Computing Surveys, Band 34, Nr. 1, S. 1–47, 2002. URL citeseer.ist.psu.edu/article/sebastiani99machine.html 19, 23, 26, 43, 47, 59

[Shalev-Shwartz und Singer, 2005]
Shai SHALEV-SHWARTZ, Yoram SINGER: *A New Perspective on an Old Perceptron Algorithm.* In: *Learning Theory, 18th Annual Conference on Learning Theory (COLT 2005)*, S. 264–278. Springer, 2005. ISBN 3-540-26556-2. URL www.cs.huji.ac.il/~shais/papers/ShalevSi05.pdf 42, 44, 91

[Soica und Hearst, 2004]
Emilia SOICA, Marti A. HEARST: *Nearly-Automated Metadata Hierarchy Creation*. In: *Companion Proceedings of Human Language Technology conference – North American chapter of the Association for Computational Linguistics annual meeting (HLT-NAACL) 2004*, S. 117–120. Association for Computational Linguistics, Boston, Massachusetts, USA, 2004. URL www.sims.berkeley.edu/~hearst/publications.html 81

[Vapnik, 2000]
Vladimir N. VAPNIK: *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 2. Auflage, 2000. ISBN 0-387-98780-0 40, 41, 42, 44

[Weston und Watkins, 1999]
Jason WESTON, C. WATKINS: *Support vector machines for multiclass pattern recognition*. In: *Proceedings of the Seventh European Symposium On Artificial Neural Networks (ESANN)*, S. 219–224. 1999. URL www.dice.ucl.ac.be/Proceedings/esann/esannpdf/es1999-461.pdf 8, 63

[Witten und Frank, 1999]
Ian H. WITTEN, Eibe FRANK: *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999. ISBN 1558605525 17, 76

[Wu u. a., 2004]
Ting-Fan WU, Chih-Jen LIN, Ruby C. WENG: *Probability Estimates for Multi-class Classification by Pairwise Coupling*. Journal of Machine Learning Research, Band 5, S. 975–1005, 2004. URL www.jmlr.org/papers/volume5/wu04a/wu04a.pdf 72

[Yang und Liu, 1999]
Y. YANG, X. LIU: *A re-examination of text categorization methods*. In: *22nd Annual International SIGIR*, S. 42–49. Berkley, 1999. URL citeseer.ist.psu.edu/yang99reexamination.html 47

[Yang und Pedersen, 1997]
Yiming YANG, Jan O. PEDERSEN: *A Comparative Study on Feature Selection in Text Categorization*. In: *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, S. 412–420. Morgan Kaufmann Publishers Inc., 1997. ISBN 1-55860-486-3. URL www.cs.cmu.edu/~yiming/publications.html 28, 29, 82

[Yao u. a., 2001]
Yuan YAO, Gian Luca MARCIALIS, Massimiliano PONTIL, Paolo FRASCONI, Fabio ROLI: *A New Machine Learning Approach to Fingerprint Classification.* In: *AI*IA 2001: Advances in Artificial Intelligence, 7th Congress of the Italian Association for Artificial Intelligence*, S. 57–63. Springer, 2001. ISBN 3-540-42601-9. URL citeseer.ifi.unizh.ch/yao01new.html 63