Diplomarbeit

Maschinelle Lernverfahren
zum adaptiven Informationsfiltern
bei sich verändernden Konzepten

Ralf Klinkenberg

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

16. Februar 1998

Betreuer:

Prof. Dr. Katharina Morik

Dipl.-Inform. Thorsten Joachims

Überarbeitete Fassung vom 16. März 1998.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung und Überblick

Durch die ständige Zunahme der weltweiten Kommunikation und Informationsverbreitung steigt der Umfang an Information, den eine einzelne Person, Institution oder Firma zu bewältigen hat. Dies führt zu einem immer stärker werdenden Bedarf an Verfahren, die dem Empfänger dieser Information dabei helfen, die von ihm gesuchte oder für ihn interessante Information aus der immer größer werdenden Datenmenge herauszufiltern bzw. diese Informationen zu klassifizieren und weiterzuleiten. Insbesondere die in den letzten Jahren stark gestiegene Popularität des Internet mit seinen verschiedenen Diensten hat stark zum Ansteigen des weltweiten Informationsaustauschs beigetragen. Täglich begeistern sich neue Anwender für die optisch ansprechende und zudem leicht bedienbare Oberfläche des World Wide Web (WWW), die Möglichkeit der schnellen und billigen weltweiten Kommunikation per elektronischer Post (E-Mail) und den leichten Zugriff auf eine große Vielfalt an Informations- und Diskussionsforen (Newsgroups).

Automatische Informationsfilter- und Textklassifikationssysteme bieten dem Empfänger großer Informationsmengen eine Hilfestellung bei deren Bewältigung. Die Aufgabe des Informationsfilterns (*Information Filtering*) ist es, Dokumente aus einem Strom von Texten in relevant bzw. nicht relevant zu klassifizieren oder sie einer von mehreren Kategorien zuzuordnen. Ein solcher Strom von Dokumenten können beispielsweise elektronische Nachrichten sein, aus denen die für einen Benutzer interessanten Texte automatisch herausgefunden und die nicht relevanten verworfen werden sollen. Beispiele für solche Nachrichtenfilter sind der *NewsWeeder* für Newsgroups [Lang, 1995] und die *Persönliche Zeitung* für die im WWW verfügbaren Artikel einiger deutscher Tageszeitungen [Veltmann, 1997]. Ein solcher Dokumentenstrom können auch die per E-Mail an ein Unternehmen geschickten Nachrichten sein, die je nach Zuständigkeit automatisch an einzelne Abteilungen oder Sachbearbeiter weitergeleitet werden sollen.

Welche Texte aus dem zu filternden Strom für ein Benutzerinteresse oder eine Kategorie relevant sind, wird durch ein Profil beschrieben, das z. B. wichtige Begriffe (Schlüsselwörter) enthalten kann. Profile können manuell erstellt werden, was aber nicht sehr effizient ist, weil dazu sowohl entsprechende Sachkenntnis als auch ein nicht unerheblicher Arbeitsaufwand notwendig sind. Dieser Aufwand ist umso größer, je öfter das Profil geändert werden muß. Profile können aber auch mit Verfahren des maschinellen Lernens anhand von relevanten Texten (Lernstichprobe) automatisch erstellt werden. Diese Verfahren lernen anhand von Trainingsbeispielen, also aus Texten mit bekannter Klassifikation, das Konzept, nach dem die Dokumente klassifiziert werden. Das gelernte Klassifikationsmodell (Profil) kann dann benutzt werden, die Klasse neuer Dokumente vorherzusagen. Bisher kaum behandelt sind Fragestellungen, inwieweit sich ein einmal erworbenes Klassifikationsmodell anhand neuer Texte adaptieren läßt. Eine solche Adaption ist besonders dann wichtig, wenn sich das der Klassifikation zugrundeliegende Konzept verändert. Man kann dabei verschiedene *Arten von Konzeptverschiebungen* unterscheiden:

1. Der Begriff verschiebt sich (*Begriffsverschiebung*), d. h. die Semantik des Wortes verändert sich im Laufe der Zeit.

2. Das Interesse verschiebt sich auf einen anderen, eventuell neuen Begriff (*Interessenverschiebung*).

3. Die zu einem Begriff verfügbaren Dokumente verschieben sich (*Verschiebung verfügbarer Dokumente*).

Die Konzeptverschiebung in Form einer *Begriffsverschiebung* läßt sich sehr gut anhand des Wortes „Reform“ erläutern. Während das Wort „Reform“ früher meist im Sinne positiver Veränderungen gebraucht wurde, für die oft auch öffentliche Gelder bereitgestellt wurden, erscheint es heute meistens in einem eher negativen Kontext. Hinter sogenannten Reformen verbergen sich heutzutage oft eher unangenehmne Maßnahmen zur Kosteneinsparung, die oft den Verlust von Arbeitsplätzen oder die Kürzung finanzieller Mittel zur Folge haben.

Zur Illusttration des Begriffs der *Interessenverschiebung* kann man einen fiktiven Zeitungsleser betrachten, der während der Fußballsaison mit großem Interesse die Berichte über die jeweils aktuellen Bundesliga-Berichte verfolgt. Nach Ende der Saison interessiert ihn das Thema Fußball nicht weiter. Stattdessen wendet er sich Berichten über Tanzsportveranstaltungen zu, die nach Ende der Fußballsaison stattfinden. In diesem Beispiel interessiert sich der Leser also für zwei Themen, die in verschiedenen Zeiträumen auftreten. Sein Interesse kann aber auch zwischen gleichzeitig auftretenden Themen wechseln. Möglicherweise interessiert sich dieser Leser aufgrund eines spektakulären Vorfalls für Nachrichten über illegale Firmenabsprachen und verfolgt Berichte über Untersuchungen des Bundeskartellamtes. Im Laufe der Zeit verschiebt sich sein Interesse dann aber mehr in Richtung legaler Firmenzusammenschlüsse und Übernahmen, obwohl weiterhin Berichte über kartellamtliche Untersuchungen verfügbar sind.

Eine *Konzeptverschiebung im Sinne der verfügbaren Dokumente* kann man am Begriff „Informatik“ beobachten. So versteht man beispielsweise heute ebenso wie vor zwanzig Jahren unter dem Begriff „Informatik“ die Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von Computern. Obwohl also der Inhalt der Informatik eigentlich unverändert geblieben ist, tauchten in Dokumenten aus dem Bereich der Informatik früher oft Begriffe wie „Elektronische Datenverarbeitung (EDV)“, „Datei“ oder „Rechenzentrum“ als Begleitwörter des Begriffs Informatik auf, während heute neue Wörter wie „Electronic Commerce“, „Virtuelle Realität“, „Agenten“ oder „World Wide Web (WWW)“ den Inhalt von Dokumenten aus dem Bereich der Informatik prägen. Aufgrund des technischen und/oder gesellschaftlichen Fortschritts können sich also die unter einem Oberbegriff gesammelten Dokumente ändern, ohne daß sich die Bedeutung des Oberbegriffs unbedingt ändern muß.

Man kann Konzeptverschiebungen auch gemäß ihrer Ausprägung und/oder Geschwindigkeit unterscheiden, was in einem gewissen Sinne orthogonal zur zuvor vorgenommenen Unterteilung gemäß des Verschiebungstyps ist. Die Menge der unter einer Kategorie als relevant betrachteten Dokumente kann sich im Laufe der Zeit entweder langsam und kontinuierlich (*Concept Drift*) oder in abrupten Sprüngen (*Concept Shift*) verschieben[1]. Während abrupte Sprünge normalerweise nur bei Interessenverschiebungen auftreten, können langsame und kontinuierliche Verschiebungen sowohl bei Konzeptverschiebungen im Sinne der zu einem Begriff verfügbaren Dokumente als auch bei Interessenverschiebungen auftreten.

In dieser Arbeit wird insbesondere der Aspekt im Sinne einer Interessenverschiebung veränderlicher Konzepte bei der Textklassifikation untersucht. Das Benutzerinteresse kann sich verändern, d. h. Texte (und Themen), die früher interessant waren, sind später möglicherweise nicht länger interessant. Ein adaptives Klassifikationsverfahren sollte in der Lage sein, sich Konzeptverschiebungen (im Sinne einer Interessenverschiebung des Benutzers) anzupassen. Ein weiteres Problem ergibt sich daraus, daß sich die Attribute, auf denen die Klassifikation basiert, ebenfalls ändern können, d. h. neue Wörter können in den Texten auftauchen und für die Klassifikation relevant werden, während andere an Bedeutung verlieren bzw. nicht mehr auftreten. Dies trifft sowohl für Konzeptverschiebungen im Sinne der verfügbaren Dokumente als auch im Sinne einer Interessenverschiebung zu. Informationsfilteraufgaben in sich dynamisch verändernden Domänen konfrontieren die Klassifikationsverfahren also mit zwei besonderen Problemen:

1. *Die Attribute ändern sich.* Manche Attribute werden im Laufe der Zeit nicht länger verwendet, während neue Attribute hinzukommen.

2. *Die Interpretation der Beispiele ändert sich.* Das zu lernende Zielkonzept, also die Interpretation mindestens einer Klasse, verändert sich.

[1] Diese Begriffsdefinitionen orientieren sich an [Kunisch, 1996] und [Allan, 1996].

Viele klassische Lernverfahren haben hier das Problem, daß sie von einem festen Satz von Attributen (Wortschatz) mit unveränderlicher Interpretation und einer statischen Konzeptinterpretation ausgehen. Weder das eine noch das andere ist hier gegeben.

Die Berücksichtigung des ersten Punktes verlangt vom Klassifikationsverfahren eine Aktualisierung der Menge der betrachteten Attribute, wodurch eventuell auch eine Anpasung oder ein Neu-Lernen des Klassifikationsmodells notwendig wird. Dies ist insbesondere dann der Fall, wenn das verwendete Verfahren Attribut-Wert-Vektoren konstanter Länge benötigt, und sich auch die Interpretation der einzelnen Vektorkomponenten nicht verändern darf.

Für die Berücksichtigung des zweiten Punktes sollten alte Datensätze in ihrer Bedeutung abgeschwächt beziehungsweise aus der Trainingsmenge entfernt werden, wenn sie nicht länger repräsentativ für die aktuelle Situation sind. Ein Ansatz hierfür ist die Gewichtung der Beispiele gemäß ihres Alters und/oder ihrer geschätzen Repräsentativität für ihre Klasse. Ein anderer, in dieser Arbeit eingehender untersuchter Ansatz ist die Verwaltung eines Zeitfensters auf den Daten, das es dem Lernverfahren erlaubt, nicht länger relevante Beispiele zu „vergessen“.

Ein adaptives Textklassifikationsverfahren sollte in der Lage sein, mit einem initialen Profil zu starten oder ein solches Profil aus einer initialen Menge von Trainingsbeispielen zu lernen. Anschließend sollte es ein sich veränderndes Profil aus einem alten Profil und neuen Beispielen lernen können. Um dieser Anforderung gerecht zu werden, muß ein adaptives Textklassifikationsverfahren Konzeptveränderungen erkennen und entsprechend reagieren können. Es sollte dazu in der Lage sein, die Menge der betrachteten Attribute zu aktualisieren, wenn dies sinnvoll oder notwendig ist, und es sollte sein Klassifikationsmodell (z. B. Klassifikationsregeln) gegebenenfalls anpassen oder neu lernen können.

Zwei zentrale Voraussetzungen für die Realisierung eines solchen adaptiven Textklassifikationsverfahrens sind also zum einen das *Erkennen von Konzeptverschiebungen* und zum anderen eine dies berücksichtigende Trainingsdatenverwaltung, die durch die *Verwaltung eines adaptiven Zeitfensters* auf den Beispieldokumenten realisiert werden kann. Diese Arbeit untersucht, wie man bei einer Textklassifikationsaufgabe mit sich verändernden Konzepten Konzeptverschiebungen erkennen kann, stellt ein Verfahren zur Verwaltung eines adaptiven Zeitfensters vor und evaluiert dieses Verfahren anhand von Experimenten mit realen Textdaten. Die beiden folgenden Abschnitte konkretisieren die Aufgabenstellung und geben einen Überblick über den Aufbau und den Inhalt dieser Arbeit.

1.1 Aufgabenstellung

Diese Arbeit befaßt sich mit dem Informationsfiltern bei sich verändernden Konzepten, also der Klassifikation von Texten aus einem Strom von Dokumenten, beim dem sich das der Klassifikation zugrunde liegende Konzept verschieben kann. Zentrale Voraussetzungen für die Realisierung eines adaptiven Textklassifikators sind das Erkennen von Konzeptverschiebungen, um entscheiden zu können, wann das Klassifikationsmodell aktualisiert oder neu gelernt werden muß, und eine geeignete Datenverwaltung, die es erlaubt, nicht länger relevante Beispieldokumente aus der Trainingsmenge zu entfernen und dennoch eine Trainingsmenge günstiger Größe zu behalten.

Ziel dieser Arbeit ist es, Methoden zur Realisierung dieser Voraussetzungen zu untersuchen. Es sollen geeignete Indikatoren und Kriterien zur Feststellung von Konzeptverschiebungen gefunden und in einer adaptiven Zeitfensterverwaltung für die Trainingsdaten eingesetzt werden. Anhand von Experimenten mit realen Textdaten sollen folgende drei Fragen untersucht werden:

1. Werden Konzeptverschiebungen mit den gewählten Kriterien erkannt ?
2. Führt die Verwendung von Zeitfenstern auf den Daten bei Textklassifikationsproblemen mit sich verändernden Konzepten zu einer Performanzsteigerung gegenüber dem Lernen auf allen bekannten Beispielen ?
3. Läßt sich durch den Einsatz der vorgestellten adaptiven Fensterverwaltung eine weitergehende Verbesserung erreichen als mit einfacheren Ansätzen mit Fenstern fester Größe ?

1.2 Überblick

Diese Arbeit beginnt mit einer Einführungen in die wichtigsten technischen Grundlagen und Verarbeitungsschritte des Informationsfilterns und der Textkategorisierung (Kapitel 2). Sie beschreibt eine Reihe von Lernverfahren aus dem Bereich des Information Retrieval und des maschinellen Lernens, die im Bereich der Textklasssifikation eingesetzt werden (Kapitel 3). Während diese Verfahren nicht speziell für das Lernen veränderlicher Konzepte ausgelegt sind, wurden die in Kapitel 4 vorgestellten maschinellen Lernverfahren speziell zum Lernen bei Konzeptverschiebungen entworfen. Anhand der dort dargestellten Lernverfahren werden verschiedene Ansätze zur Erkennung von und/oder Anpassung an Konzeptverschiebungen erläutert.

Kapitel 5 widmet sich einem der verbreitesten Ansätze, Lernverfahren den Umgang mit sich verändernden Konzepten zu ermöglichen, der Verwaltung von Zeitfenstern auf den Trainingsdaten. Es werden verschiedene statische Formen der Fensterverwaltung beschrieben und eine adaptive Fensterverwaltung vorgestellt, die speziell für die Anwendung in der Textklassikation konzipiert ist. Die Verwendung einer derartiger Zeitfensterverwaltung erlaubt es auch

Lernverfahren, die nicht speziell auf das Lernen sich verändernder Konzepte ausgelegt sind, solche Konzepte zu lernen. Die Performanz der in Kapitel 3 beschriebenen Lernverfahren in Kombination mit den verschiedenen Fensterverwaltungsansätzen wird anhand zwei verschiedener simulierter Szenarien auf realen Textdaten verglichen (siehe Kapitel 6), um zu untersuchen, ob die Verwendung eines Zeitfensters gegenüber dem Lernen auf allen bekannten Daten zu einer Verbesserung der Lernergebnisse führt, und ob die Erkennung von Konzeptverschiebungen bei der adaptiven Fensterverwaltung funktioniert. Zudem wird untersucht, ob die durchgeführten Anpassungen der Fenstergröße beim adaptiven Ansatz zu einer Performanzsteigerung der Lernverfahren gegenüber den statischen Fensterverwaltungsansätzen führen. Abschließend werden die Ergebnisse dieser Arbeit zusammengefaßt (Kapitel 7) und mögliche Erweiterungen der vorgestellten Ansätze sowie weitere Perspektiven diskutiert (Kapitel 8).

Kapitel 2

Informationsfiltern und Textkategorisierung

Im ersten Abschnitt dieses Kapitels werden kurz die Gemeinsamkeiten und Unterschiede des *Informationsfilterns* und des *Information Retrieval* beschrieben. Beide Ansätze der Informationssuche lassen sich unter dem Begriff der *Textkategorisierung* subsumieren, der in Abschnitt 2.2 formal definiert wird. Bevor im Kapitel 3 mehrere Lernalgorithmen zur Textklassifikation vorgestellt werden, werden verschiedene Formen der Textrepräsentation (Abschnitt 2.3) und die Verarbeitungsschritte beim Informationsfiltern (Abschnitt 2.4) beschrieben, die in der Regel unter anderem die Normalisierung von Texten (Abschnitt 2.5), die Auswahl und Gewichtung von Attributen (Abschnitte 2.6 und 2.7) sowie die Bewertung der Klassifikationsverfahren anhand verschiedener Performanzmaße (Abschnitt 2.8) umfassen.

2.1 Die Beziehung zwischen Informationsfiltern und Information Retrieval

Sowohl beim Informationsfiltern als auch beim Information Retrieval geht es darum, Informationen aus großen Mengen von Texten zu finden. Beim *Information Retrieval*, dem älteren der beiden Forschungsgebiete, gilt es, Dokumente aus einer als statisch angenommen Dokumentenkollektion zu finden, die möglichst gut auf eine Benutzeranfrage passen. Die Dokumentenkollektion wird typischerweise gemäß verschiedener, meist ad hoc gestellter Anfragen durchsucht, die von verschiedenen Benutzer gestellt wurden. Ein typisches Beispiel hierfür ist eine elektronische Sammlung von Artikeln oder Forschungsberichten, die von Benutzern mit Hilfe von Anfragen durchsucht werden kann. Jede Anfrage wird in der Regel nur einmal gestellt, eventuell aber durch Feedback von Seiten des Benutzers verfeinert oder erweitert. Der Benutzer kann, wenn er die vom System als für seine Anfrage als relevant betrachteten Dokumente gezeigt bekommt, seine Anfrage entweder in veränderter Form neu stellen oder aber durch Angabe, welcher der präsentierten Dokumente er wirklich als relevant empfand, verfeinern (*Relevance Feedback*, siehe auch Abschnitt 3.1).

Während man beim Information Retrieval von einer statischen Dokumentenkollektion und häufig wechselnden, meist nur einmal von einem Benutzer gestellten Anfragen ausgeht, ist beim *Informationsfiltern* die Anfrage statisch und die Dokumentenkollektion ändert sich ständig. Eine typische Anwendung ist die Weiterleitung von Nachrichten aus einem Nachrichtenstrom (Ticker, Newswire) zu einem bestimmten Thema zu einem Benutzer, der ein bestimmtes Interesse an diesem Thema hat. Die Anfrage des Nutzers besteht über einen längeren Zeitraum und wird in der Regel als ein Interessen- oder Benutzerprofil gespeichert, mit dem neue Nachrichten aus dem Dokumentenstrom verglichen werden, um ihre Relevanz zu bestimmen. Wenn dieses Interessenprofil automatisch den sich eventuell ändernden Wünschen des Benutzers angepaßt werden soll, so ist auch hier Feedback über die Relevanz der präsentierten Dokumente notwendig.

Sowohl beim Information Retrieval als auch beim Informationsfiltern handelt es sich um eine Klassifikation von Texten in *relevant* bzw. *nicht relevant* in Bezug auf ein bestimmtes Nutzerinteresse. Der folgende Abschnitt definiert den Begriff der *Textkategorisierung* genauer. Beiden Informationsucheansätzen gemeinsam ist die Notwendigkeit, Texte und Benutzeranfragen (oder -profile) zu repräsentieren, um Dokumente mit den Anfragen sowie eventuell auch Dokumente untereinander vergleichen zu können. Weiterhin gemein ist beiden Ansätzen der Bedarf an Verfahren, die aus Vorgaben oder Feedback durch den Benutzer, also aus Anfragen und/oder klassifizierten Dokumenten, automatisch ein Klassifikationsmodell generieren, um neue passende Dokumente zu finden. Diese Aspekte werden in den weiteren Abschnitten dieses Kapitels und im folgenden Kapitel über Lernverfahren zur Textkategorisierung behandelt.

2.2 Definition von Textkategorisierung

Die Textkategorisierung ist ein spezielles Klassifikationsproblem, bei dem es darum geht, Dokumente in eine vorgegebene Menge von Klassen einzuteilen. Jede Klasse stellte eine semantische Kategorie dar, wie beispielsweise *relevant*, *nicht relevant* oder ein bestimmtes Thema oder Interesse. Es wurden viele Verfahren entwickelt, die das Konzept, nach dem Dokumente klassifiziert werden sollen, anhand von Trainingsbeispielen lernen. Diese Lernaufgabe fällt in den Bereich des überwachten Lernens (*Supervised Learning*), wie er im maschinellen Lernen und in der Mustererkennung betrachtet wird.

In dieser Arbeit wird folgende Definition der *Textkategorisierung* aus [Joachims, 1996, Seite 21] benutzt. Die Menge der Klassen, $\mathcal{C} = \{C_1, C_2, ...\}$, und somit auch ihre konstante Anzahl sind bekannt. Jedes Dokument wird genau einer Klasse zugewiesen. Es gibt eine Menge von Trainingsbeispielen D, also von Dokumenten, deren Klasse bekannt ist. Außerdem existiert eine Funktion $T(d) \in \mathcal{C}$, die jedem Dokument d eine Klasse zuweist. Für die

Dokumente der Trainingsmenge ist $T(d)$ gegeben. Die Lernaufgabe ist, aus den Trainingsbeispielen eine Hypothese H zu induzieren, die T approximiert. $H(d)$ kann benutzt werden, um vorher ungesehene Dokumente zu klassifizieren. Gesucht ist die Hypothese, die die höchste Genauigkeit erzielt und $T(d)$ am besten approximiert. Diese Definition läßt sich auch dahingehend erweitern, daß Dokumente in mehreren Klassen enhalten sein können, aber dieser Fall wird hier nicht näher betrachtet.

2.3 Repräsentation von Texten

Die Repräsentation von Beispielen und Hypothesen hat einen großen Einfluß auf die Generalisierungsfähigkeit von Lernverfahren. Die Hypothesensprache und somit die Menge der von einem Lernalgorithmus lernbaren Hpothesen wird nicht nur durch den Algorithmus bestimmt, sondern auch entscheidend von der Repräsentation der Beispiele, hier also der Repräsentation der Texte. Dieser Abschnitt stellt eine Auswahl verschiedener Repräsentationsansätze vor, die bei der Textkategorisierung auf der Basis des Textinhaltes, der sogenannten *inhaltsbasierten Textklassifikation*, verwendet werden. Am ausführlichsten werden wortbasierte Repräsentationen behandelt. Diese sind am weitesten verbreitet und liegen auch den in dieser Arbeit beschriebenen Experimenten zugrunde.

Bei der Betrachtung von Wörtern als Indexierungstermen wird meist die vereinfachende Annahme gemacht, daß die Reihenfolge der Wörter im Text vernachlässigt werden kann. Dokumente werden dann nicht als Sequenzen von Wörtern, sondern als *Multimengen („bags“) von Wörtern* dargestellt. Deswegen wird diese Repräsentationsform auch häufig als *Bag-of-Words-Ansatz* bezeichnet. Die Bag-of-Words-Repräsentation ist konsistent mit der im maschinellen Lernen gebräuchlichen Attribut-Wert-Darstellung von Beispielen. Hier ist jedes unterschiedliche Wort ein Attribut und der Wert eines solchen Attributes für ein Dokument ist die Anzahl der Vorkommen des entsprechenden Wortes. Diese Anzahl wird *Term Frequency* $TF(w, d)$ des Wortes w in Dokument d genannt.

Die Repräsentation von Dokumenten als Multimengen von Wörtern ist im Information Retrieval sehr verbreitet und wird dort auch als *Vektorraummodell* bezeichnet. Bei der Transformation eines Textes in diese Repräsentation geht mit der Reihenfolgeinformation der Wörter zwar Information über das Dokument verloren, aber auch komplexere Formalismen, die die Reihenfolge von Wörtern in Betracht ziehen, haben bisher nicht zu konsistent wesentlich besseren Klassifikationsergebnissen geführt (siehe z. B. [Lewis, 1992]). Die Bag-of-Words-Repräsentation scheint ein guter Kompriß zwischen Modellkomplexität und Ausdrucksfähigkeit zu sein. Deswegen und wegen ihrer einfachen Handhabbarkeit, wird sie auch den in dieser Arbeit beschriebenen Experimenten zugrunde gelegt.

Alternativ können Texte auch als *Multimengen von n-Grammen* oder *Multimengen von Wortstämmen* repräsentiert werden. Bei der Textrepräsentation durch Trigramme (oder allgemeiner n-Gramme) werden Dokumente nicht, wie im vorherigen Abschnitt beschrieben, durch die im Dokumenttext vorkommenden Wörter beschrieben, sondern durch alle in ihnen vorkommenden Zeichenketten der Länge drei (bzw. n). Das Wort „Haus" wird beispielsweise auf die Trigram-Multimenge „_Ha", „Hau", „aus" und „us_" abgebildet. n-Gram-Repräsentationen sind relativ robust gegenüber Schreibfehlern und erlauben sprachunabhängig die Erkennung von Wortähnlichkeiten, d. h. von gleichen Wortbestandteilen. Um solche Wortähnlichkeiten beispielsweise bei einer wort-basierten Darstellungen zu berücksichtigen, ist in der Regel mehr Aufwand und außerdem linguistisches Wissen erforderlich, um z. B. eine Stammformreduktion (siehe Abschnitt 2.6) durchführen zu können, womit dieser Schritt dann auch nicht mehr sprachunabhängig ist.

Indixierungsterme müssen nicht unbedingt aus einzelnen Wörtern bestehen, sondern können sich auch aus mehreren Wörtern zusammensetzen. *Mehr-Wort-Terme* lassen sich beispielsweise auf der Basis statistischer Methoden bilden, wobei dann die Satzstruktur nicht betrachtet wird. Statt dessen werden Terme nach Statistiken gebildet, welche Wörter oft zusammen auftreten (*Korrelationsstatistiken*, siehe z. B. [Fuhr et al., 1991]).

Wie in diesem Abschnitt beschrieben, lassen sich Texte (unter anderem) also als

- Multimengen von Wörtern,
- Multimengen von n-Grammen,
- Multimengen von Wortstämmen oder
- Multimengen von Mehr-Wort-Termen

repräsentieren. Offensichtlich verwenden die hier beschriebenen Ansätze keine syntaktische Information über die Struktur der Texte oder ihrer Sätze (wie beispielsweise die Information über die Wortreihenfolge). Stattdessen wird lediglich versucht, über die Häufigkeit des Auftretens bestimmter Attibute (z.B. der Wörter) eine (sehr begrenzte) Art von Semantik dieser Texte zu erfassen.

2.4 Verarbeitungsschritte beim Informationsfiltern

Der Prozeß des Informationsfilterns läßt sich in eine Reihe von Verarbeitungsschritten untergliedern, die in diesem Abschnitt kurz vorgestellt und in den folgenden Abschnitten näher erläutert werden. Der Filterprozeß besteht aus zwei Phasen. In der ersten Phase, der *Anpassungs-* oder *Lernphase* erfolgt eine Anpassung des Systems an die Dokumentenkollektion und das Benutzerinteresse, das aus der gegebenen Klassifikation der Trainingsbeispiele zu induzieren ist. In der zweiten Phase, dem eigentlichen *Filtern*, werden neue Dokumente klassifiziert, weswegen die Phase auch als *Klassifikationsphase* bezeichnet wird. Die folgende Beschreibung der einzelnen Schritte dieser Phasen geht von einem inhaltsbasierten Filtersystem mit wortbasierter Textrepräsentation aus.

Die Anpassungsphase umfaßt im wesentlichen die folgenden fünf Schritte. Zuerst werden die Dokumente, die als einfache Zeichenketten vorliegen, in einem *Textnormalisierung* genannten Vorverarbeitungsschritt in einzlene Zeichen und Zeichenketten, sogenannte Token, zerlegt, von denen solche entfernt werden, die bei der Lösung der Klassifikationsaufgabe eher hinderlich erscheinen (siehe Abschnitt 2.5). Im dann folgenden Schritt der *Termgenerierung* werden aus den verbleibenden Token die Terme gewonnen, mit denen die Dokumente indiziert werden. Dieser Schritt liefert einerseits ein Lexikon der in der Dokumentenkollektion vorkommenden Attribute und transformiert andererseits die Dokumente der Kollektion in die im vorherigen Abschnitt beschriebene Attribut-Wert-Vektordarstellung (Abschnitt 2.3). Anschließend kann als nächster Schritt eine *Attributauswahl* erfolgen, die je nach Art entweder eine Teilmenge der bestehenden Attribute verwendet oder die Attribute in eine neue, kompaktere Darstellung transformiert (Abschnitt 2.6). In entsprechender Weise werden das Lexikon und die Dokumentvektoren dieser Attributauswahl angepaßt. Meistens erfolgt dann eine *Attributgewichtung* und eine Normierung der Dokumentvektoren (Abschnitt 2.7). Basierend auf den Trainingsdokumenten in dieser Darstellung erfolgt schließlich der *Lernschritt*, der das gewünschte Klassifikationsmodell mit seinen Parametern liefert (Kapitel 3).

In der Klassifikationsphase durchlaufen neue Dokumente fast die gleichen Schritte wie zuvor die Trainingsdokumente. Zuerst durchläuft ein neues Dokument die *Textnormalisierung* und wird in der *Indexierung* gemäß dem Attributlexikon in einen Vektor transformiert. Entsprechend der in der Lernphase durchgeführten *Attributauswahl* und *Attributgewichtung* erfolgt gegebenenfalls eine Transformation und/oder eine Gewichtung des Dokumentvektors, der dann im *Klassifikationsschritt* mit Hilfe des in der Lernphase erworbenen Modells klassifiziert wird. Auf der Basis dieser Klassifikation (und gegebenenfalls ihrer Konfidenz) wird das neue Dokument dem Benutzer entweder als relevant präsentiert oder nicht.

Wenn der Benutzer die ihm präsentierten Dokumente bzw. die Vorhersagen über ihre Klassifizierung bewertet, kann man die Performanz der Lernverfahren anhand verschiedener *Performanzmaße* beurteilen bzw. vergleichen (Abschnitt 2.8). Außerdem kann dieses Feedback des Benutzers zu einer erneuten Adaptions- oder Lernphase herangezogen werden. Kapitel 5 beschreibt verschiedene Möglichkeiten, auch die alten Trainingsdokumente ganz oder teilweise in diesen erneuten Adaptions- oder Lernschritt mit einzubeziehen.

2.5 Normalisierung von Texten

Vor der Termgenerierung, also bevor die Dokumente in eine der in Abschnitt 2.3 beschriebenen Repräsentationen überführt werden, ist in der Regel eine gewisse Vorverarbeitung der Texte notwendig, die sogenannte *Textnormalisierung*. Dieser Arbeitsschritt dient der Entfernung unerwünschter *Token*, d. h. von Zeichen und Zeichenketten, die für die Klassifikation des Textes eher hinderlich erscheinen. Die Eingabe dieses Schrittes sind Textdateien, die die zu verarbeitenden Texte enthalten.

Handelt es sich bei den Texten beispielsweise um HTML-Dokumente (oder SGML-Dokumente), so werden die HTML-Tags (bzw. SGML-Tags) entfernt und so der eigentliche Text aus dem Dokument extrahiert. Bei E-Mail- oder elektronischen Zeitungsnachrichten (News) werden die „Subject"-Zeile und der eigentliche Nachrichtentext extrahiert, nicht aber der Kopf des Dokuments (MIME-Header). Außerdem werden Interpunktionszeichen und oft auch Sonderzeichen aus den Texten entfernt. Manchmal werden hierbei Token, die Ziffern oder Sonderzeichen enthalten auch vollständig entfernt. Sehr oft werden die verbleibenden Token in Kleinbuchstaben umgewandelt. Das Resultat dieses Verabeitungsschrittes sind die Texte als Sequenzen von Token.

Für die in dieser Arbeit beschriebenen Experimente werden aus allen Dokumenten eventuell vorhandene SGML-Tags und Sonderzeichen entfernt. Token, die nur aus Ziffern bestehen, werden ebenso entfernt. In alphanumerischen Token, die neben Ziffern auch andere Zeichen enthalten, werden alle Ziffern in Einsen transformiert, also z. B. von „45billion" in „11billion".

Je nach Betrachtungsweise kann man auch die in Textklassifikationssystemen oft durchgeführten Schritte der *Stammfomreduktion* und der *Entfernung von Stoppwörtern* unter die Textnormalisierung fassen. In der hier vorgenommen Beschreibung sind diese Schritte jedoch der im nächsten Abschnitt beschriebenen Attributauswahl zugeordnet.

2.6 Attributauswahl

Das Hauptziel der *Attributauswahl*, auch *Attributselektion* genannt, ist eine *Dimensionsreduktion* des Repräsentationsraumes für die Dokumente möglichst ohne Verlust von für die Klassifikation wichtiger Information. Ein wichtiger Grund für dieses Ziel ist die Tatsache, daß die Laufzeit der meisten Lernverfahren sehr stark von der Anzahl der Attribute in der Beispielrepräsentation beeinflußt wird. Neben einem deutlichen Laufzeitvorteil ermöglicht die Attributauswahl außerdem eine kompaktere, also Speicherplatz sparendere Speicherung und Verarbeitung der Dokumente. Aus dem maschinellen Lernen ist aber auch bekannt, das durch die Auswahl eines „guten" Attributsatzes oftmals die Performanz eines Lernsystems optimiert werden kann. Ein guter

Attributsatz zeichnet sich dadurch aus, daß er zum einen genug Information zum Lösen des Lernproblems bereitstellt, zum anderen aber keine irrelevanten Attribute enthält.

Es gibt eine Vielzahl von Selektionskriterien für die Attributauswahl, von denen hier nur einige der in der Textkategorisierung gebräuchlichsten aufgeführt werden. Die folgenden Selektionsverfahren lassen sich nach verschiedenen Kriterien gliedern. Das hier verwendete Kriterium unterscheidet die Selektionsverfahren danach, ob sie die gegebene Attributmenge auf eine *Teilmenge der gegebenen Attributemenge* abbilden, oder ob sie auch *neue Attribute einführen*, also auch die Repräsentation verändern. Entsprechend der Zuordnung zu einer dieser beiden Kategorien werden die Selektionsverfahren in einem der beiden folgenden Unterabschnitte beschrieben.

Alternativ kann man die Kriterien auch nach ihrer Funktionsweise gliedern und so linguistische und statistische Ansätze unterscheiden. *Linguistische Selektionsverfahren* wie die im folgenden beschriebene Stoppworteliminierung, Stammformreduktion, oder Wortartauswahl benötigen linguistisches Wissen und erfordern teilweise eine morphologische Analyse der Wörter, wodurch sie sprachabhängig sind. Die anderen beschriebenen Verfahren lassen sich als *statistische Selektionsverfahren* klassifizieren, die sprachunabhängig arbeiten.

2.6.1 Selektion einer Teilmenge von Attributen

Die verbreiteste Methode zur Attributauswahl ist die *Stoppwortelimination*. Die Annahme ist, daß extrem häufige Wörter wie beispielsweise Artikel und Konjunktionen, die in den meisten Dokumenten vorkommen, wenig über den Inhalt einzelner Dokumente aussagen und deswegen unabhängig vom Klassifikationsproblem nicht relevant sind. Normalerweise werden statische, von Hand erstellte Stoppwortlisten verwendet. Alternative kann man aber auch die n häufigsten Terme der Dokumentenkollektion entfernen [Lang, 1995], wodurch die „Stoppwortliste“ automatisch erstellt werden kann und Korpus-abhängig wird.

Während bei der Stoppwortelimination Wörter entfernt werden, die in der Regel besonders häufig vorkommen, werden beim *Document Frequency Tresholding* [Yang und Pedersen, 1997] im Gegenteil gerade die besonders seltenen Wörter aus der Attributmenge gelöscht. Es werden nur die Wörter selektiert, die in mindestens n Dokumenten der Kollektion vorkommen. Durch die Wahl von n läßt sich also die Anzahl der selektierten Attribute steuern. Dieses Selektionsverfahren basiert auf der Annahme, daß nur selten vorkommende Wörter entweder nicht informativ sind oder durch ihr seltenes Auftreten die Performanz des Lernsystems nur unwesentlich beeinflußen [Yang und Pedersen, 1997]. Wenn es sich bei den seltenen Termen um Rauschen in den Daten, also z. B. falsch geschriebene Wörter handelt, kann dieser Schritt sogar eine Steigerung der Performanz des Lernsystems zur Folge haben.

Eine weitere linguistische Methode zur Attributauswahl ist die *Selektion nach Wortart*, d. h. nur bestimmte Wortarten wie beispielsweise Substantive, Eigennamen und Adjektive werden ausgewählt, die übrigen werden verworfen. Experimente zum automatischen Clustern deutschsprachiger Zeitungsartikel haben z. B. gezeigt, daß Substantive und Eigennamen bei der Charakterisierung von Textinhalten sehr dominant sind und bei der Verwendung ohne andere Wortarten bereits zu sehr guten Clustern führen [Schewe, 1997]. Während die Hinzunahme von Adjektiven die Ergebnisse noch geringfügig verbesserte, veränderte die zusätzliche Hinzunahme von Verben das Ergbnis nicht weiter. Andere Wortarten wie Artikel, Konjunktionen, Pronomen oder Präpositionen werden oft schon durch Stoppwortlisten entfernt.

Während die bisher beschriebenen Verfahren zur Attributauswahl nur Kollektionsstatistiken oder linguistisches Wissens verwendet haben, nicht aber das konkrete Klassifizierungsproblem in Betracht gezogen haben, folgt nun die Beschreibung von Selektionskriterien, die versuchen, möglichst relevante Attribute im Hinblick auf die Klassifikation der Dokumente auszuwählen.

Das Auswahlkriterium *Information Gain* aus der Informationstheorie wird im maschinellen Lernen sowohl im anderen Domänen (siehe z. B. [Quinlan, 1986] und [Mitchell, 1997]) als auch in der Textkategorisation sehr häufig eingesetzt (siehe z. B. [Yang und Pedersen, 1997]). Teilweise wird dieses Maß auch als *Mutual Information* bezeichnet (siehe z. B. [Joachims, 1996]). Es mißt die Anzahl der Bits, die für die Vorhersage der Klassifikation an Information gewonnen werden, wenn die Anwesenheit oder das Fehlen eines Wortes w in einem Dokument d bekannt ist. Sei $\mathcal{C} = \{C_1, C_2, ...\}$ die Menge der Klassen, sei $T(d)$ die Klasse von d und sei $OCC(w,d) = 1$, wenn w in d vorkommt, und $OCC(w,d) = 0$, anderenfalls, dann berechnet sich das Information Gain des Wortes w wie folgt:

$$\begin{aligned} IG(w) \quad = \quad & -\sum_{C\in\mathcal{C}} \Pr(T(d)=C)\cdot\log\Pr(T(d)=C) \\ & +\sum_{C\in\mathcal{C}} \Pr(T(d)=C\wedge\neg OCC(w,d))\cdot\log\Pr(T(d)=C|\neg OCC(w,d)) \\ & +\sum_{C\in\mathcal{C}} \Pr(T(d)=C\wedge OCC(w,d))\cdot\log\Pr(T(d)=C|OCC(w,d)) \end{aligned} \tag{2.1}$$

$IG(w)$ ist also ein Maß für die Informationsmenge, die das Vorkommen eines Wortes w unabhängig von anderen Wörtern in Bezug auf die Klasse eines Dokumentes liefert[1]. Bei der Attributauswahl gemäß Information Gain werden die Wörter mit den höchsten $IG(w)$ selektiert.

χ^2 *Statistiken* stellen ein weiteres, auf das Klassifikationsproblem bezogenes Maß für die Attributauswahl dar (siehe z. B. [Yang und Pedersen, 1997]). χ^2 Statistiken messen die Abhängigkeit der Vorkommenswahrscheinlichkeit eines Wortes in einem Dokument von der Klassifikation des Dokumentes. Diese Statistiken haben den Nachteil, daß sie bei selten vorkommenden Wörtern nicht

[1]In Abschnitt 3.8 ist das Information Gain Maß detailierter beschrieben. In der dortigen Beschreibung wäre der Attributtest X dann $OCC(w,d)$ mit den Ausgängen *wahr* und *falsch*.

mehr zuverlässig sind. Yang und Pedersen haben mehrere Maße zur Attributauswahl für statistische Lernverfahren in der Textkategorisierung untersucht und stellten bei ihren Experimenten auf zwei Textkollektionen fest, daß Information Gain und die von ihnen gewählte χ^2 Statistik auf diesen Daten die günstigste Attributauswahl leisteten. Auch das relativ einfache und weniger aufwendige Document Frequency Tresholding erzielte relativ gute Ergebnisse.

2.6.2 Attributselektion und Veränderung der Repräsentation

Das Ziel der in diesem Abschnitt beschriebenen Verfahren ist die Transformation der Attributmenge in eine kleinere Menge, die die Beispiele „effizienter“ beschreiben sollen. Es soll eine Dimensionsreduktion erreicht werden, bei der möglichst wenig relevante Information über die Beispiele verloren geht. Dieser Schritt wird auch als *Reparametrisierung* bezeichnet.

Die *Grundformreduktion* ist ein morphologisches Verfahren, daß Wörter auf ihre Grundform reduziert, d. h. Verben werden auf den Infinitiv reduziert, Substantive auf den Nominativ Singular. Ein ähnliches Verfahren ist die *Stammformreduktion*, bei der die Wörter nicht auf ihre Grundform, sondern auf ihren Wortstamm zurückgeführt werden. So werden im Englischen beispielsweise die Wörter „computer“, „computing“ und „computability“ auf das gleiche Attribut „comput“ abgebildet. In Sprachen mit vielen Komposita, wie beispielsweise der deutschen Sprache, kann ein Wort auch auf mehrer Stämme abgebildet werden, so daß theoretisch die Anzahl der Attribute sogar steigen kann.

Beim Einsatz von *Thesauri* sollen Wörter nach semantischen Kategorien zusammengefaßt werden. Ein Thesaurus speichert Relationen zwischen Wörtern. So können beispielsweise Synonyme in Äquivalenzklassen zusammengefaßt werden. Häufig sind in Thesauri auch weitergehende Relationen wie „ist spezieller als“, „ist genereller als“, „ist Teil von“, „besteht aus“, etc. enthalten.

Weitere Attributselektionsverfahren, die eine Reparametrisierung vornehmen, hier aber nicht ausführlicher beschrieben werden, sind das auf der Singular Value Decomposition (SVD) beruhende *Latent Semantic Indexing* [Deerwester et al., 1990] und das *Term Clustering*, bei dem versucht wird, semantisch ähnliche Terme mit Methoden der Clusteranalyse zusammenzufassen (siehe z. B. [Lebowitz, 1987], [Cheeseman et al., 1988]).

2.6.3 Kombination mehrerer Verfahren zur Attributauswahl

Oft werden verschiedene Verfahren der Attributauswahl kombiniert, z. B. indem zuerst eine Stammformreduktion durchgeführt wird, dann die Stoppwörter sowie eventuell nur in ein oder zwei Texten auftretende Terme entfernt werden und schließlich eine Auswahl aus den verbleibenden Attributen mittels Information Gain oder eines anderen statistischen Maßes erfolgt.

In den in dieser Arbeit beschriebenen Experimenten wird keine Stammformreduktion durchgeführt, aber es erfolgt eine Entfernung von Stoppwörtern aus einer vorgegebenen Stoppwortliste und es werden nur Terme mit einer Mindesthäufigkeit von drei zur Dokumentrepräsentation verwendet. Anschließend erfolgt für einen Teil der Verfahren eine Attributselektion mit Hilfe des Information Gain Maßes.

2.7 Attributgewichtung

In diesem Abschnitt wird die in der Textkategorisierung sehr weit verbreitete *TFIDF-Gewichtung* der Attribute in den Dokumentvektoren beschrieben [Salton und Buckley, 1988]. Mittels der *Termfrequenz (Term Frequency)* und der *Dokumentfrequenz (Document Frequency)* der Wörter eines Dokumentes aus einer Dokumentenkollektion wird eine Gewichtung der Wörter in der Vektorraumdarstellung durchgeführt. Die *Term Frequency* $TF(w,d)$ gibt an, wie oft das Wort w in Dokument d auftritt. Die *Document Frequency* $DF(w)$ ist die Anzahl der Dokumente, in den das Wort w mindestens einmal auftritt. Die Grundidee bzw. Annahme hierbei ist, daß Wörter, die in wenigen Dokumenten vorkommen, aber häufig im aktuellen Dokument auftreten, sehr gute Deskriptoren für den Inhalt dieses Dokumentes sind. Ein hohes Gewicht sollen also intuitiv Wörter mit einem hohen $TF(w,d)$-Wert und einem geringen $DF(w)$-Wert erhalten. Letztere Anforderung kommt in der Definition der sogenannten *inversen Dokumentfrequnez (Inverse Document Frequency)* zum Ausdruck:

$$IDF(w) = \log \frac{|D|}{DF(w)} \tag{2.2}$$

Dabei ist D die Dokumentenkollektion, die der Gewichtung zugrunde liegt (Trainingsmenge), also $|D|$ die Anzahl der Dokumente in dieser Kollektion. Das *TFIDF-Gewicht* eines Wortes w im Dokumentvektor eines Dokumentes d kombiniert die beiden geschilderten Anforderungen wie folgt:

$$TFIDF(w,d) = TF(w,d) \cdot IDF(w) \tag{2.3}$$

Die Repräsentation eines Dokumentes (oder einer Anfrage) d ergibt sich entsprechend aus der elementweisen Multiplikation des Vektors der Term Frequency Werte des Dokumentes mit dem IDF-Vektor der zugrundeliegenden Dokumentenkollektion:

$$\vec{d} = \overrightarrow{TF}\,(d) \;*\; \overrightarrow{IDF} \tag{2.4}$$

In der Regel wird nach der Gewichtung der Vektorkomponenten eine Normierung der Dokumentvektoren auf die (euklidische) Länge Eins vorgenommen:

$$\vec{d'} = \frac{\vec{d}}{||\vec{d}\,||} \tag{2.5}$$

	T(d)=„relevant“	T(d)=„nicht relevant“
H(d)=„relevant“	A	B
H(d)=„nicht relevant“	C	D

Abbildung 2.1: Contingency Table für ein Klassifikationsproblem mit den zwei Klassen „relevant“ (positive Beispiele) und „nicht relevant“ (negative Beispiele). $T(d)$ entspricht der korrekten Klassifikation, $H(d)$ ist die Klassifikation des Lernalgorithmus (Hypothese).

$||\vec{d}\,||$ bezeichnet dabei die Länge des Vektors $\vec{d}$. Durch die Normierung wird die Repräsentation von der Länge des Dokuments und des zugehörigen Vektors unabhängig. Eine Übersicht über andere Gewichtungsformeln als die hier vorgestellte TFIDF-Gewichtung findet sich in [Salton und Buckley, 1988].

2.8 Performanzmaße

In diesem Abschnitt wird eine Auswahl der im maschinellen Lernen und im Information Retrieval verwendeten Performanzmaße vorgestellt. Der Schwerpunkt liegt hierbei auf dem Maß *Accuracy*, im Deutschen auch Klassifikationsgenauigkeit genannt, und den Maßen *Precision* und *Recall*, da sie bei den für diese Arbeit durchgeführten Experimenten zur Bewertung eingesetzt werden. Diese Maße werden anhand von Abbildung 2.1 erläutert, die eine sogenannte *Contingency Table* für ein Klassifikationsproblem mit den beiden Klassen „relevant“ und „nicht relevant“ zeigt. A beschreibt die Anzahl der relevanten Beispiele, die vom Klassifikator als relevant erkannt wurden. B ist die Anzahl vom Lernalgorithmus fälschlicherweise als relevant klassifizierter, in Wirklichkeit nicht relevanter Beispiele. C gibt die Anzahl relevanter Beispiele an, die vom Lernalgorithmus fälschlicherweise als irrelevant klassifiziert wurden, und D ist die Anzahl vom Algorithmus korrekt als irrelevant erkannter Beispiele. Diese Anzahlwerte beziehen sich in der Regel auf die Klassifikation nicht zum Lernen herangezogener Beispiele (Testmenge).

Das im maschinellen Lernen am häufigsten verwendete Performanzmaß für Lernverfahren ist das Maß *Accuracy*, das die Wahrscheinlichkeit angibt, daß ein zufällig aus der Verteilung gezogenes Beispiel richtig klassifiziert wird. Alternativ wird oft die als *Fehlerrate* bezeichnete Gegenwahrscheinlichkeit betrachtet. Die Accuracy läßt sich abschätzen, indem man die Anzahl der korrekten Klassifikationen durch die Anzahl aller Klassifikationen dividiert. Für ein Klassifikationsproblem mit zwei Klassen ergibt sich in Bezug auf Abbildung 2.1 folgende Formel:

$$Accuracy = \frac{A+D}{A+B+C+D} = 1 - Fehlerrate \tag{2.6}$$

Die Accuracy ist ein sehr allgemein anwendbares und anschauliches Maß, aber es gibt auch Anwendungen, in denen der Einsatz dieses Maßes problematisch ist, und man besser auf andere Maße zurückgreift, die die Performanz des Klassifikationssystems praxisorientierter messen. Bei manchen Problemen sind Fehlklassifizierungen in einer Klasse mit höheren Kosten verbunden als in anderen Klassen. Wenn es beispielsweise um die Entscheidung über die Behandlung einer lebensbedrohlichen Krankheit geht, ist es in der Regel schlimmer einen kranken Patienten für gesund zu halten und nicht zu behandeln als einen gesunden Patienten fälschlicherweise einer Behandlung zu unterziehen. In solchen Fällen kann z. B. die Verwendung eines Kostenmodells (siehe z. B. [Michie et al., 1994]) oder die getrennte Betrachtung der Klassifikationsgenauigkeit der Instanzen einzelner Klassen hilfreich sein (siehe z. B. [Klinkenberg, 1996], [Klinkenberg und St. Clair, 1996]). Letztere Betrachtungsweise ist auch dann interessant, wenn eine sehr ungleichmässige Klassenverteilung vorliegt, weil sie eine differenziertere Analyse der Performanz eines Klassifikationssystems erlaubt.

Im Information Retrieval erfolgt die Performanzmessung meistens mit Hilfe *Precision/Recall*-basierter Maße, die sich nur auf Klassifikationsprobleme mit zwei Klassen anwenden lassen. Der Klasse der relevanten Dokumente steht die meist deutlich größere Anzahl der nicht relevanten gegenüber. Unter *Precision* versteht man die Wahrscheinlichkeit, daß ein vom System als relevant bezeichnetes Dokument wirklich relevant ist. *Recall* ist die Wahrscheinlichkeit, daß ein relevantes Dokument vom System auch als solches erkannt wird. Anhand der Contingency Table aus Abbildung 2.1 lassen sich diese beiden Maße wie folgt abschätzen:

$$Precision = \frac{A}{A+B} \tag{2.7}$$

$$Recall = \frac{A}{A+C} \tag{2.8}$$

Wenn man die vom System klassifizierten Beispiele anhand ihrer geschätzten Relevanz ordnet und verschiedene Schwellwerte für die Entscheidung zwischen relevant und nicht relevant betrachtet, sieht man, daß sich ein Trade-Off zwischen Precision und Recall ergibt. Um diesen zu erfassen, betrachtet man sogenannte *Precision/Recall-Graphen*. Hierzu wird der Recall künstlich verändert und die Precision wird an festgelegten Recall-Punkten gemessen. Die Recall-Punkte erhält man durch den Abstieg in der Rangordnung der Dokumente, wobei gegebenenfalls interpoliert wird.

Wenn die Rangordnung der Dokumente nur schwach geordnet ist, ist zu beachten, daß mehrere Dokumente den gleichen Rang haben können. Die von Raghavan et al. beschriebene PRR-Methode untersucht für den jeweils letzten Rang, wieviele irrelevante Dokumente erwartungsgemäß für einen gewünschten Recall-Wert $recall$ in Kauf genommen werden müssen [Raghavan et al., 1989]. Sei n die Gesamtzahl der relevanten Dokumente. Sei r die Anzahl der relevanten und i die Anzahl der irrelevanten Dokumente im letzten teilweise relevanten Rang, j die Anzahl der irrelevanten Dokumenten in Rängen oberhalb dieses letzten teilweise relevanten Ranges und s die Anzahl der noch für das Erreichen des gewünschten Recall-Wertes benötigten relevanten Dokumente des letzten teilweise relevanten Ranges, dann berechnet sich die Precision nach der PRR-Methode als:

$$Precision(recall) = \frac{recall \cdot n}{recall \cdot n + j + s \cdot i/(r+1)} \tag{2.9}$$

Zur Mittelung über die Precision/Recall-Graphen mehrerer binärer Klassifikationsaufgaben wird in der Regel eine der beiden folgenden Methoden angewandt. Bei der *Makrobewertung* werden die Precision/Recall-Graphen für jede Aufgabe getrennt berechnet und anschließend gemittelt. Wenn bei einer Aufgabe keines der Testbeispiele relevant ist, entsteht im Nenner der Recall-Formel (Formel 2.8) eine Null, so daß dieser Spezialfall gesondert zu behandeln ist (siehe z. B. [Fuhr und Knorz, 1984]). Die *Mikrobewertung* bereitet in dieser Hinsicht weniger Probleme. Hier werden die Klassifizierung aller Aufgaben zu einer Rangordnung zusammengefaßt, für die dann ganz normal Precision und Recall berechnet wird. Dabei muß allerdings die Kompatibilität der Relevanzmaße zwischen den Aufgaben gewährleistet sein.

Ein verbreitetes Maß, um die Bewertung eines Systems wie bei der Accuracy auf eine einzelne Zahl zu bringen, ist der sogenannte *Precision-Recall-Breakeven-Punkt* [Lewis, 1992]. Der Breakeven-Punkt ist der Punkt, an dem die Precision gleich dem Recall ist, also der Schnittpunkt des Precision/Recall-Graphen mit der Identitätsfunktion. Wenn es mehrere solcher Schnittpunkte gibt, wird in der Regel der höchste gewählt. Alternativ wird manchmal auch der *Mittelwert der Precision* über verschiedenen Recall-Punkten berechnet (z. B. über 0.1, 0.2, ..., 1.0). Während der Break-Even-Point ein an einer binären Klassifikation orientiertes Maß ist, eignet sich der Mittelwert der Precision eher für eine rangordnungsorientierte Verfahrensbewertung.

Anwendungsabhängig können auch andere Maße wie beispielsweise die *Precision at n*, d. h. die erwartete Anzahl wirklich relevanter Dokumente unter den n Dokumenten mit dem höchsten Rang dividiert durch n, interessant sein. Ein Beispiel hierfür wäre ein Benutzer, der jeden Tag die n (nach Meinung des Systems) für ihn wichtigsten Nachrichten aus einem Strom von Zeitungsartikeln lesen will. Analog wäre es eventuell sinnvoll die *Accuracy at n* zu betrachten, wenn in einem nur teilautomatisierten System die Fälle mit der größten Unsicherheit manuell entschieden werden sollen. Diese beiden Maße sind allerdings eher ungebräuchlich. Sie sind hier nur erwähnt, um zu zeigen, daß die Wahl des Performanzmaßes zur Beurteilung eines Klassifikationsverfahrens auch von der geplanten Anwendung abhängt.

Im Rahmen der Experimente dieser Arbeit sind Accuracy, Precision und Recall die zentralen Bewertungsmaße. Die Maße werden im simulierten Zeitablauf sowie gemittelt über den Zeitablauf betrachtet. Die Betrachtung der Maße im Zeitablauf ermöglicht es zu sehen, wie schnell und wie gut sich ein Verfahren an im Zeitablauf auftretende Konzeptverschiebungen anpassen kann.

Ein in realen Anwendungen nicht zu unterschätzendes, hier aber nur kurz erwähntes Problem ist die Verfügbarkeit geeigneter Daten für die Beurteilung eines eingesetzten Verfahrens. Beim überwachten Lernen geht man von der Verfügbarkeit klassifiziert vorliegender Beispiele für die Trainings- und Testmenge aus. In realen Anwendungen gibt es aber nicht immer auch ein Feedback über die vom System durchgeführten Klassifikationen, und falls es Feedback gibt, ist die Frage, wie repräsentativ dieses Feedback für das eigentlich zu maximierende Maß, die Benutzerzufriedenheit, ist. Während in den für diese Arbeit durchgeführten Experimenten angenommen wird, daß es ein vollständiges Feedback für alle Beispiele gibt, gibt es in realen Anwendungen oft nur ein Feedback über die von System für relevant gehaltenen Dokumente, nicht aber über die Dokumente, die das System für irrelevant hielt und dem Benutzer gar nicht erst gezeigt hat.

Kapitel 3

Lernverfahren zur Textkategorisierung

Eine Vielzahl von Lernverfahren ist bereits für die Textkategorisierung eingesetzt worden. Dieses Kapitel stellt eine kleine Auswahl dieser Lernverfahren vor, wobei der Schwerpunkt auf der Darstellung der in den Experimenten zu dieser Arbeit eingesetzten Verfahren liegt. Eine formale Definition der Lernaufgabe bei der Textkategorisierung wurde bereits in Abschnitt 2.2 vorgenommen. Kurz zusammengefaßt ist die Aufgabe das Lernen eines Klassifikators für eine feste, bekannte Anzahl von Klassen anhand einer gegebenen Trainingsmenge von Dokumenten, deren Klassifizierung bekannt ist. Dieser Klassifikator soll anschließend die automatische Klassifikation neuer, vorher ungesehener Dokumente ermöglichen und dabei eine maximale Übereinstimmung mit der realen Klassifikation gemäß des unbekannten Zielkonzeptes erreichen. Die Dokumente, also die Trainings- und Testbeispiele, liegen bei allen in diesem Kapitel beschriebenen Verfahren gemäß der Bag-of-Words-Repräsentation als Attributwertvektoren vor. Die Repräsentation der Hypothesen ist je nach Verfahren unterschiedlich.

Bei den in diesem Kapitel beschriebenen Verfahren handelt es sich um Verfahren zur Textklassifikation, bei denen das Konzept nach dem die Texte zu klassifizieren sind, als stabil, also sich nicht verändernd, angenommen wird. Eine Ausnahme hierbei bildet im gewissen Sinne der in Abschnitt 3.5 beschriebene Winnow-Algorithmus, ein Online-Lernverfahren, das sich auch ohne weitere Anpassung für das Lernen sich verändernder Konzepte einsetzen lassen könnte, auch wenn es nicht dazu konzipiert wurde. Im Rahmen dieser Arbeit wird Winnow allerdings gemäß seiner Konzeption als Lernverfahren für statische Konzepte eingesetzt.

Kapitel 5 beschreibt Ansätze für einen dynamischen Rahmen für Lernverfahren für statische Konzepte, der ihnen das Lernen sich verändernder Konzepte erlaubt. In Kapitel 6 werden diese Ansätze anhand eines Teils der in diesem Kapitel beschriebenen Lernverfahren mit Hilfe mehrerer Experimente evaluiert.

3.1 Der Rocchio-Algorithmus

Der *Rocchio-Algorithmus* [Rocchio Jr., 1971] ist eines der am häufigsten eingesetzten Verfahren in der Textkategorisierung und wird sehr oft als Vergleichsmaßstab für andere Verfahren herangezogen. Er wurde ursprünglich als Methode zum *Relevance Feedback* im Rahmen des Information-Retrieval-Systems SMART [Salton, 1971] entwickelt und später für die Textkategorisierung leicht modifiziert. In diesem Abschnitt wird zuerst die ursprüngliche Version für den Einsatz im Information Retrieval beschrieben und im Anschluß zwei in der Textkategorisierung gebräuchliche Varianten.

Entsprechend dem Vektorraum-Retrieval-Modell [Salton, 1991] werden Dokumente und Anfragen wie in Abschnitt 2.3 beschrieben als Wortvektoren repräsentiert. Diese Wortvektoren werden wie in Abschnitt 2.7 dargestellt gemäß der TFIDF-Heuristik gewichtet und anschließend in der Regel auf die euklidische Länge Eins normiert.

Es wird ein *Ähnlichkeitsmaß* definiert, welches die semantische Ähnlichkeit zweier Vektoren messen soll. Stellt ein Benutzer eine Anfrage an das Retrieval-System, so werden die Dokumente der Kollektion gemäß ihrer so definierten Ähnlichkeit zur Anfrage sortiert, denn es wird angenommen, daß ein Dokument um so relevanter ist, je ähnlicher es der gestellten Anfrage ist. Als Maß für die Ähnlichkeit zweier Dokumente (oder eines Dokumentes und einer Anfrage), wird der Kosinus des Winkels zwischen den zugehörigen Vektoren benutzt. Der Kosinus des Winkels zwischen zwei Dokumentvektoren $\vec{d_1}$ und $\vec{d_2}$ berechnet sich wie folgt:

$$cos(\angle(\vec{d_1}, \vec{d_2})) = \frac{\vec{d_1} \cdot \vec{d_2}}{||\vec{d_1}|| \cdot ||\vec{d_2}||} = \frac{\vec{d_1} \cdot \vec{d_2}}{\sqrt{\vec{d_1} \cdot \vec{d_1}} \cdot \sqrt{\vec{d_2} \cdot \vec{d_2}}} \tag{3.1}$$

Dieses Ähnlichkeitsmaß ist von der Länge der Dokumente und der zugehörigen Dokumentvektoren unabhängig. Es lassen sich auch andere Wortgewichtungen, Normierungen der Vektorlänge oder Ähnlichkeitsmaße (siehe z. B. [Wang et al., 1992]) einsetzen, wodurch Varianten des Verfahrens entstehen (siehe z. B. [Joachims, 1997a] sowie den Abschnitt 3.3 über das Verfahren PrTFIDF). Auf dem Rocchio-Algorithmus basierende Klassifikationsverfahren werden auch oft als *TFIDF-Klassifikatoren* bezeichnet.

Beim *Relevance Feedback*, einem iterativen interaktiven Retrieval-Prozeß, präsentiert das Retrievalsystem dem Benutzer eine Rangordung von Dokumenten in Bezug auf seine Anfrage $\vec{q}$, aus der der Benutzer eine Auswahl von Dokumenten als relevant bzw. nicht relevant kennzeichnet. Der *Rocchio-Algorithmus* beschreibt mit folgender Formel, wie mit Hilfe dieser Kennzeichnungen eine neue Anfrage $\vec{q'}$ zu erzeugen ist. Die relevanten Dokumente werden zur ursprünglichen Anfrage addiert, während die irrelevanten von ihr subtrahiert werden:

$$\vec{q'} = \vec{q} + \alpha \frac{1}{|D_+|} \sum_{\vec{d} \in D_+} \vec{d} - \beta \frac{1}{|D_-|} \sum_{\vec{d} \in D_-} \vec{d} \tag{3.2}$$

Dabei ist D_+ die Menge der vom Benutzer als relevant gekennzeichneten Dokumente und D_- die der nicht relevanten Dokumente. $|D_+|$ und $|D_-|$ bezeichnen die Kardinalitäten dieser Mengen. Über die Parameter α und β läßt sich der Einfluß der relevanten und nicht relevanten Dokumente auf die neue Anfrage einstellen. Vektorkomponenten von $\vec{q\prime}$, deren Wert negativ ist, werden gemäß Rocchio auf Null gesetzt [Rocchio Jr., 1971]. Diese Form der Anfrageverfeinerung kann beliebig oft wiederholt werden und soll dem Benutzer den Suchprozeß dadurch erleichtern, daß er sich nicht spezielle charakteristische Terme zur Verfeinierung seiner Anfrage überlegen muß, sondern nur noch anhand der Textkennzeichnungen beschreiben braucht, was er für relevant hält und was nicht.

Der Rocchio-Algorithmus läßt sich relativ einfach für den Einsatz bei binären Textklassifikationsproblemen anpassen. Seien die Klassen o. B. d. A. mit + und − bezeichnet. Nun sind $|D_+|$ nicht die vom Benutzer als relevant markierten Dokumente, sondern die Trainingsdokumente, die sich in der Klasse + befinden. Entsprechend ist $|D_-|$ die Menge der Trainingsdokumente der anderen Klasse. Da es keine initiale Anfrage $\vec{q}$ gibt, sieht die modifizierte Formel wie folgt aus [Joachims, 1996]:

$$\vec{r} = \alpha \frac{1}{|D_+|} \sum_{\vec{d} \in D_+} \vec{d} - \beta \frac{1}{|D_-|} \sum_{\vec{d} \in D_-} \vec{d} \tag{3.3}$$

Auch hier werden Komponenten von $\vec{r}$, deren Wert negativ ist, auf Null gesetzt. Anstelle den relativen Einfluß der positiven und negativen Beispiele auf die Richtung des Vektors $\vec{r}$ über α *und* β einzustellen, reicht es beispielsweise aus, α konstant auf Eins zu setzen und den relativen Einfluß allein über β zu regulieren, da jede von α verursachte Richtungsänderung des Vektors ebenso durch eine umgekehrt proportionale Veränderung von β erreichbar ist (außer der Einstellung $\alpha = 0$, die aber auch keinen Sinn macht). Hierbei kann sich zwar die Länge des Vektors $\vec{r}$ ändern, wenn man aber das von der Dokumentlänge unabhängige Kosinusmaß verwendet, spielt dies keine Rolle. Bei anderen Ähnlichkeitsmaßen muß dies natürlich nicht unbedingt gelten.

Der so berechnete Vektor $\vec{r}$ kann wie folgt zur Klassifikation neuer Dokumente benutzt werden. Wenn die Ähnlichkeit eines neu zu klassifizierenden Dokumentes d mit $\vec{r}$ größer ist als ein Schwellwert θ, wird dieses Dokument als + klassifiziert, sonst als − [Joachims, 1996]:

$$H_{ROCCHIO}(d) = \begin{cases} + & wenn \quad \cos(\vec{r}, \vec{d}) > \theta \\ \\ - & sonst \end{cases} \tag{3.4}$$

Dabei kann die Höhe von $\cos(\vec{r}, \vec{d})$ als Konfidenzwert benutzt werden. Experimente zur Textkategorisierung mit dem Rocchio-Algorithmus in dieser oder ähnlicher Form finden sich z. B. in [Lang, 1995], [Balabanovic und Shoham, 1995], [Joachims, 1996], [Joachims, 1997b], [Joachims et al., 1997] und [Veltmann, 1997].

Auch in den in dieser Arbeit beschriebenen Experimenten wird der Rocchio-Algorithmus in der hier angegebenen Form zur binären Klassifikation eingesetzt. Dabei wird der Schwellwert θ nicht manuell gesetzt, sondern automatisch durch *v-Fold-Cross-Validation* auf den Trainingsdaten bestimmt. Hierzu wird die Trainingsmenge in v gleich große Teilmengen zerlegt und in v Lernläufen des Rocchio-Algorithmus wie folgt vorgegangen. Im v. Lauf wird auf der Vereinigungsmenge aller v Teilmengen außer der v. Teilmenge gelernt. Anhand der v. Teilmenge wird dann ein optimaler Schwellwert für den so gelernten Klassifikator für diese Teilmenge ermittelt, d. h. ein Schwellwert, der auf dieser v. Teilmenge eine optimale Klassifikationsgenauigkeit (Accuracy) erreicht. Gibt es mehrere in diesem Sinne optimale Schwellwerte, wird der niedrigste optimale Schwellwert gewählt. Wenn alle v Lernläufe beendet sind, wird auf der kompletten Trainingsmenge, also allen v Teilmengen, ein neuer Klassikator gelernt, dessen Schwellwert auf den Mittelwert der v Schwellwerte gesetzt wird.

Bei Klassifikationsproblemen mit zwei oder mehr Klassen läßt sich die folgende, beispielsweise in [Joachims, 1997a] eingesetzte Variante des Rocchio-Algorithmus anwenden. Sei $\mathcal{C} = \{C_1, C_2, ...\}$ die Menge aller Klassen und D die Menge aller Trainingsdokumente. Anstelle nun wie oben für die Klasse $+$ nur für eine Klasse einen Vektor zu berechnen, wird nun für jede Klasse ein Repräsentantenvektor $\vec{c_j}$ berechnet:

$$\vec{c_j} = \alpha \frac{1}{|C_j|} \sum_{\vec{d} \in C_j} \frac{\vec{d}}{||\vec{d}||} - \beta \frac{1}{|D - C_j|} \sum_{\vec{d} \in D - C_j} \frac{\vec{d}}{||\vec{d}||} \tag{3.5}$$

Auch hier werden Komponenten von $\vec{c_j}$ mit negativem Wert auf Null gesetzt. Die Menge der gelernten Prototypvektoren stellt das gelernte Klassifikationsmodell dar. Um ein neues Dokument d' zu klassifizieren, wird die Ähnlichkeit seines Dokumentvektors $\vec{d'}$ zu den Prototypvektoren $\vec{c_j}$ anhand des Kosinus-Maßes ermittelt und es wird ihm die Klasse des ähnlichsten Prototypvektors zugewiesen, also die des Vektors, mit dem er den höchsten Kosinus-Wert hat:

$$\begin{aligned} H_{TFIDF}(d') &= \arg\max_{C_j \in \mathcal{C}} \; cos(\vec{c_j}, \vec{d'}) && (3.6) \\ &= \arg\max_{C_j \in \mathcal{C}} \left(\frac{\vec{c_j}}{||\vec{c_j}||} \cdot \frac{\vec{d'}}{||\vec{d'}||} \right) && (3.7) \\ &= \arg\max_{C_j \in \mathcal{C}} \left(\frac{\sum_{i=1}^{|F|} c_j^{(i)} \cdot d'^{(i)}}{\sqrt{\sum_{i=1}^{|F|} (c_j^{(i)})^2}} \right) && (3.8) \end{aligned}$$

Die Funktion $\arg\max_{x \in X}(f(x))$ liefert hierbei den Wert x aus X, für den $f(x)$ maximal wird. F ist die Menge der Attribute, also z. B. der zur Repräsentation verwendeten Wörter. In Gleichungszeile (3.8) wird auf die Normalisierung des Dokumentvektors $\vec{d'}$ verzichtet, weil sie das Ergebnis der Funktion arg max nicht beeinflußt.

3.2 Naiver Bayes'scher Klassifikator

Der in diesem Abschnitt vorgestellte Naive Bayes'sche Klassifikator (siehe [Joachims, 1996], [Joachims, 1997a]) basiert auf einem probabilistischen Modell von Texten. Das Modell macht zwar einige stark vereinfachende Annahmen, aber dies geschieht in der Hoffnung, immer noch ausreichend viele der für die Klassifikationsaufgabe relevanten Charakteristika der Texte zu erfassen. Im folgenden werden wortbasierte Unigram-Modelle von Text benutzt, d. h. es wird angenommen, daß die Wörter in einem Dokument unabhängig von den anderen Wörtern im Text auftreten. Diese sogenannte *konditionale Unabhängigkeitsannahme* trifft für Texte nachweislich nicht zu, ermöglicht es aber, das Problem der Textklassifikation handhabbar zu machen.

Es gibt für jede Klasse $C_j \in \mathcal{C}$ ein solches Unigram-Modell, insgesamt also $|\mathcal{C}|$ solcher Modelle. Es wird angenommen, daß alle Dokumente einer Klasse von dem zu dieser Klasse gehörenden Modell erzeugt werden. Im folgenden wird versucht, die Wahrscheinlichkeit $\Pr(C_j|d)$ zu schätzen, daß sich ein Dokument d in der Klasse C_j befindet. Gemäß der Bayes'schen Regel ist ein Klassifikator dann optimal, wenn er ein Dokument d der Klasse C_j zuweist, für die $\Pr(C_j|d)$ am höchsten ist:

$$H_{BAYES}(d') = \arg\max_{C_j \in \mathcal{C}} \Pr(C_j|d') \tag{3.9}$$

$\Pr(C_j|d')$ wird entsprechend der Länge l der Dokumente aufgespalten in

$$\Pr(C_j|d') = \sum_{l=1}^{\infty} \Pr(C_j|d', l) \cdot \Pr(l|d') \tag{3.10}$$

Dabei ist $\Pr(l|d')$ Eins für die Länge l von Dokument d und Null sonst. Wenn man den Satz von Bayes auf $\Pr(l|d')$ anwendet, erhält man:

$$\Pr(C_j|d') = \frac{\Pr(d'|C_j, l') \cdot \Pr(C_j|l')}{\sum_{C' \in \mathcal{C}} \Pr(d'|C', l') \cdot \Pr(C'|l')} \tag{3.11}$$

$\Pr(d'|C_j, l')$ ist die Wahrscheinlichkeit, daß Dokument d' in Klasse C_j auftritt, wenn seine Länge l' gegeben ist. $\Pr(C_j|l')$ ist die a priori Wahrscheinlichkeit, daß ein Dokument der Länge l' zur Klasse C_j gehört. Nun wird die Annahme getroffen, daß die Klasse eines Dokumentes nicht von seiner Länge abhängt, daß also gilt $\Pr(C_j|l') = \Pr(C_j)$. Damit läßt sich eine Abschätzung $\hat{\Pr}(C_j)$ der Wahrscheinlichkeit $\Pr(C_j)$ aus dem Anteil der Trainingsdokumente berechnen, der zur Klasse C_j gehört:

$$\hat{\Pr}(C_j) = \frac{|C_j|}{\sum_{C' \in \mathcal{C}} |C'|} = \frac{|C_j|}{|D|} \tag{3.12}$$

Dabei ist $|C_j|$ die Anzahl der Trainingsdokumente in Klasse C_j und $|D|$ die Gesamtanzahl von Trainingsdokumenten.

Die Wahrscheinlichkeit $\Pr(d'|C_j, l')$, daß ein Dokument d' in Klasse C_j auftritt, wenn seine Länge l' gegeben ist und nur Dokumente der Klasse C_j berücksichtigt werden, läßt sich mit Hilfe des auf der konditionalen Unabhängigkeitsannahme beruhenden Unigram-Modells abschätzen. Da das Auftreten eines Wortes nur von der Klasse des Dokumentes abhängt, nicht aber vom Auftreten anderer Wörter im Dokument oder von der Dokumentlänge, kann $\Pr(d'|C_j, l')$ wie folgt angenähert werden:

$$\Pr(d'|C_j, l') \approx \prod_{i=1}^{|d'|} \Pr(w_i|C_j) \tag{3.13}$$

Der Index i läuft über die Sequenz der Wörter in Dokument d' und w_i bezeichnet das i-te Wort der Sequenz. $|d'|$ ist die Anzahl der Wörter in Dokument d'. Alle Wörter in d' sind Elemente des Attributmenge F. Die Abschätzung von $\Pr(d'|C_j)$ läßt sich auf die unabhängige Abschätzung der einzelnen Wahrscheinlichkeiten $\Pr(w_i|C_j)$ reduzieren. Ein Bayes'scher Schätzer für $\Pr(w_i|C_j)$, der auch Laplace-Schätzer genannt wird, wird hierzu eingesetzt:

$$\hat{\Pr}(w_i|C_j) = \frac{1 + TF(w_i, C_j)}{|F| + \sum_{w' \in |F|} TF(w', C_j)} \tag{3.14}$$

$TF(w, C_j)$ ist die Gesamtanzahl des Auftretens von Wort w in Dokumenten der Klasse C_j. Dieser Bayes'sche Schätzer benutzt die Annahme, daß die Beobachtung aller Wörter a priori gleich wahrscheinlich ist, und vermeidet so, Wahrscheinlichkeiten fälschlicherweise als Null zu schätzen.

Aus den Gleichungen (3.9), (3.11) und (3.13) resultiert die Klassifikationsregel für den Naiven Bayes'schen Klassifkator:

$$\begin{aligned}
H_{BAYES}(d') &= \arg\max_{C_j \in \mathcal{C}} \frac{\Pr(C_j) \cdot \prod\limits_{i=1}^{|d'|} \Pr(w_i|C_j)}{\sum\limits_{C' \in \mathcal{C}} \Pr(C') \cdot \prod\limits_{i=1}^{|d'|} \Pr(w_i|C')} && (3.15) \\
&= \operatorname{argmax}_{C_j \in \mathcal{C}} \frac{\Pr(C_j) \cdot \prod\limits_{w \in F} \Pr(w|C_j)^{TF(w,d')}}{\sum\limits_{C' \in \mathcal{C}} \Pr(C') \cdot \prod\limits_{w \in F} \Pr(w|C')^{TF(w,d')}} && (3.16)
\end{aligned}$$

Falls die Wahrscheinlichkeit $\Pr(C_j|d')$ nicht als Konfidenzmaß für die getroffene Entscheidung benötigt wird, kann auf die Berechnung des Nenners verzichtet werden, da diese das Ergebnis von arg max nicht beeinflußt. Anwendungen mit Naiven Bayes'schen Klassifikatoren in der Textkategorisierung finden sich z. B. in [Lewis, 1992] [Lang, 1995], [Joachims, 1996], [Joachims, 1997a].

3.3 Der PrTFIDF-Algorithmus

Der *PrTFIDF*-Algorithmus ist ein vom TFIDF-Klassifikator abgeleiteter probabilistischer Klassifikator [Joachims, 1997a]. Dieser Algorithmus ist äquivalent zum TFIDF-Klassifikator für mehrere Klassen (siehe Gleichung 3.7 in Abschnitt 3.1), wenn man eine modifizierte Version der Inverse Document Frequency (IDF) zur Wortgewichtung heranzieht, die Dokumentlänge mit einer Division durch die Anzahl der Wörter im Dokument normiert und als Ähnlichkeitsmaß zweier Dokumente das Skalarprodukt ihrer Dokumentvektoren verwendet. Die folgende Darstellung orientiert sich an [Joachims, 1997a].

Der im vorherigen Abschnitt vorgestellte Naive Bayes'sche Klassifikator schätzt die Wahrscheinlichkeit $\Pr(C_j|d')$, daß ein Dokument d' zur Klasse C_j gehört, basierend auf der vereinfachenden Annahme der Wortunabhängigkeit. Der PrTFIDF-Algorithmus verwendet eine andere Abschätzung für diese Wahrscheinlichkeit. Diese Abschätzung wurde inspiriert vom *Retrieval with Probabilistic Indexing (RPI)* Ansatz [Fuhr, 1989], bei dem eine Menge von Deskriptoren X benutzt wird, um den Inhalt von Dokumenten zu repräsentieren. Dabei wird ein Deskriptor x einem Dokument d mit einer bestimmten Wahrscheinlichkeit $\Pr(x|d)$ zugeordnet. Mit dem Satz von der Gesamtwahrscheinlichkeit (in Zeile (3.17)) und dem Satz von Bayes (in Zeile (3.18)) läßt sich $\Pr(C_j|d')$ nun wie folgt ausdrücken:

$$\Pr(C_j|d) = \sum_{x \in X} \Pr(C_j|x,d) \cdot \Pr(x|d) \tag{3.17}$$

$$= \sum_{x \in X} \frac{\Pr(d|C_j,x)}{\Pr(d|x)} \Pr(C_j|x) \cdot \Pr(x|d) \tag{3.18}$$

Hier wird nun die vereinfachende Annahme gemacht, daß $\Pr(d|C_j,x) = \Pr(d|x)$, womit dann folgende Näherung gilt:

$$\Pr(C_j|d) \approx \sum_{x \in X} \Pr(C_j|x) \cdot \Pr(x|d) \tag{3.19}$$

Diese Annahme besagt, daß der Deskriptor x bereits soviel Information über d enthält, daß keine weitere Information über d gewonnen wird, wenn man weiß, daß d zur Klasse C_j gehört. Die Gültigkeit dieser Annahme hängt natürlich vom Klassifikationsproblem und der Wahl der Deskriptorenmenge X ab. Eine potentielle Wahl für die Menge der Deskriptoren im Sinne der Bag-of-Words-Repräsentation der Dokumente ist es, jede Multimenge von n Wörtern aus der Attributmenge F als Deskriptor heranzuziehen. Mit dem Parameter Wortanzahl n wird dann die Qualität der Annäherung im Gegensatz zur Komplexität der Abschätzung reguliert. Wenn n gleich der Anzahl der Wörter in Dokument d ist, ist die Wahrscheinlichkeit $\Pr(C_j|d)$ gleich der Wahrscheinlichkeit $\Pr(C_j|x)$, aber mit abnehmendem n nimmt natürlich auch die Gültigkeit dieser Abschätzung ab.

Für den PrTFIDF-Klassifikator wird nun der einfachste Fall solcher Deskriptoren gewählt, nämlich der mit $n = 1$, d. h. die Deskriptoren sind gleich den

Wörtern. Damit läßt sich die Abschätzung in Zeile (3.19) wie folgt umschreiben:

$$\Pr(C_j|d) \quad \approx \quad \sum_{w \in F} \Pr(C_j|w) \cdot \Pr(w|d) \tag{3.20}$$

Jetzt müssen noch die beiden Wahrscheinlichkeiten $\Pr(C_j|w)$ und $\Pr(w|d)$ abgeschätzt werden. $\Pr(w|d)$ kann mit Hilfe der Dokumentrepräsentation von d abgeschätzt werden:

$$\hat{\Pr}(w|d) = \frac{TF(w,d)}{\sum_{w' \in F} TF(w',d)} = \frac{TF(w,d)}{|d|} \tag{3.21}$$

Hierbei ist $|d|$ die Anzahl der Wörter in Dokument d. Die andere Wahrscheinlichkeit, $\Pr(C_j|w)$, also die Wahrscheinlichkeit, daß Dokument d in Klasse C_j ist, wenn man nur ein zufällig aus d gezogenes Wort w kennt, läßt sich mit dem Satz von Bayes umschreiben:

$$\Pr(C_j|w) = \frac{\Pr(w|C_j) \cdot \Pr(C_j)}{\sum_{C' \in \mathcal{C}} \Pr(w|C') \cdot \Pr(C')} \tag{3.22}$$

$\Pr(C_j)$ kann man über den Anteil der Trainingsdokumente abschätzen, die zur Klasse C_j gehören:

$$\hat{\Pr}(C_j) = \frac{|C_j|}{\sum_{C' \in \mathcal{C}} |C'|} = \frac{|C_j|}{|D|} \tag{3.23}$$

$\Pr(w|C_j)$ läßt sich wie folgt abschätzen:

$$\hat{\Pr}(w|C_j) = \frac{1}{|C_j|} \sum_{d \in C_j} \hat{\Pr}(w|d) \tag{3.24}$$

Wenn man die Abschätzungen (3.21) bis (3.24) sukzessive in Zeile (3.20) einsetzt, erhält man die Entscheidungsregel von PrTFIDF:

$$H_{PrTFIDF}(d') \quad = \quad \arg\max_{C_j \in \mathcal{C}} \sum_{w \in F} \frac{\Pr(w|C_j) \cdot \Pr(C_j)}{\sum\limits_{C' \in \mathcal{C}} \Pr(w|C') \cdot \Pr(C')} \cdot \Pr(w|d') \tag{3.25}$$

In [Joachims, 1997a] wird nun eine modifizierte Version der Inverse Document Frequency $IDF(w)$ definiert. Die ursprüngliche Definition von $IDF(w)$ wurde in Abschnitt 2.7 in Gleichung (2.2) auf Seite 16 vorgestellt:

$$IDF(w) \quad = \quad log\left(\frac{|D|}{DF(w)}\right) \tag{3.26}$$

$$DF(w) \quad = \quad \sum_{d \in D} \begin{cases} 1 & \textit{Dokument } d \textit{ enthält Wort } w \\ 0 & \textit{sonst} \end{cases} \tag{3.27}$$

Die modifizierte Version, die sich aus dem PrTFIDF-Algorithmus ergibt, ist wie folgt definiert:

$$IDF'(w) \quad = \quad sqrt\left(\frac{|D|}{DF'(w)}\right) \tag{3.28}$$

$$DF'(w) \quad = \quad \sum_{d \in D} \frac{TF(w,d)}{|d|} \tag{3.29}$$

Es gibt offensichtlich zwei Unterschiede in diesen beiden Definitionen der Inverse Document Frequency. Erstens ist $DF'(w)$ nicht die Anzahl der Dokumente, in denen Wort w vorkommt, sondern die Summe der relativen Häufigkeiten von w in den Dokumenten der Kollektion. Damit nutzt $IDF'(w)$ nicht nur die binäre Vorkommensinformation, sondern berücksichtigt stattdessen die genauere Häufigkeitsinformation. Der zweite Unterschied ist die Benutzung der Wurzelfunktion anstelle des Logarithmus, um den Einfluß der Document Frequency auf den Wert der Inverse Document Frequency zu dämpfen. Allerdings haben beide Funktionen eine ähnliche Form und reduzieren den Einfluß großer Document Frequency Werte.

In [Joachims, 1997a] wird gezeigt, daß die oben angegebene PrTFIDF-Entscheidungsregel (Zeile (3.25)) äquivalent zur Entscheidungsregel des TFIDF-Klassifikators ist (siehe Gleichung (3.7) auf Seite 24 in Abschnitt 3.1), wenn man:

- $IDF'(w)$ anstelle von $IDF(w)$ zur Wortgewichtung in den Dokumentvektoren einsetzt,
- die Dokumente zur Normierung nicht durch ihre euklidische Länge, sondern durch die Anzahl der in ihnen jeweils vorkommenden Wörter ($|d|$) dividiert,
- das Skalarprodukt als Ähnlichkeitsmaß für zwei Dokumentvektoren verwendet und
- die Parameter $\alpha_j = \frac{|C_j|}{|D|}$ und $\beta = 0$ setzt.

Somit bietet der PrTFIDF-Algorithmus eine Möglichkeit, die TFIDF-Parameter α und β automatisch zu setzen und dabei die anhand der Trainingsdokumente geschätzten a priori Wahrscheinlichkeiten $\Pr(C_j)$ der Klassen C_j zu berücksichtigen. Eine Anwendung dieses Verfahrens in der Textklassifikation ist in [Joachims, 1997a] beschrieben.

3.4 k-Nächste Nachbarn-Verfahren

Bei instanz-basierten Lernverfahren wie den *k-Nächste Nachbarn-Verfahren (k-NN)* (siehe beispielsweise [Joachims, 1996], [Mitchell, 1997]) besteht der Lernschritt darin, einfach alle Trainingsbeispiele zu speichern. Es erfolgt also keine Berechnung von Klassenrepräsentanten wie beispielsweise beim Rocchio-Algorithmus. Eine Generalisierung von den Trainingsbeispielen erfolgt erst, wenn ein neues Beispiel klassifiziert werden soll. Für die Klassifikation eines neuen Beispiels werden die k nächsten Trainingsbeispiele zu diesem neuen Beispiel anhand eines Ähnlichkeitsmaßes (oder Abstandsmaßes) bestimmt und die unter diesen Trainingsbeispielen am häufigsten vertretene Klasse wird für das neue Beispiele vorhergesagt. Ist C_j die Klasse mit der größten Anzahl k_{C_j} von Beispielen unter den k nächsten Nachbarn, so wird das neue Dokument mit der

Konfidenz k_{C_j}/k als der Klasse C_j zugehörig klassifiziert. Die Entscheidungsregel für den k-Nächste Nachbarn-Klassifikator lautet also:

$$H_{kNN}(d) = \arg\max_{C_j \in \mathcal{C}} \; k_{C_j} \tag{3.30}$$

Wie die Entscheidungsregeln für den Naiven Bayes'schen Klassifikator und den PrTFIDF-Klassifikator in den vorherigen Abschnitten läßt sich auch diese Entscheidungsregel aus der Bayes'schen Regel herleiten. Eine solche Herleitung für den euklidischen Abstand als Abstandsmaß ist in [Michie et al., 1994] und [Joachims, 1996] beschrieben. k-NN-Klassifikatoren sind relativ unempfindlich gegen Rauschen in den Daten, da verrauschte Beispiele durch nicht-verrauschte Nachbarn überstimmt werden. Bei gleichmäßiger Gewichtung aller Attribute im euklidischen Raum sind sie allerdings empfindlich gegen eine große Anzahl irrelevanter Attribute, da sie zwischen verschiedenen Attributen nicht differenzieren (siehe hierzu [Mitchell, 1997]).

Das Abstandsmaß, das benutzt wird, um ähnliche Beispiele mit bekannter Klassifikation zu finden, sollte so beschaffen sein, daß Dokumente, die anhand dieses Maßes ähnlich sind, möglichst zur selben Klasse gehören. Für Textkategorisierungsprobleme werden in der Regel andere Maße als die euklidsche Distanz gewählt. Wenn Dokumente in der Vektorraum-Repräsentation mit TFIDF-Gewichtung dargestellt werden, haben Dokumente mit unterschiedlicher Länge selbst dann einen großen euklidschen Abstand, wenn sie die gleiche Verteilung von Wörtern enthalten. Dieses Problem läßt sich durch eine Normierung der Vektoren auf die euklidische Länge Eins beheben. Meistens wird im Information Retrieval und bei der Textkategorisierung jedoch der Winkel bzw. der umgekehrt monoton mit dem Winkel verlaufende Kosinus zwischen Dokumentvektoren als Abstands- bzw. Ähnlichkeitsmaß gewählt. Der Kosinus ist unabhängig von der Dokumentlänge. Bei Verwendung des Kosinus als Ähnlichkeitsmaß ergibt sich die gleiche Entscheidungsregel wie für den euklidischen Abstand als Distanzmaß. Eine entsprechende Herleitung aus der Bayes'schen Regel ist in [Joachims, 1996] beschrieben.

Bei *distanz-gewichteten k-NN-Verfahren* wird der Abstand der k nächsten Nachbarn des zu klassifizierenden Dokumentes d bei der Klassifikation berücksichtigt. Je näher ein solcher Nachbar d_i dem Dokument d ist, desto höher ist sein Gewicht w_i bei der Entscheidung:

$$H_{kNN'}(d) = \arg\max_{C_j \in \mathcal{C}} \sum_{d_i \in kNN(d) \,\wedge\, T(d_i)=C_j} w_i \tag{3.31}$$

Hierbei ist $kNN(d)$ die Menge der k nächsten Nachbarn von Dokument d'. Für den euklidischen Abstand berechnet sich das Gewicht des Nachbarn d_i von d wie folgt:

$$w_i = \frac{1}{||\vec{d} - \vec{d_i}||} \tag{3.32}$$

Bei der Verwendung des Kosinus als Ähnlichkeitsmaß ergibt sich folgendes Gewicht:

$$w_i = \cos(\vec{d}, \vec{d_i}) \tag{3.33}$$

Soll die Summe der Gewichte der zu einer Klasse gehörenden nächsten Nachbarn im rechten Teil der Gleichung (3.31) als Konfidenz in die getroffene Entscheidung verwendet werden, wie es zum Beispiel für die Erstellung einer Rangordnung von Dokumenten gemäß ihrer angenommenen Relevanz notwendig ist, so ist das jeweilige Gewicht w_i (bzw. alternativ die Summe in Gleichung (3.31)) mit einer Division durch die Summe der Gewichte aller k nächsten Nachbarn (ohne Rücksicht auf ihre Klasse) zu normieren. Für den Kosinus als Ähnlichkeitsmaß berechnen sich die Gewichte dann wie folgt:

$$w_i = \frac{\cos(\vec{d}, \vec{d_i})}{\sum_{d_k \in kNN(d)} \cos(\vec{d}, \vec{d_k})} \tag{3.34}$$

Dem schnellen und einfachen Lernschritt bei k-NN-Klassifikatoren steht bei großen Trainingsmengen und/oder langen Attributvektoren der große Speicherplatzbedarf zur Speicherung aller Trainingsbeispiele und die lange Laufzeit für die Klassifikation neuer Beispiele entgegen, die zur Klassifikation mit allen Trainingsbeispielen verglichen werden müssen. Dennoch werden k-NN-Klassifikatoren häufig zur Textkategorisierung eingesetzt, da sich mit ihnen eine relativ gute Performanz erzielen läßt (siehe z.B. [Joachims, 1996], [Joachims, 1997a], [Yang, 1997]).

3.5 Der Winnow-Algorithmus

Der Winnow-Algorithmus ([Littlestone, 1988], [Littlestone, 1991]) lernt eine lineare Schwellwertfunktion, die auch zur Klassifikation von Texten benutzt werden kann. Dieser Abschnitt beschreibt eine iterative Version von Winnow [Joachims, 1996], die auch in den Experimenten zu dieser Arbeit zum Einsatz kommt. Es kann gezeigt werden, daß dieser in der algorithmischen Lerntheorie entwickelte Algorithmus bei angemessener Parameterwahl in der Lage ist, jedes linear separierbare Problem effizient zu lernen [Littlestone, 1988]. Auch wenn Textklassifikationsprobleme im allgemeinen nicht linear separierbar sind, stärkt empirische Evidenz die Vermutung, daß Winnow sich auch in dieser Domäne einsetzen läßt (siehe z.B. [Blum, 1997], [Joachims, 1996]). Eine für die Textkategorisierung attraktive Eigenschaft von Winnow ist, daß die obere Schranke für die Anzahl der Fehler, die dieses Verfahren macht, nur logarithmisch mit der Anzahl der irrelevanten Attribute wächst [Littlestone, 1988].

Winnow ist nur auf binäre Klassifikationsprobleme anwendbar. Seien die Klassen o. B. d. A. $+$ und $-$. Winnows Hypothesensprache ist die Menge der linearen Schwellwertfunktionen. Sei $OCC(w, d)$ gleich 1, wenn das Wort w in

Dokument d mindestens einmal vorkommt, und 0 sonst. Für einen gegebenen Satz von Gewichten $weight(w)$ und einen gegebenen Schwellwert θ, klassifiziert Winnow ein Dokument d gemäß folgender Entscheidungsregel:

$$H_{WINNOW}(d) = \begin{cases} + & wenn \quad \sum_{w \in F} weight(w) \cdot OCC(w,d) > \theta \\ \\ - & sonst \end{cases} \tag{3.35}$$

F ist die Menge der Attribute. Die Summe $\sum_{w \in F} weight(w) \cdot OCC(w,d)$ wird berechnet und mit dem Schwellwert θ verglichen. Wenn die Summe größer als θ ist, wird das Dokument als zu $+$ gehörig klassifiziert, sonst als zu $-$ gehörig. Die Höhe der Summe kann als Konfidenz für die Zugehörigkeit von Dokument d zur Klasse $+$ verwendet werden.

In der Lernphase durchsucht Winnow den Hypothesenraum nach einer Hyperebene, die positive und negative Beispiele trennt und die Anzahl der falsch klassifizierten Beispiele minimiert. Initial wird das Gewicht $weight(w)$ jedes Wortes $w \in F$ auf den Wert 1 gesetzt. Bei gegebener Lernrate γ werden die Gewichte dann inkrementell für jedes falsch klassifizierte Beispiel in der im folgenden beschriebenen Weise multiplikativ verändert.

Durchlaufe alle Dokumente der Trainingsmenge m-mal und wende die folgenden Regeln für jedes Dokument d an:

- Winnow klassifiziert Dokument d als $+$, aber die korrekte Klasse ist $-$:
  Die Gewichte $weight(w)$ aller Wörter w mit $OCC(w,d) = 1$ werden durch die Lernrate γ geteilt.
- Winnow klassifiziert Dokument d als $-$, aber die korrekte Klasse ist $+$:
  Die Gewichte $weight(w)$ aller Wörter w mit $OCC(w,d) = 1$ werden mit der Lernrate γ multipliziert.
- Winnows Klassifikation ist korrekt:
  Die Gewichte bleiben unverändert.

Während die hier beschriebene Version von Winnow in der Lernphase m-mal eine Trainingsmenge durchläuft und anschließend in der Anwendungsphase nicht weiter lernt, ist der ursprünglich von Littlestone entworfene Winnow-Algorithmus (in [Littlestone, 1988] als Winnow2 bezeichnet) eigentlich ein Online-Lernverfahren. Beim Online-Lernen gibt es die Trennung von Lern- und Anwendungsphase nicht, d. h. jedes neue Beispiel wird direkt auch zum Lernen verwendet, sobald seine echte Klasse bekannt ist. Die Online-Version von Winnow aktualisiert ihre Gewichte ebenfalls nach den oben angegebenen Regeln. Die in [Littlestone, 1988] und [Littlestone, 1991] beschriebenen theoretischen Resultate für die oberen Fehlerschranken von Winnow beziehen sich auf das Online-Lernen statischer Konzepte. Zu diesen Resultaten gehört, daß die Anzahl der Klassifikationsfehler von Winnow bei linear separierbaren Problemen höchstens um einen konstanten Faktor vom Optimum entfernt ist. Außerdem erreicht Winnow durch seine multiplikative Gewichtsaktualisierung, wie bereits oben erwähnt, daß die Obergrenze für die Anzahl an Klassifikationsfehlern nur logarithmisch mit der Anzahl irrelevanter Attribute wächst, während diese Obergrenze beispielsweise beim Perceptron-Algorithmus [Rosenblatt, 1962] linear mit der Anzahl irrelevanter Attribute steigt. Der Perceptron-Algorithmus ist auch ein Lernverfahren für lineare Schwellwertfunktionen, aktualisiert seine Gewichte jedoch additiv.

Winnow wurde zwar als Online-Lerner statischer Konzepte entworfen, wurde aber auch schon in einer Domäne mit Konzeptverschiebungen erfolgreich eingesetzt (siehe [Blum, 1997]). Allerdings hatten die Konzeptverschiebungen in dieser Domäne nicht mit einem Wechsel der relevanten Attribute zu tun, wie dies in der Textklassifikation zu erwarten ist, sondern lediglich mit den Werten dieser Attribute und ihrer Bedeutung für das zu lernende Konzept. Die dort erzielten Ergebnisse sind also nicht unbedingt auf das Lernen sich verändernder Konzepte bei der Textklassifikation übertragbar. Ungeachtet eines möglichen Einsatzes als Online-Lernverfahren wird Winnow in den in dieser Arbeit beschriebenen Experimenten in der in [Joachims, 1996] beschriebenen Version eingesetzt, die auf einer Trainingsmenge lernt und in der Anwendungsphase ihre Gewichte unverändert läßt, um die Vergleichbarkeit mit den anderen eingesetzten Lernverfahren zu gewährleisten, die ebenfalls zwischen Lern- und Anwendungsphase unterscheiden. Außerdem ließe sich bei einem Einsatz der Online-Version schwer feststellen, inwieweit die erreichte Adaptivität an die Konzeptverschiebungen auf die in dieser Arbeit untersuchte Zeitfensterverwaltung zurückzuführen und inwieweit sie den Online-Lern-Eigenschaften von Winnow zuzuschreiben wäre, d. h. der in den Experimenten zu untersuchende Effekt der Fensterverwaltung ließe sich nicht mehr eindeutig feststellen.

3.6 Support-Vektor-Maschinen

Die folgende Darstellung der *Support-Vektor-Maschinen (SVMs)* orientiert sich an [Joachims, 1997b]. Die in der algorithmischen Lerntheorie entwickelten Support-Vektor-Maschinen basieren auf dem *Structural Risk Minimization* Prinzip [Vapnik, 1995] und lernen binäre Klassifikatoren, o. B. d. A. für die Klassen + und −. Die Idee der Minimierung des strukturellen Risikos ist es, eine Hypothese h zu finden, für die man den niedrigsten echten Fehler garantieren kann. Der echte Fehler einer Hypothese h ist die Wahrscheinlichkeit, daß h ein zufällig ausgewähltes unbekanntes Testbeispiel falsch klassifiziert. Die folgende obere Schranke verbindet den echten Fehler einer Hypothese h mit ihrem Fehler auf der Trainingsmenge und ihrer Komplexität [Vapnik, 1995]:

$$Fehler_{echt}(h) \leq Fehler_{training}(h) + 2\sqrt{\frac{d(ln\frac{2n}{d}+1) - ln\frac{\eta}{4}}{n}} \tag{3.36}$$

Diese obere Schranke gilt mit einer Wahrscheinlichkeit von mindestens $1 - \eta$. n ist die Anzahl der Trainingsbeispiele und d ist die *VC-Dimension (VCdim)*

[Vapnik, 1995] des Hypothesenraums. Die VC-Dimension eines Hypothesenraums ist ein Maß für die Ausdrucksfähigkeit dieses Hypothesenraums. Die VC-Dimension eines Hypothesenraums H ist definiert als die maximale Anzahl m von Beispielen, die mit Funktionen (Hypothesen) aus H auf alle 2^m Möglichkeiten in zwei Mengen aufgeteilt werden können. Lineare Schwellwertfunktionen mit n Attributen haben beispielsweise eine VC-Dimension von $n+1$.

Ungleichung (3.36) spiegelt den bekannten Trade-Off zwischen der Komplexität des Hypothesenraums und dem Trainingsfehler wieder. Einerseits führt ein sehr einfacher Hypothesenraum (kleine VCdim) wahrscheinlich zu einer schlechten Annäherung an die Zielfunktion und somit zu einem hohen Fehler auf der Trainingsmenge und einem hohen echten Fehler. Ein zu ausdrucksstarker Hypothesenraum (hohe VCdim) andererseits führt zwar wahrscheinlich zu einem kleinen Fehler auf den Trainingsdaten, aber der zweite Term auf der rechten Seite von Ungleichung 3.36 wird sehr groß. Dieses Problem wird mit Überspezialisierung bezeichnet (*Overfitting*). Es ist also entscheidend, einen Hypothesenraum mit der „richtigen“ Komplexität für ein gegebenes Klassifikationsproblem auszusuchen.

Für die Minimierung des strukturellen Risikos wird eine Struktur über den Hypothesenräumen H_i definiert, bei der die VC-Dimensionen d_i der Hypothesenräume wachsen:

$$H_1 \subset H_2 \subset ... \subset H_i \subset ... \qquad \text{und} \qquad \forall i : d_i \leq d_{i+1} \tag{3.37}$$

Das Ziel ist es, den Index i^* zu finden, für den die in Ungleichung (3.36) angegebene obere Schranke für den echten Fehler minimal wird. Im folgenden werden lineare Schwellwertfunktionen der folgenden Form als Hypothesen betrachtet:

$$h(\vec{d}) = sign\{\vec{w} \cdot \vec{d} + b\} = \begin{cases} +1, & \text{falls } \vec{w} \cdot \vec{d} + b > 0 \\ -1, & \text{sonst} \end{cases} \tag{3.38}$$

Dabei bezeichnet $\vec{w}$ den Gewichtsvektor der Schwellwertfunktion und $\vec{d}$ den Vektor eines Beispiels. Die Struktur von Hypothesenräumen mit ansteigender VCdim wird nun nicht über die Anzahl der Attribute gebildet, mit der die VCdim wie oben erwähnt linear steigt, sondern über eine verfeinerte Struktur, die in folgendem Lemma zum Ausdruck kommt.

Lemma 1. [Vapnik, 1982] *Seien die Hypothesen Hyperebenen der Form* $h(\vec{d}) = sign\{\vec{w} \cdot \vec{d} + b\}$. *Wenn alle Beispielvektoren* $\vec{d_i}$ *in einer Kugel vom Radius R enthalten sind und alle Beispiele* $\vec{d_i}$ *die Bedingung*

$$|\vec{w} \cdot \vec{d_i} + b| \geq 1, \quad mit\ ||\vec{w}|| = A \tag{3.39}$$

erfüllen, dann hat die Menge dieser Hyperebenen eine VCdim d, die durch folgende obere Schranke begrenzt ist

$$d \leq min([R^2 A^2], n) + 1 \tag{3.40}$$

Wie man sieht, hängt die VCdim dieser Hyperebenen nicht notwendigerweise von der Anzahl der Attribute ab, sondern von der euklidischen Länge $||\vec{w}||$ des Gewichtvektors $\vec{w}$. Man kann also auch in hochdimensionalen Räumen lernen, wie sie in der Textkategorisierung üblich sind, wenn die Hypothese einen kleinen Gewichtsvektor hat.

In ihrer Grundform finden Support-Vektor-Maschinen die Hyperebene h, die die beiden Klassen in den Trainingsdaten mit dem kürzesten Gewichtsvektor trennt. Diese Hyperebene h separiert positive und negative Beispiele mit maximalem Abstand zu den nächsten Beispielen dieser Klassen. Die Beispiele mit dem geringsten Abstand zu dieser Hyperebene heißen *Support-Vektoren*. Das Finden dieser Hyperebene kann in folgendes Optimierungsproblem überführt werden:

$$\text{Minimiere:} \quad ||\vec{w}|| \tag{3.41}$$

$$\text{so daß:} \quad \forall i : y_i[\vec{w} \cdot \vec{d}_i + b] \geq 1 \tag{3.42}$$

Dabei hat y_i den Wert $+1$, wenn Dokument d_i zu Klasse $+$ gehört, und den Wert -1, wenn d_i zur Klasse $-$ gehört. Die Nebenbedingungen (3.42) fordern die korrekte Klassifikation aller Trainingsbeispiele. Das oben angegebene Lemma ermöglicht Rückschlüsse über die VCdim des Strukturelements, von dem die gefundene separierende Hyperebene stammt. Da das angegebene Optimierungsproblem numerisch schwer zu handhaben ist, werden Lagrange-Multiplikatoren benutzt, um das Problem in ein äquivalentes quadratisches Optimierungsproblem zu transformieren [Vapnik, 1995]:

$$\text{Minimiere:} \quad -\sum_{i=1}^{n} \alpha_i + \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{d}_i \cdot \vec{d}_j \tag{3.43}$$

$$\text{so daß:} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{und} \quad \forall i : \alpha_i \geq 0 \tag{3.44}$$

Zur Lösung dieser Art von Optimierungsproblemen gibt es effziente Algorithmen, die garantiert das Optimum finden. Das Ergebnis der Optimierung sind die Koeffizienten α_i^*, für die (3.43) minimal wird. Aus diesen Koeffizienten läßt sich die (3.41) und (3.42) erfüllende Hyperebene konstruieren:

$$\vec{w} \cdot \vec{d} = \left(\sum_{i=1}^{n} \alpha_i^* y_i \vec{d}_i \right) \cdot \vec{d} = \sum_{i=1}^{n} \alpha_i^* y_i (\vec{d}_i \cdot \vec{d}) \quad \text{und} \quad b = \frac{1}{2}(\vec{w} \cdot \vec{d}_+ + \vec{w} \cdot \vec{d}_-) \tag{3.45}$$

Wie in Gleichung (3.45) zu sehen, wird der Gewichtsvektor der resultierenden Hyperebene als Linearkombination aus den Trainingsbeispielen berechnet, wobei nur die Beispiele $\vec{d}_i$ einen Beitrag leisten, deren Koeffizient α_i^* größer als Null ist. Die Vektoren dieser Beispiele werden *Support-Vektoren* genannt. Sie sind die Beispiele mit minimalem Abstand zur separierenden Hyperebene. Um b zu berechnen, können zwei beliebige Support-Vektoren $\vec{d}_+$ (aus Klasse $+$) und $\vec{d}_-$ (aus Klasse $-$) benutzt werden.

Support-Vektor-Maschinen lassen sich auch für nicht-lineare Hypothesenräume einsetzen. Hierzu werden die Beispiele mit einer nicht-linearen Abbildung in einen (möglicherweise höherdimensionalen) neuen Attributraum überführt, in dem in der bereits beschriebenen Weise eine separierende Hyperebene gefunden wird. Zum Lernen nicht-linearer Hypothesen setzen SVMs Kernelfunktionen $K(\vec{d_1}, \vec{d_2})$ ein. Abhängig von der Art der Kernelfunktion lernen SVMs dann polynomielle Klassifikatoren, radiale Basisfunktionen (Radial Basis Functions (RBF)) oder zwei-lagige neuronale Netz mit sigmoider Aktivierungsfunktion:

$$K_{polynom}(\vec{d_1}, \vec{d_2}) = (\vec{d_1} \cdot \vec{d_2} + 1)^d \tag{3.46}$$

$$K_{rbf}(\vec{d_1}, \vec{d_2}) = exp(\gamma(\vec{d_1} - \vec{d_2})^2) \tag{3.47}$$

$$K_{sigmoid}(\vec{d_1}, \vec{d_2}) = tanh(s(\vec{d_1} \cdot \vec{d_2}) + c) \tag{3.48}$$

Diese Kernelfunktionen erfüllen Mercer's Theorem (siehe [Vapnik, 1995]), d. h. sie berechnen das Skalarprodukt der Vektoren $\vec{d_1}$ und $\vec{d_2}$, nachdem sie sie mit einer nicht-linearen Abbildung Φ in einen neuen Attributraum abgebildet haben:

$$\Phi(\vec{d_1}) \cdot \Phi(\vec{d_2}) = K(\vec{d_1}, \vec{d_2}) \tag{3.49}$$

Um eine solche Kernelfunktion zu verwenden, muß man lediglich jedes Auftreten des Skalarproduktes zweier Beispielvektoren in den Gleichungen (3.43) und (3.45) durch die gewünschte Kernelfunktion ersetzen.

Durch die Verwendung von Kernelfunktionen werden Parameter eingeführt wie beispielsweise der Polynomgrad d beim polynomiellen Kernel oder die Varianz γ beim Kernel für radiale Basisfunktionen. Die Parameter können mit Hilfe folgender Heuristik [Vapnik, 1995], die von der oberen Schranke in (3.36) inspiriert wurde, automatisch auf günstige Werte gesetzt werden. Zuerst lernt die SVM Klassifikationsmodelle für verschiedene Werte von d bzw. γ. Dann wird die VCdim der erhaltenen Hypothesen anhand von Ungleichung (3.40) abgeschätzt und die Hypothese mit der niedrigsten VCdim ausgewählt. Dabei läßt sich die Länge des Gewichtsvektors anhand folgender Formel berechnen:

$$||w||^2 = \sum_{i,j \in SupportVektoren} \alpha_i \alpha_j y_i y_j K(\vec{d_1}, \vec{d_j}) \tag{3.50}$$

Wenn alle Dokumente auf die euklidische Länge Eins normiert sind, läßt sich leicht zeigen, daß der Radius R der Kugel, die alle Trainingsbeispiele einschließt, in folgender Weise nach oben beschränkt ist:

$$K_{polynom}: \quad R^2 \leq 2^d - 1 \qquad\qquad K_{rbf}: \quad R^2 \leq 2(1 - exp(-\gamma)) \tag{3.51}$$

Die beschriebene Prozedur zur Auswahl geeigneter Parameter läßt sich vollautomatisch durchführen, ohne die Testbeispiele heranzuziehen oder eine aufwendige Cross-Validation-Technik anzuwenden.

Bis zu diesem Punkt lag der Beschreibung der SVMs die Annahme zugrunde, daß sich die Trainingsdaten ohne Fehler separieren lassen. Um auch *nicht separierbare Klassen* lernen zu können, kann man in den Nebenbedingungen (3.42) als *Slack Variables* bezeichnete Korrekturwerte zulassen [Cortes und Vapnik, 1995], was dann entsprechende Anpassungen an (3.41), (3.43) und (3.44) nach sich zieht. Die in den Experimenten zu dieser Arbeit verwendete Version der SVM setzt jedoch einen einfacheren Ansatz ein (siehe [Joachims, 1997b]), bei dem während der Optimierung von (3.43) die Werte der Koeffizienten α_i beobachtet werden. Trainingsbeispiele mit sehr hohen α_i-Werten tragen sehr zur Schwierigkeit der Trennbarkeit der Klassen bei. Wenn der Wert eines α_i einen bestimmten Schwellwert überschreitet (in den Experimenten zu dieser Arbeit $\alpha_i \geq 1000$), wird das entsprechende Beispiel aus der Trainingsmenge entfernt und die SVM mit der verbleibenden Trainingsmenge trainiert.

Support-Vektor-Maschinen mit polynomiellem und RBF-Kernel sind bereits sehr erfolgreich zur Lösung von Textklassifikationsproblemen eingesetzt worden (siehe z.B. [Joachims, 1997b], [Graf, 1997]).

3.7 Das Regellernverfahren CN2

Das Verfahren CN2 ([Clark und Niblett, 1989], [Clark und Boswell, 1991]) kann wahlweise zur Erzeugung geordneter und ungeordneter Regelmengen eingesetzt werden. Sowohl die geordnete als auch die ungeordnete Form der induzierten Regelmengen enthalten eine Default-Regel, die dann angewandt wird, wenn ein zu klassifizierendes Beispiel von keiner anderen Regel abgedeckt wird. Die Default-Regel sagt stets für alle zu klassifizierenden Beispiele die in der Trainingsmenge am häufigsten vertretene Klasse vorher. In diesem Abschnitt wird nur die CN2-Version für ungeordnete Regeln beschrieben, die auch in den Experimenten zu dieser Arbeit zum Einsatz kommt. Die Version für geordnete Regeln funktioniert aber sehr ähnlich.

Für die Regeldarstellung benutzt CN2 eine Notation in vielwertiger Propositionallogik. Der Grundbaustein für den Bedingungsteil von CN2-Regeln sind die sogenannten *Selektoren*. Ein *Selektor* ist ein Attribut-Wert-Test, der ein Attribut mit einem Wert (oder einer Disjunktion von Werten) durch einen Vergleichsoperator wie beispielsweise „=“, „≤“oder „>“ in Beziehung setzt, und dessen Gültigkeit bei der Auswertung des Bedingungsteils einer Regel für ein Beispiel überprüft wird. Eine Konjunktion von Selektoren bildet den Bedingungsteil einer Regel, in der Literatur zu CN2 teilweise auch als *Complex* bezeichnet. Das folgende Beispiel ist eine Entscheidungsregel, deren Bedingung aus zwei Selektoren zusammengesetzt ist, und die eine bestimmte Bedingung dafür beschreibt, wann das Wetter „schön“ ist:

```
IF (Aussichten = sonnig ∨ heiter) ∧ (Temperatur ≥ 25)
THEN (Klasse = schönes Wetter)
```

Man sagt, ein Selektor oder eine Bedingung *deckt* ein Beispiel *ab*, wenn das Beispiel diesen Ausdruck erfüllt. Die leere Bedingung ist eine Konjunktion aus null Selektoren und deckt alle Beispiele ab. Deswegen ist die leere Bedingung die allgemeinste Bedingung (*Most General Condition = MGC*). Für den in dieser Arbeit vorgenommenen Einsatz von CN2 für die Textkategorisierung liegen die Dokumente in Form TFIDF-gewichteter Vektoren vor, d. h. die Attribute entsprechen den Wörtern und die Attributwerte den TFIDF-Werten dieser Wörter (siehe Abschnitte 2.3 und 2.7).

Bei der Induktion der Regelmenge wendet CN2 eine iterierte, breitenbeschränkte *general-to-specific* Suche nach der jeweils nächsten besten Regelbedingung an, die die allgemeinste, also leere Bedingung schrittweise durch das konjunktive Hinzufügen einzelner Selektoren spezialisiert. Das Ziel beim Entwurf von CN2 war, den AQ-Algorithmus [Michalski et al., 1986] mit seiner strahlenförmigen Suche durch den Raum der Regelbedingung so zu modifizieren, daß er nicht nur mit den Trainingsdaten vollständig konsistente Regelbedingungen betrachtet, sondern seine Suche auch auf nicht perfekte Regeln erweitert, die auf vielen Beispielen korrekt sind, aber auch auf einigen Beispielen Fehler machen dürfen. Dadurch sollte CN2 robuster gegenüber Rauschen und Inkonsistenzen in den Trainingsdaten werden als der AQ-Algorithmus.

Um eine ungeordnete Reglemenge zu lernen, induziert CN2 nacheinander für jede Klasse eine eigene Menge ungeordneter Regeln für die angegebene Menge von Trainingsbeispielen (siehe Prozedur `CN2_ErzeugeRegelmenge` in Abbildung 3.1). Die Beispiele der Klasse, für die jeweils gerade Regeln gelernt werden, werden als *positive Beispiele* bezeichnet und die Beispiele der anderen Klassen als *negative Beispiele* (im Hinblick auf die zu lernende Klasse). Die Prozedur `CN2_ErzeugeRegelnFürEineKlasse` sucht auf den Trainingsbeispielen solange die jeweils beste Bedingung für eine die gewünschte Klasse vorhersagende Regel, bis keine weitere solche Bedingung gefunden wird (siehe Abbildung 3.1). Aus jeder gefundenen Bedingung wird eine Regel generiert und der Regelmenge zugefügt. Die von einer solchen Regel abgedeckten positiven Beispiele werden vor dem Lernen der jeweils nächsten Regel aus der Trainingsmenge entfernt.

Die CN2-Prozedur zum Finden der jeweils besten Regelbedingung (siehe Abbildung 3.2) entspricht der STAR-Prozedur des AQ-Algorithmus mit der oben erwähnten Modifikation. Diese spezialisierende Suchprozedur behält stets eine größenbeschränkte Menge (Star) der bisher besten gefundenen Regelbedingungen und untersucht immer nur Spezialisierungen von Elementen dieser Menge auf ihrer strahlenförmigen Suche durch den Raum der möglichen Regelbedingungen. Die maximale Größe der Menge wird durch die benutzerdefinierte Konstante `MaxStar` angegeben. Ein Regelbedingung wird dabei entweder durch das Hinzufügen eines Selektors zur Konjunktion spezialisiert oder durch das Entfernen eines disjunktiven Elements (Wertes) aus einem ihrer Selektoren.

```
Prozedur CN2_ErzeugeRegelmenge (Beispiele, Klassen, MaxStar,
                                KonfidenzSchwellwert);
  RegelMenge := {};
  Für jede Klasse in Klassen:
    Regeln := CN2_ErzeugeRegelnFürEineKlasse (Beispiele, Klasse, MaxStar,
                                              KonfidenzSchwellwert);
    RegelMenge := RegelMenge ∪ Regeln;
  Return RegelMenge;


Prozedur CN2_ErzeugeRegelnFürEineKlasse (Beispiele, Klasse, MaxStar,
                                         KonfidenzSchwellwert);
  Regeln := {};
  Wiederhole:
    BesteBedingung := FindeBesteSpezialisierung (Beispiele, Klasse, MaxStar,
                                                 KonfidenzSchwellwert);
    Falls BesteBedingung ≠ null
    dann Regeln := Regeln ∪ {„IF BesteBedingung THEN PREDICT Klasse“};
         Entferne von Beispiele alle Beispiele der Klasse Klasse, die von
         BesteBedingung abgedeckt werden (d. h. die BesteBedingung erfüllen);
  bis BesteBedingung = null;
  Return Regeln;
```

Abbildung 3.1: Der CN2-Algorithmus zum Lernen ungeordneter Regelmengen (im Original *CN2unordered* bzw. *CN2ForOneClass*, siehe [Clark und Boswell, 1991]).

```
Prozedur FindeBesteSpezialisierung (Beispiele, Klasse, MaxStar,
                                    KonfidenzSchwellwert);
  Sei MGC die allgemeinste Bedingung (Most General Condition, also „true“);
  Sei Selektoren die Menge aller Atttribut-Wert-Tests;
  Star := {MGC};
  NewStar := {};
  BesteBedingung := null;

  Solange Star nicht leer wiederhole:
    Für jede Bedingung ∈ Star:
      Für jeden AttributTest ∈ Selektoren, der noch nicht in Bedingung
      enthalten ist:
        NeueBedingung := Bedingung ∧ AttributTest;
        Falls NeueBedingung besser als BesteBedingung gemäß Laplace Accuracy
              und NeueBedingung statistisch signifikant gemäß Likelihood Ratio
                  Statistic mit Schwellwert KonfidenzSchwellwert
        dann BesteBedingung := NeueBedingung;
        Falls NeueBedingung ∉ Star und NeueBedingung erfüllbar
        dann NewStar := NewStar ∪ {NewComplex};
        Falls |NewStar| > MaxStar
        dann Entferne die gemäß Laplace Accuracy schlechteste Bedingung
             von NewStar;
    Star := NewStar;

  Return BesteBedingung;
```

Abbildung 3.2: Die CN2-Suchprozedur zum Finden der besten Spezialisierung (im Original *FindBestCondition*, siehe [Clark und Boswell, 1991]).

Die Prozedur `FindeBesteSpezialisierung` muß während des Lernprozeßes zwei Entscheidungen treffen. Zum einen wird eine Rangordnung der generierten Regelbedingungen benötigt, um zu bestimmen, welche Bedingung die aktuell beste ist (`BesteBedingung`), und um zu entscheiden, welche Regel gegebenenfalls als schlechteste aus der Kandidatenmenge zu entfernen ist, wenn die Maximalgröße der Menge (`MaxStar`) überschritten wird. In der überarbeiteten Version von CN2 [Clark und Boswell, 1991] wird als Bewertungsmaß für Güte einer Regelbedingung die *Laplace'sche Fehlerabschätzung* bzw. das sich daraus ergebende Genauigkeitsmaß $(1 - Laplace'scheFehlerabschätzung)$ benutzt. Diese erwartete Genauigkeit (*Laplace Accuracy*) berechnet sich wie folgt:

$$LaplaceAccuracy(n_c, n_{total}, k) = \frac{n_c + 1}{n_{total} + k} \tag{3.52}$$

wobei k die Anzahl der Klassen in der Domäne, n_c die Anzahl der von einer Regel korrekt vorhergesagten Beispiele der Klasse c und n_{total} die Anzahl aller von der Regel abgedeckten Beispiele ist.

Andererseits soll sichergestellt werden, daß jede gute Bedingung auch statistisch signifikant ist, d. h. eine Regelmässigkeit lokalisiert, die wahrscheinlich nicht zufälliger Natur ist, sondern eine echte Korrelation zwischen Attributwerten und Klassenzugehörigkeiten in der Trainingsmenge darstellt. Um die Signifikanz einer Bedingung zu testen, verwendet CN2 die sogenannte *Likelihood Ratio Statistic* [Kalbfleish, 1979]:

$$LikelyhoodRatio(F, E) = 2 \cdot \sum_{i=1}^{k} f_i \cdot \log \frac{f_i}{e_i} \tag{3.53}$$

wobei die Verteilung $F = (f_1, ..., f_k)$ die beobachtete Häufigkeitsverteilung der Beispiele beschreibt, die eine bestimmte Bedingung erfüllen, und $E = (e_1, ..., e_k)$ die erwartete Verteilung der gleichen Zahl von Beispielen ist, die man unter der Annahme einer zufälligen Auswahl aus der Trainingsmenge erhalten würde. Unter geeigneten Annahmen läßt sich zeigen, daß diese Statistik annähernd wie χ^2 (Chi Quadrat) mit $(k - 1)$ Freiheitsgraden verteilt ist. CN2 verwirft alle gefundenen Regeln, deren Signifikanz gemäß ihres Likelihood Ratio Wertes einen benutzerdefinierten Konfidenzschwellwert (`KonfidenzSchwellwert`) nicht erreichen. Zusammen sollen Laplace'sche Fehlerabschätzung und Signifikanztest dazu dienen festzustellen, ob bei der Suche gefundene Bedingungen sowohl „gut“ (hohe Klassifikationsgenauigkeit auf der Trainingsmenge) als auch „zuverlässig“ sind (die hohe Genauigkeit ist kein bloßer Zufall). Ein abschließender Test jeder Regel vor ihrer Aufnahme in die Regelmenge stellt sicher, daß jede Regel besser als eine Default-Regel ist, die immer die größte Klasse aus der Trainingsmenge für alle zu klassifizierenden Beispiele vorhersagt.

Die Anwendung der ungeordneten Klassifikationsregeln zur Klassifikation eines Textes bzw. seines Dokumentvektors erfolgt durch das Sammeln aller Regeln, deren Bedingungsteil durch den Text erfüllt wird. Hierbei sind *drei Fälle* denkbar: Der Text erfüllt die Bedingung genau einer Regel, mehrerer

Regeln oder keiner Regel. Ist *genau eine Regel auf den Text anwendbar*, so ist der Fall einfach. Der Text wird als zu der Klasse zugehörig klassifiziert, die von der anwendbaren Regel vorhergesagt wird. Wird der *Text durch mehrere Regeln abgedeckt*, so ergibt sich solange kein Problem, wie die Regeln alle die gleiche Klasse für den Text vorhersagen. Wenn verschiedene Regeln, die den Text abdecken, verschiedene Klassen vorhersagen, so ist dieser Konflikt aufzulösen, um dem Text eine eindeutige Klasse zuweisen zu können. Das induktive Lernsystem CN2 setzt hier folgende Heuristik zur Konfliktauflösung ein (siehe [Clark und Boswell, 1991]). Zu jeder Regel wird gespeichert, wieviele Trainingsbeispiele der einzelnen Klassen diese Regel abdeckt. Im Konfliktfall werden diese Verteilungen abgedeckter Beispiele von allen anwendbaren Regeln addiert, um die wahrscheinlichste Klasse für das zu klassifizierende Beispiel zu ermitteln. In der so erhaltenen kumulierten Verteilung wird die Klasse mit den meisten Beispielen als die vorhergesagte Klasse ausgewählt.

Etwas formaler läßt sich diese Entscheidungsregel wie folgt beschreiben. Seien $\mathcal{C} = \{C_1, C_2, ...\}$ die Menge der Klassen, $\mathcal{D}$ die Trainingsmenge (Dokumentenkollektion in Attributwertvektordarstellung), die $\mathcal{R} = \{R_1, R_2, ...\}$ die Menge der von CN2 induzierten Regeln und $PredictedClass(R_i)$ die von Regel R_i vorhergesagte Klasse. Sei ferner $Covers(R_i, d)$ gleich 1, wenn Regel R_i Beispiel d abdeckt, und 0 sonst. Die Anzahl n_{ij} der Trainingsbeispiele aus Klasse C_j, die von Regel R_i abgedeckt werden, läßt sich dann wie folgt schreiben:

$$n_{ij} = \sum_{d_k \in \mathcal{D} \ \wedge \ T(d_k) = C_j} Covers(R_i, d_k) \tag{3.54}$$

Dabei ist $T(d)$ die echte Klasse von Dokument d. Dann klassifiziert CN2 ein neues Dokument d' gemäß folgender Entscheidungsregel:

$$H_{CN2}(d') = \arg\max_{C_j \in \mathcal{C}} \sum_{R_i \in \mathcal{R} \ \wedge \ PredictedClass(R_i) = C_j \ \wedge \ Covers(R_i, d)} n_{ij} \tag{3.55}$$

Falls die Summe auf der rechten Seite von Gleichung (3.55) für alle Klassen C_j Null ist, also der zu klassifierende *Text von keiner Regel erfaßt* wird, so wird die *Default-Regel* angewandt, die jedes Beispiel als zu der in der Trainingsmenge am häufigsten vertretenen Klasse zugehörig klassifiziert.

Zu CN2 sehr ähnliche propositionale Regellernverfahren sind bereits zur Textklassifikation eingesetzt worden (siehe Abschnitt 3.9). Die von derartigen Begriffslernern erzeugten Regeln haben den Vorteil, für Menschen sehr viel verständlicher zu sein als die Klassifikationsmodelle anderer Klassifikatoren. Sie können bei Bedarf auch von Menschen verifiziert oder nachbearbeitet werden.

3.8 Der Entscheidungsbaum- und Regellerner C4.5

Das induktive Lernsystem C4.5 [Quinlan, 1993] wurde auf der Basis des Top-Down-Entscheidungsbaumlerners ID3 [Quinlan, 1986] entwickelt. Wie ID3 generiert auch C4.5 Klassifikatoren in der Form von Entscheidungsbäumen.

C4.5 kann diese aber auch wahlweise in geordnete Regelmengen überführen. Ein Entscheidungsbaum ist entweder ein Blatt, das eine Klasse anzeigt, oder ein Entscheidungsknoten, an dem ein bestimmter Attribut-Wert-Test über ein einzelnes Attribut auszuführen ist. Für jeden Ausgang dieses Attributtests hat der Entscheidungsknoten einen Unterbaum, der wiederum ein Entscheidungsbaum ist. Der Attributtest in einem Entscheidungsknoten ist einem Selektor in einer Entscheidungsregel sehr ähnlich (siehe Abschnitt 3.7), kann aber im Gegensatz zu diesem auch mehr als zwei mögliche Ausgangswerte haben. Also neben den Wahrheitswerten als Ausgangswerten eines Attributtests können beispielsweise auch die einzelnen Werte eines diskreten Attributes oder mehrere Intervalle aus dem Wertebereich eines kontinuierlichen Attributes den Unterbäumen eines Entscheidungsbaums zugeordnet werden, so daß C4.5 nicht nur binäre Entscheidungsbäume lernen kann.

Um ein Beispiel anhand eines Entscheidungsbaums zu klassifizieren, startet man an der Wurzel des Baums und bewegt sich so lange durch den Baum, bis man ein Blatt erreicht. Bei jedem Entscheidungsknoten wird der Ausgang des entsprechenden Attributtests für das gegebene Beispiel ermittelt, und die weitere Entscheidung über die Klasse des Beispiels ergibt sich aus dem Unterbaum, der dem Ausgang dieses Tests für dieses Beispiels entspricht. Wenn dieser rekursive Prozeß ein Blatt erreicht, wird die von diesem Blatt angezeigte Klasse als Klasse für das Beispiel vorausgesagt.

Zum Lernen eines Entscheidungsbaums wendet C4.5 folgende rekursive „Teile-und-Herrsche"-Strategie an [Quinlan, 1993]. Seien $\mathcal{C} = \{C_1, C_2, ...\}$ die Menge der Klassen und T eine Menge von Trainingsbeispielen, dann gibt es in einem Schritt während des Entscheidungsbaumkonstruktionsprozeßes drei mögliche Situationen:

- T enthält ein oder mehrere Beispiele, die alle zu einer Klasse C_j gehören: Der Entscheidungsbaum für T ist ein Blatt, das die Klasse C_j vorhersagt.
- T ist leer, enthält also keine Beispiele: Der Entscheidungsbaum ist ein Blatt, das die häufigste Klasse in seinem Elternknoten vorhersagt.
- T enthält Beispiele aus verschiedenen Klassen: Es wird der gemäß des *Gain-Ratio-Kriteriums* beste Attributtest ausgewählt (siehe Beschreibung weiter unten). Wenn dieser Test die disjunkten Ausgänge $\{O_1, O_2, ..., O_n\}$ hat, wird T so in disjunkte Teilmengen T_1, T_2, ..., T_n partitioniert, daß Teilmenge T_i alle Beispiele aus T enthält, die bei dem gewählten Attributtest den Ausgang O_i haben. Der Entscheidungsbaum für T ist dann ein Entscheidungsknoten, an dem der ausgewählte Test auszuführen ist, und der für jeden Ausgang des Tests genau einen Unterbaum hat. Die selbe Entscheidungsbaumkonstruktionsprozedur wird jetzt rekursiv auf jede der Teilmengen von T angewandt, so daß der i-te Zweig des hier konstruierten Entscheidungsknotens zu einem Entscheidungsbaum für die Teilmenge T_i führt.

Während ID3 das Maß *Information Gain* (siehe auch Abschnitt 2.6) zur Auswahl des jeweils besten Attributes an einem bestimmten Knoten benutzte, um den resultierenden Entscheidungsbaum so flach wie möglich zu halten, verwendet C4.5 das Maß *Gain Ratio*, weil Information Gain Attributtests mit sehr vielen Ausgängen gegenüber solchen mit wenigen Ausgängen stark bevorzugt, was für eine gute Generalisierung oft nicht wünschenswert ist. Gemäß der Informationstheorie gilt, daß die von einer Nachricht übertragene Information von der Wahrscheinlichkeit des Auftretens dieser Nachricht abhängt und als minus der Logarithmus zur Basis zwei dieser Wahrscheinlichkeit in Bits gemessen werde kann. Wenn also beispielsweise eine von acht gleich wahrscheinlichen Nachrichten auftritt, bedeutet dies eine übertragene Information von $-log_2(1/8)$, also 3 Bits. Sei X ein möglicher Attributtest mit n Ausgängen, der die Trainingsmenge T in die Teilmengen T_1, T_2, ..., T_n partitioniert. Sei $freq(C_j, T)$ die Anzahl der Trainingsbeispiele in T, die zur Klasse C_j gehören und sei $|T|$ die Anzahl der Beispiele in T. Wenn man aus der Menge T zufällig ein Beispiel auswählt und seine Klasse C_j bekannt gibt, so hat diese Nachricht eine Wahrscheinlichkeit von:

$$\frac{freq(C_j, T)}{|T|} \tag{3.56}$$

Somit liefert diese Nachricht eine Information von:

$$-\log_2 \frac{freq(C_j, T)}{|T|} \quad \text{bits} \tag{3.57}$$

Um die erwartete Information einer solchen Nachricht über die Klassenzugehörigkeit eines zufällig aus der Menge T gewählten Beispiels zu berechnen, bildet man die über die Klassenhäufigkeiten gewichtete Summe der Informationsmengen von Nachrichten über die Zugehörigkeit zu einzelnen Klasssen:

$$Info(T) = -\sum_{C_j \in \mathcal{C}} \frac{freq(C_j, T)}{|T|} \cdot \log_2 \frac{freq(C_j, T)}{|T|} \quad \text{bits} \tag{3.58}$$

Diese Informationsmenge wird auch als *Entropy* der Menge T bezeichnet. Nachdem die Beispielmenge T entsprechend der n Ausgänge des Attributtests X partitioniert worden ist, ergibt sich die erwartungsgemäß benötigte Information wie folgt:

$$Info_X(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \cdot Info(T_i) \tag{3.59}$$

Damit ist die durch die Partitionierung von T durch den Attributtest gewonnene Information wie folgt meßbar:

$$IG(X) = Info(T) - Info_X(T) \tag{3.60}$$

Dieses Maß erfaßt also den Informationsgewinn, der durch die Verwendung von X erzielt wird, und wird deswegen *Information Gain* genannt (siehe auch Abschnitt 2.6). Teilweise wird dieses Maß auch als *Mutual Information* zwischen dem Attributtest X und der Klasse bezeichnet.

Information Gain bevorzugt Attributtests mit sehr vielen Ausgängen. In dem Extremfall, daß ein Attribut die Menge T so partitioniert, daß jedes einzelne Beispiel aus T eine eigene Teilmenge bildet, ist der Informationsgewinn gemäß Information Gain maximal. Allerdings bildet ein solches Attribut keine gute Basis für eine gute Generalisierung, wie sie bei der Konstruktion eines Entscheidungsbaums gewünscht wird. Deswegen betrachtet man analog zu $Info(T)$ die Information die aus dem Wissen des Ausgangs des Attributtests für ein Beispiel entsteht, also die Information, die alleine durch Partitionierung der Menge T in n Teilmengen gemäß X entsteht (unabhängig von der Klassenzugehörigkeit der Beispiele in den Teilmengen):

$$SplitInfo(T) = -\sum_{C_j \in \mathcal{C}} \frac{|T_i|}{|T|} \cdot \log_2 \frac{|T_i|}{|T|} \quad \text{bits} \tag{3.61}$$

Das Maß *Gain Ratio* betrachtet den Informationsgewinn eines Attributes gemäß Information Gain im Verhältnis zu der durch eine solche Partitionierung bereits potentiell gewonnen Information:

$$GainRatio(X) = \frac{IG(X)}{SplitInfo(X)} \tag{3.62}$$

Das *Gain-Ratio-Kriterium* wählt einen Attributtest mit möglichst maximalem Gain Ratio unter der Nebenbedingung, daß dieser Test auch mindestens das durchschnittliche Information Gain aller untersuchten Attributtests erreicht.

Nachdem ein Entscheidungsbaum gemäß der oben beschriebenen Konstruktionsprozedur generiert wurde, wird er einem *Pruning*-Prozeß unterzogen, bei dem der Baum dadurch vereinfacht wird, daß einer oder mehrere Unterbäume durch Blätter oder einen ihrer Zweige ersetzt werden. Ziel dieses Schrittes ist es, eine eventuell erfolgte Überspezialisierung des Baums, d. h. eine Überanpassung an die Trainingsdaten (*Overfitting*) wieder rückgängig zu machen, um die erwartete Fehlerrate auf ungesehenen Daten zu senken. Unter der Annahme, daß man die Fehlerrate eines Baums und seiner Unterbäume (inklusive der Blätter) vorhersagen kann, wird der folgende Ansatz für das Pruning verwendet. Ausgehend von der Wurzel wird jeder Knoten untersucht, der kein Blatt ist. Wenn die Ersetzung eines Unterbaums durch ein Blatt oder durch seinen am häufigsten benutzten Zweig zu einer niedrigeren geschätzten Fehlerrate führt, wird diese Ersetzung durchgeführt. Rekursiv wird auf diese Weise der gesamte Baum durchlaufen. Da die erwartete Fehlerrate des gesamten Baums sinkt, wenn die erwartete Fehlerrate eines seiner Unterbäume sinkt, führt dieser Prozeß zu einem Baum mit minimaler erwarteter Fehlerrate im Hinblick auf die beschriebenen Formen des Prunings.

Das C4.5-System verwendet folgende pessimistische Fehlerabschätzung. Wenn ein Blatt des Baums N Trainingsbeispiele abdeckt und dabei E falsch klassifiziert, so ist sein Trainingsfehler E/N. Vereinfachend könnte man dies als die Beobachtung von E Ereignissen in N Versuchen ansehen und die N Trainingsbeispiele als eine entsprechende Stichprobe für ein solches Ereignis auf der gesamten Domäne betrachten, was sie natürlich in Wirklichkeit nicht sind. Unter Annahme dieser vereinfachenden Sichtweise läßt sich ein Konfidenzintervall für die Schätzung der Wahrscheinlichkeit dieses Ereignisses, also des Auftretens eines Fehlers, für ein vom Benutzer definiertes Konfidenzniveau mit Hilfe der Binomialverteilung berechnen. Sei $U_{CF}(E,N)$ die obere Grenze dieses Konfidenzintervalls. C4.5 verwendet diese obere Grenze für die Abschätzung der Fehlerrate der Vorhrsagen des untersuchten Blattes.

Alternativ zum Pruning eines Entscheidungsbaums kann C4.5 einen Baum auch wahlweise in eine geordnete Regelmenge transformieren, die dann in einem dem Pruning ähnlichen Schritt generalisiert wird. Zu Beginn der Transformation wird jeder Pfad des nicht durch Pruning vereinfachten Baums von der Wurzel zum einem Blatt zu einer initialen Regel. Die Entscheidungsknoten auf einem solchen Pfad werden zu Attributtests (Selektoren), die als Konjunktion die Regelbedingung ergeben. Die von einer Regel vorhergesagte Klasse ergibt sich aus dem Blatt, mit dem der zugehörige Pfad endet. Die so erhaltenen Regeln werden durch die Entfernung von Attributtests, die nicht zur Diskriminierung der Klassen beizutragen scheinen, aus den Regelbedingungen vereinfacht. Hierzu wird die bereits im vorherigen Absatz beschriebene pessimistische Fehlerabschätzung verwendet. Für jede Regel wird iterativ versucht, jeweils einen Attributtest zu entfernen, so lange hierdurch die geschätzte Fehlerrate der Regel nicht steigt.

Die so generalisierten Regeln werden nach den von ihnen vorhergesagten Klassen gruppiert und aus den Regeln zu einer Klasse werden jeweils solange einzelne Regeln entfernt, wie dadurch die erwartete Fehlerrate der gesamten Regelgruppe dieser Klasse nicht steigt. Die Regelgruppen der Klassen werden nun so sortiert, daß die Anzahl falscher positiver Klassifikationen minimiert wird. Falsche positive Klassifikationen sind die von den Regeln einer Klasses fälschlicherweise als zu dieser Klasse gehörig klassifizierten Beispiele. An diese geordnete Regelmenge wird eine Default-Regel angehängt, die für alle nicht von den Regeln abgedeckten Beispiele die am häufigsten unter den nicht abgedeckten Beispielen in der Trainingsmenge aufgetretene Klasse vorhersagt.

Soll ein neues Beispiel anhand der geordneten Regelmenge klassifiziert werden, wird es der Reihe nach mit den Regeln in der Menge verglichen. Das Beispiel erhält die von der ersten Regel, deren Bedingung es erfüllt, vorhergesagte Klasse. Erfüllt es keine der Regelbedingungen, wird die Default-Regel zur Klassifikation angewandt. Beispiele für den Einsatz von Entscheidungsbäumen oder aus diesen abgeleiteten Entscheidungsregeln in der Textkategorisierungen finden sich in [Lewis und Ringuette, 1994], [Moulinier et al., 1996] und [Joachims, 1997b].

3.9 Weitere Lernverfahren

Neben den bereits beschriebenen Verfahren CN2 (Abschnitt 3.7) und C4.5 (Abschnitt 3.8) sind auch schon andere auf einer *propositionalen Regeldarstellung* basierende Lernverfahren für die Textkategorisierung eingesetzt worden. Beispiele hierfür sind Anwendungen der Verfahren SWAP-1 [Apté und Damerau, 1994], RIPPER [Cohen, 1996] und CHARADE [Moulinier et al., 1996], [Moulinier und Ganascia, 1996]. Wie bei CN2 und C4.5 werden die Wörter in den verwendeten propositionalen Darstellungen als Attribute benutzt. Während CN2 und C4.5 jedoch die reel-wertigen TFIDF-Gewichte der Wörter als Attributwerte verwenden, nutzen diese drei Verfahren nur die binäre Information, ob ein Wort in einem Text vorkommt oder nicht. Für SWAP-1 wurde die Repräsentation dahingehend erweitert, daß auch Paare aufeinanderfolgender Wörter als Attribute berücksichtigt werden [Apté und Damerau, 1994]. Wie CN2 lernen diese Verfahren Regeln, deren Bedingungen Konjunktion von Attributtests sind.

Andere zur Textkategorisierung eingesetzte Regellernverfahren verwenden *relationale Repräsentationen*, die in ihrer Ausdruckskraft der propositionalen Darstellung überlegen sind. In einer relationalen Repräsentation können neben den Attributen eines Textes auch Beziehungen zwischen diesen Attributen ausgedrückt werden. Dies ermöglicht es beispielsweise, die Reihenfolge von Wörtern in einem Text zu repräsentieren, die bei propositionalen Darstellungen verlorengeht. In Experimenten zur Textklassifikation mit den Verfahren FOIL [Quinlan und Cameron-Jones, 1993] und FLIPPER [Cohen, 1995a] aus dem Bereich der *induktiven logischen Programmierung (ILP)* verwendete Cohen Relationen zwischen Wörtern, die die Nachbarschaft („benachbart", „höchstens durch ein Wort getrennt" und „höchstens durch zwei Wörter getrennt") bzw. die Reihenfolge („hinter" und „direkt hinter") von Wörtern in einem Text ausdrücken (siehe [Cohen, 1995a], [Cohen, 1995b]). Die durch die ausdrucksstärkere Repräsentation gegenüber propositionalen Lernverfahren bei manchen Problemen erzielbaren Verbesserungen sind allerdings eher gering und Cohen kommt zu dem Schluß, daß andere Parameter wie beispielsweise die Güte des vom jeweiligen Lernverfahren eingesetzten Pruning-Verfahrens das Lernergebnis stärker beeinflußen [Cohen, 1995a].

Kapitel 4

Verfahren zum Lernen bei Konzeptverschiebungen

In diesem Kapitel werden einige maschinelle Lernverfahren beschrieben, die speziell für das Problem des Lernens bei Konzeptverschiebungen entwickelt wurden. Zuerst werden einige der verbreitesten Ansätze für das Lernen sich verändernder Konzepte vorgestellt und anschließend in den folgenden Abschnitten anhand verschiedener Lernverfahren näher erläutert.

Ein direkt vom Problem sich verändernder Konzepte während des Lernens betroffener Bereich ist das *inkrementelle* oder *Online-Lernen* [Littlestone, 1988] (siehe auch Abschnitt 3.5), bei dem eine Hypothese inkrementell aus einzelnen klassifizierten Beispielen aus einem zeitlichen Strom von Beispielen gelernt wird. Veränderungen in der Lernumgebung können auch Veränderungen des zu lernenden Konzeptes bewirken, das das Lernverfahren mit jedem Lernschritt anzunähern versucht. Derartige Konzeptveränderungen können in verschieden starker Ausprägung und Schnelligkeit auftreten. Manche Autoren unterscheiden Konzeptverschiebungen deswegen grob in solche, die langsam und kontinuierlich erfolgen (*Concept Drift*), und solche, die in abrupten Sprüngen auftreten (*Concept Shift*) (siehe z.B. [Allan, 1996], [Kunisch, 1996]). In der algorithmischen Lerntheorie werden die Begriffe der Ausprägung einer Konzeptveränderung (*Extend of Drift*) und der Konzeptverschiebungsrate (*Rate of Drift*) genauer definiert (siehe Abschnitt 4.2.1).

Ausgehend von der grundlegenden Arbeit von Schlimmer und Granger [Schlimmer und Granger Jr., 1986] über den STAGGER-Algorithmus (siehe auch Abschnitt 4.1.1), ein inkrementellen Verfahren zum Lernen sich verändernder Konzepte, wurden verschiedene Techniken zur Behandlung von Konzeptverschiebungen entwickelt. Die allen gemeinsame Grundidee ist, daß das Lernverfahren seine Hypothesen dynamisch an das sich verändernde Zielkonzept anpassen können muß, wobei dies eventuell, aber nicht notwendigerweise erfordert, daß das Lernverfahren diese Konzeptveränderungen auch erkennt.

Die entwickelten Verfahren lassen sich nach unterschiedlichen Kriterien gliedern:

- **Zugrundeliegendes Lernszenario:**
  Wird *inkrementell* aus jedem einzelnen Beispiel durch Anpassung der aktuellen Hypothese gelernt *oder* wird aus einer Menge aufeinanderfolgender Beispiele (*Batch*) ein neue Hypothese generiert (*Batch-Lernen*) ?

- **Form der Datenhaltung (Speicherung bekannter Beispiele):**
  Wird die Information über die Klassifikation bereits gesehener Beispiele nur *implizit* im Klassifikationsmodell gespeichert, beispielsweise in Form von Gewichten oder anderen Konfidenzwerten für Hypothesen bzw. in den Hypothesen selbst, *oder* erfolgt auch eine *explizite Datenhaltung* ?
  Und im Fall einer expliziten Datenhaltung: Erfolgt eine *Beispielgewichtung* gemäß des Alters und/oder Nutzens der Beispiele für die Klassifikation *oder* wird mehr kategorisch ein *Zeitfenster* verwaltet ?

- **Erkennung von Konzeptverschiebungen:**
  Werden Konzeptverschiebungen explizit erkannt ?
  Werden hierzu beispielsweise bestimmte Performanzmaße des Klassifikators im Zeitablauf beobachtet (*Monitoring*) ?
  Oder wird ein Verfahren zur *Erkennung von Kontexten* eingesetzt ?

Natürlich lassen sich die Lernverfahren auch nach anderen Kriterien gliedern, aber die in diesem Kapitel vorgenommene Beschreibung der verschiedenen Ansätze orientiert sich an den hier vorgestellten Kriterien. Zunächst werden Online-Lernverfahren vorgestellt, die ohne eine explizite Speicherung von Beispielen auskommen (Abschnitt 4.1). In Abschnitt 4.2 werden Verfahren vorgestellt, die eine explizite Datenhaltung mit Hilfe von Zeitfenstern vornehmen, die entweder eine feste Größe haben oder mit Hilfe einer Heuristik flexibel an die aktuelle Situation angepaßt werden können. Es werden sowohl Online- als auch Batch-Lernverfahren vorgestellt, die diese Form der Datenverwaltung gemeinsam haben, und es werden einige theoretische Ergebnisse zum Lernen mit Zeitfenstern beschrieben. Abschnitt 4.3 beschreibt ein instanz-basiertes Online-Verfahren, das die gespeicherten Beispiele entsprechend ihres Alters und ihres Nutzens für die Klassifikation gewichtet. Einen ganz anderen Ansatz verfolgt der in Abschnitt 4.4 beschriebene Verfahrenstyp, der versucht, Kontexte zu erkennen, bei deren Wechsel sich das zu lernende Konzept verschiebt, um sich so besser an Konzeptverschiebungen anpassen zu können.

In den in dieser Arbeit beschriebenen Experimenten werden die hier vorgestellten Lernverfahren nicht eingesetzt. Stattdessen wird einer der beim maschinellen Lernen sich verändernder Konzepte am häufigsten eingesetzten Ansätze, nämlich der der Verwaltung von Zeitfenstern (Abschnitt 4.2), aufgegriffen und zusammen mit den in Kapitel 3 beschriebenen Textklassifikationsverfahren benutzt (siehe Kapitel 5 bzw. 6). Auch die Idee der automatischen Anpassung der Fenstergröße (Abschnitt 4.2.2) wird in einer auf die Textkategorisierung angepaßten Form übernommen.

4.1 Online-Lernen ohne Speicherung von Beispielen

4.1.1 Das Lernverfahren STAGGER

Der STAGGER-Algorithmus [Schlimmer und Granger Jr., 1986] war eines der ersten Verfahren zum Lernen sich verändernder Konzepte. STAGGER ist ein Lernverfahren für binäre Klassifikationsprobleme (o. B. d. A. mit den Klassen + und −) und kann nur endlich-wertige Attribute handhaben. Das Verfahren basiert auf einer verteilten Konzeptrepräsentation, die aus einer Menge gewichteter symbolischer Deskriptoren besteht. Ein solcher Deskriptor ist eine logische Verknüpfung von Attribut-Wert-Tests, die den Selektoren von CN2 sehr ähnlich sind (siehe Abschnitt 3.7), aber nur den Test auf Gleichheit als Vergleichsoperator verwenden. Ein Deskriptor ist eine disjunktive Liste konjunktiver Klauseln. Eine konjunktive Klausel ist eine Konjunktion von Attributtests, wobei ein Attributtest den Wert eines Attributes eines Beispiels auf das Enthaltensein in einer disjunktiven Liste zulässiger Attributwerte testet. Die Negation eines einzelnen Attributwertes wird durch die Disjunktion aller Werte des entsprechenden Attributes ohne diesen Wert ausgedrückt. Komplexere Negationen werden durch die Anwendung der de Morgan'schen Regeln realisiert.

STAGGER lernt inkrementell, d. h. mit dem Erhalt jedes neuen klassifizierten Beispiels modifiziert der STAGGER-Algorithmus die Deskriptoren, indem er ihre Gewichte anpaßt und eventuell neue Deskriptoren einführt und alte verwirft. Neue Deskriptoren werden durch die logische Verknüpfung bestehender Deskriptoren durch Konjunktion (Spezialisierung), Disjunktion (Generalisierung) oder Negation (Inversion) gebildet. Die initiale Konzeptbeschreibung besteht aus allen Attributtests, die jeweils mit zwei einheitlich auf Eins initialisierten Gewichten versehen sind. Bei jeder Klassifikation eines neuen Beispiels wird eines der beiden Gewichte verwendet. Wenn der Deskriptor das zu klassifierende Beispiel abdeckt, wird das eine Gewicht benutzt, wenn er es nicht abdeckt das andere. Das eine Gewicht gibt die Vorhersagekraft an, wenn der Deskriptor ein Beispiel abdeckt, und das andere die Vorhersagekraft, wenn er ein Beispiel nicht abdeckt.

STAGGER gewichtet jeden Deskriptor gemäß zweier Bayes'scher Maße. Das erste der beiden, die *logische Hinreichendkeit* (*Logical Sufficiency (LS)*), approximiert den Grad, zu dem das Vorhandensein einer Beobachtung F (z. B. „Beispiel d erfüllt den Deskriptor x") die Erwartung eines Ausgangs O (z. B. „Beispiel d gehört zur Klasse +") erhöht:

$$LS(F, O) = \frac{\Pr(F|O)}{\Pr(F|\neg O)} \tag{4.1}$$

LS hat einen Wert zwischen Null und plus unendlich und kann im Sinne relativer Chancen (*Odds*) interpretiert werden. Aus den Chancen zugunsten eines Ausgangs läßt sich die Wahrscheinlichkeit dieses Ausgangs mit Hilfe der Beziehung $p = \frac{Odds}{1+Odds}$ berechnen. Ein LS-Wert kleiner als Eins zeigt eine negative Korrelation zwischen F und O an, Eins die Unabhängigkeit von F und O und ein Wert größer als Eins eine positive Korrelation zwischen F und O.

Das zweite Maß zur Gewichtung eines Deskriptors ist die *logische Notwendigkeit* (*Logical Necessity (LN)*), die den Grad annähert, zu dem die Abwesenheit einer Beobachtung F die Erwartung eines Ausgangs O senkt:

$$LN(F,O) = \frac{\Pr(\neg F|O)}{\Pr(\neg F|\neg O)} \tag{4.2}$$

Auch LN hat einen Wert zwischen Null und positiv unendlich. Ein LN-Wert kleiner als Eins bedeutet eine positive Korrelation zwischen F und O, weil die Abwesenheit der Beobachtung F die Abwesenheit des Ausgangs O vorhersagt. Ein Wert größer als Eins zeigt eine negative Korrelation an. Ein Wert von Eins zeigt sowohl bei LS als auch bei LN an, daß die Beobachtung F keinen Einfluß auf den Ausgang O hat.

Bei der Klassifikation eines neuen Dokumentes d wird die Erwartung, daß dieses Beispiel zur Klasse + gehört, als Produkt aus der a priori Chance für ein Vorliegen der Klasse +, den LS-Gewichten aller durch d erfüllten Deskriptoren und den LN-Gewichten aller von d nicht erfüllten Deskriptoren berechnet:

$$Odds(+|d) = Odds(+) \cdot \prod_{x \in X \wedge Covers(x,d)} LS(x,+) \cdot \prod_{x \in X \wedge \neg Covers(x,d)} LN(x,+) \tag{4.3}$$

Dabei ist X die Menge der aktuellen Deskriptoren. $Covers(x,d)$ ist wahr, wenn Deskriptor x Beispiel d abdeckt, also Beispiel d den Deskriptor x erfüllt, und falsch sonst. Der resultierende Wert $Odds(+|d)$ beschreibt die Chancen (Odds) zugunsten der Zugehörigkeit von Beispiel d zur Klasse +. Ein Wert deutlich größer als Eins zeigt Konfidenz in die Zugehörigkeit von Beispiel d zur Klasse + an.

Zur Berechnung der zwei Bayes'schen Gewichtsmaße LS und LN werden für jeden Deskriptor vier Zähler C_+, C_-, I_+ und I_- verwaltet, die bei der Erzeugung des Deskriptors alle auf Null gesetzt und anschließend ständig wie folgt erhöht werden:

- Wenn der Deskriptor ein neues positives Beispiel (Klasse +) abdeckt, wird sein C_+ Zähler ums Eins erhöht (*Confirming Evidence*).

- Wenn der Deskriptor ein neues negatives Beispiel (Klasse −) nicht abdeckt, wird sein C_- Zähler ums Eins erhöht (*Confirming Evidence*).

- Wenn der Deskriptor ein neues positives Beispiel (Klasse +) nicht abdeckt, wird sein I_+ Zähler ums Eins erhöht (*Infirming Evidence*).

- Wenn der Deskriptor ein neues negatives Beispiel (Klasse −) abdeckt, wird sein I_- Zähler ums Eins erhöht (*Infirming Evidence*).

Mit Hilfe dieser Zähler lassen sich LS und LN wie folgt abschätzen [Schlimmer und Granger Jr., 1986]:

$$LS = \frac{C_+(I_- + C_-)}{I_-(C_+ + I_+} \quad \text{und} \quad LN = \frac{I_+(I_- + C_-)}{C_-(C_+ + I_+} \tag{4.4}$$

Die ebenfalls für die Klassifikation benötigten a priori Odds für ein positives Beispiel (Klasse $+$) für einen Deskriptor werden geschätzt durch den Quotienten aus der Summe der positiven Hinweise und der Summe der negativen Hinweise für die Gültigkeit des Deskriptors:

$$Odds(+) = \frac{C_+ + I_+}{C_- + I_-} \tag{4.5}$$

Die beschriebenen vier Zähler für jeden Deskriptor werden mit jedem neu eintreffenden Beispiel in der angegebenen Weise aktualisiert. Wenn eine Fehlklassifikation erfolgt, wird außerdem versucht, bestehende Deskriptoren durch Spezialisierung, Generalisierung und Inversion zu modifizieren. Auf diese Weise soll im von der speziellsten zur generellsten Hypothese halbgeordneten Hypothesenraum jeweils nur eine Grenzlinie möglichst effizienter Deskriptoren zur aktuellen Deskriptorenmenge von STAGGER gehören. Bei der Fehlklassifikation eines positiven Beispiels, wird durch die disjunktive Verknüpfung zweier (gemäß der Bayes'schen Gewichtsmaße) guter Deskriptoren versucht, eine Generalisierung zu finden, die das positive Beispiel abdeckt. Umgekehrt wird bei der Fehlklassifikation eines negativen Beispiels versucht, durch die konjunktive Verknüpfung zweier guter Deskriptoren eine Spezialisierung zu erreichen, die das negative Beispiel nicht mehr abdeckt. Für schlechte Deskriptoren kann eine Inversion erfolgen. Außerdem können logische Verknüpfungen durch Backtracking zurückgenommen werden, wenn sie sich als nicht mehr effizient erweisen. Gesteuert wird dieser Prozeß durch eine Heuristik, die sich an den Bayes'schen Gewichten der Deskriptoren orientiert.

Durch die Verwendung der Bayes'schen Maße LS und LN gelingt STAGGER eine recht gute Anpassung an Konzeptverschiebungen, ohne zu empfindlich auf durch Rauschen in den Daten verursachte Klassifikationsfehler zu reagieren [Schlimmer und Granger Jr., 1986]. Allerdings ist STAGGER sehr empfindlich gegen ein Übertrainieren, d. h. je länger dieses Verfahren auf einem bestimmten Zielkonzept trainiert wurde, desto langsamer paßt es sich an Konzeptverschiebungen an. Dies ist eine Folge der Art und Weise, wie die Zähler C_+, C_-, I_+ und I_- für einen Deskriptor verwaltet werden. Es erfolgt zwar keine explizite Datenhaltung, aber diese Zähler speichern alle korrekten und fehlerhaften Anwendungen der gesamten Existenzzeit des Deskriptors, so daß dies auf jeweils einen Deskriptor bezogen den gleichen Effekt hat wie die Verwaltung eines linear in der Anzahl der gesehenen Beispiele wachsenden Fensters auf den Trainingsdaten (*volles Gedächtnis*). Wenn also die Zähler C_+ und I_+ lange Zeit erhöht wurden, ohne daß auch die Zähler C_- und I_- erhöht wurden, werden entsprechend viele Fehlklassifikationen benötigt, um zu erkennen, daß ein Deskriptor nicht länger geeignet zur Beschreibung des Zielkonzeptes ist.

Durch den Einsatz einer geschickten Zeitfensterverwaltung (siehe Abschnitt 4.2) oder durch eine im Zeitablauf erfolgende Abwertung der Zähler ließe sich die Adaptivität von STAGGER möglicherweise verbessern. Um STAGGER für die Textklassifikation einsetzen zu können, müßte man sich außerdem wegen der Form, in der die Negation realisiert ist, auf endlich-wertige Attribute beschränken, also beispielsweise auf binäre Attribute über das Vorkommen eines Wortes in einem Text oder diskretisierte Wortgewichte. Bei der Behandlung von Konzeptverschiebungen in der Textklassifikation hätte STAGGER außerdem das Problem, daß er eine Änderung der zur Hypothesenkonstruktion verwendeten Attribute nicht vorsieht und somit das Auftreten neuer, sehr relevanter Wörter nicht berücksichtigen könnte.

4.1.2 Adaptive TFIDF-Klassifikatoren

Der in Abschnitt 3.1 vorgestellte *Rocchio-Algorithmus* [Rocchio Jr., 1971] wurde wie dort bereits beschrieben zuerst zum *Relevance Feedback* verwendet und später zum Einsatz in der Textklassifikation zum Lernen auf einer Trainingsmenge ohne initiale Anfrage modifiziert. Für den Einsatz beim Informationsfiltern mit Konzeptverschiebungen bietet es sich allerdings an, die Idee des Relevance Feedback wieder aufzugreifen und den Rocchio-Algorithmus wie in seiner ursprünglichen Form inkrementell einzusetzen, um aus einem alten Benutzerprofil und neuen, vom Benutzer klassifizierten Beispielen eine neues Benuzterprofil zu erzeugen.

Balabanovic et al. beschreiben in [Balabanovic et al., 1997] ein adaptives Programm namens *LIRA*, daß dem Benutzer jeden Tag eine Auswahl interessanter WWW-Seiten präsentiert. Der Benutzer bewertet diese Seiten anhand einer Skala von -5 bis $+5$, ob sie ihn eher nicht interessieren oder doch sehr interessieren. Mit Hilfe dieses Feedbacks versucht LIRA dann am nächsten Tag, bessere, d. h. dem aktuellen Interesse besser angepaßte, Vorschläge zu machen. Das Benutzerprofil P und die Dokumente d_i (WWW-Seiten) werden als Wortvektoren repräsentiert (siehe Abschnitt 2.3), wobei die Wörtern mit Hilfe einer Variante der TFIDF-Gewichtung gewichtet und die Dokumentvektoren auf die Länge Eins normiert werden. Sei e_i die Bewertung des Benutzers für Dokument d_i, dann berechnet LIRA das neue Profil P' wie folgt aus dem alten Profil P und dem Benutzer-Feedback für die Dokumente d_i aus der Menge D der vorgeschlagenen Seiten:

$$\vec{P'} = \vec{P} + \sum_{d_i \in D} e_i \cdot \vec{d_i} \tag{4.6}$$

Wie beim ursprünglichen Rocchio-Algorithmus werden also die relevanten Dokumente zur alten Anfrage (Benutzerprofil) addiert, da sie positive Gewichte e_i haben, und die nicht relevanten werden subtrahiert, da sie negative Gewichte e_i haben. Allerdings erlauben die Gewichte hier eine feinere Unterscheidung in verschiedene Relevanzgrade, als dies beim Originalalgorithmus der Fall war.

Auch das ebenfalls WWW-Seiten vorschlagende System *Fab* [Balabanovic, 1997] verwendet eine solche inkrementelle Form des Rocchio-Algorithmus, um Benutzerprofile an die sich ändernden Interessen der Benutzer anzupassen. Zusätzlich wird hier aber angenommen, daß sich Benutzerinteressen nicht nur mit jeder Inanspruchnahme des Systems, sondern allgemein mit der Zeit verändern. Deswegen werden die Benuzterprofile - zusätzlich zur Aktualisierung beim Eintreffen von Benutzer-Feedback - jede Nacht gemäß einer einfachen Alterungsfunktion abgeschwächt. Dies geschieht durch die Multiplikation aller Gewichte des Profils mit einem gewissen Faktor (hier 0.97).

Ein sehr ähnlicher Ansatz wird in [Allan, 1996] als *inkrementelles Relevance Feedback* für Informationsfilteraufgaben beschrieben. Auch hier werden eine inkrementelle Variante des Rocchio-Algorithmus und eine Alterungsfunktion eingesetzt. Die Alterungsfunktion wird auch hier für das schnellere „Vergessen“ alter Kontexte bei Konzeptverschiebung eingesetzt und führte bei den durchgeführten Experimenten zu einer deutlichen Verbesserungen der Ergebnisse [Allan, 1996].

Weder von Balabanovic et al. noch von Allan wird allerdings das Auftreten neuer Attribute berücksichtigt, was bei einer Vektorraumdarstellung mit Vektoren konstanter Länge auch nicht so leicht möglich ist. Wenn man davon ausgeht, daß im Dokumentenstrom neu auftretende Wörter auch sehr wichtig für die Relevanzentscheidung werden können, so erscheint eine Berücksichtigung solcher Wörter jedoch sinnvoll.

4.1.3 Weitere Online-Lernverfahren

In der Lerntheorie sind einige Online-Lernverfahren, die meist für das Lernen statischer Konzepte entworfen wurden, sehr ausführlich untersucht worden. In Abschnitt 3.5 wurde der *Winnow*-Algorithmus [Littlestone, 1988] bereits sehr ausführlich beschrieben. Ein sehr ähnlicher, inkrementell lernender Algorithmus, der ebenfalls im Bereich der Lerntheorie entworfen wurde, ist der *Weighted Majority Algorithmus* [Littlestone und Warmuth, 1994], der seine Gewichte wie Winnow multiplikativ aktualisiert. Obwohl sowohl Winnow als auch Weighted Majority ursprünglich für das Lernen statischer Konzepte entworfenen wurden, wurden sie auch schon erfolgreich in einer Domäne mit sich verändernden Konzepten eingesetzt [Blum, 1997].

Weighted Majority basiert auf der Annahme, daß aus allen gegebenen Attributen, vielleicht bereits eine kleine Menge aus beispielsweise zwei oder drei Attributen ausreicht, um gute Vorhersagen machen zu können. Bei der Verwendung von Zweiermengen, erzeugt Weighted Majority für jedes Paar von Attributen eine eigene Vorhersagestrategie, einen sogenannten *Experten*. Der globale Algorithmus sammelt dann für eine Klassifikation die Vorhersagen aller $\binom{n}{2}$ Experten und trifft seine Entscheidung in Form einer gewichteten Mehrheitsentscheidung der Experten.

Initial haben alle Experten das Gewicht Eins. Bei jeder falschen Vorhersage eines Experten wird sein Gewicht halbiert (oder gemäß eines anderen konstanten Faktors reduziert). Bei korrekten Vorhersagen bleibt sein Gewicht unverändert. Um seine Vorhersagen machen zu können, verwaltet jeder Experte eine Tabelle mit allen Wertekombinationen seiner Attribute, in der er die jeweils letzten k Vorkommen dieser Wertekombinationen zusammen mit der korrekten Klassifizierung der zugehörigen Beispiele ablegt. Für die Klassifikation eines Beispiels sieht der Experte in seiner Tabelle unter der Wertekombination dieses Beispiels für seine Attribute nach und sagt die in den letzten k Fällen am häufigsten für diese Kombination aufgetretene Klasse voraus.

Auch wenn sich in [Blum, 1997] eine erfolgreiche Anwendung der Verfahren Winnow und Weighted Majority auf eine Domäne mit Konzeptverschiebungen findet, lassen sich diese Ergebnisse nicht unbedingt auf Konzeptverschiebungen bei Informationsfilterproblemen übertragen, weil dort eine andere Art von Verschiebung zu erwarten ist. Bei den von Blum beschriebenen Experimenten tratt zwar eine Konzeptverschiebung auf, aber die gleichen Attribute waren vor und nach der Verschiebung relevant, lediglich die Bedeutung ihrer Werte für die Vorhersage war eine andere. In diesem Fall hat Weighted Majority nach nur k-maligem Auftreten einer Attributkombination bereits seine ganze Tabelle für diese Kombination ausgetauscht und sich perfekt angepaßt.

Wenn sich aber, wie es bei der Textklassifikation zu erwarten ist, auch die für die Klassifikation relevanten Attribute ändern, bekommen Winnow und Weighted Majority ähnliche Probleme wie das in Abschnitt 4.1.1 beschriebene STAGGER-Verfahren. Wenn sie sehr lange auf ein Zielkonzept trainiert wurden, wird es entsprechend lange dauern, bis sie sich an eine Konzeptverschiebung angepaßt haben, denn nun müssen vorher relevante Attribute im Vergleich zu den anderen Attributen genauso häufig abgewertet werden, wie sie vorher aufgewertet wurden. Umgekehrt müssen die jetzt relevanten Attribute entsprechend oft aufgewertet werden (bzw. alle anderen Attribute abgewertet werden), ehe die Anpassung gelingt. Außerdem müßten Winnow und Weighted Majority wie STAGGER für die Berücksichtigung neuer Attribute erweitert werden.

4.2 Online- und Batch-Lernen mit Zeitfenstern

4.2.1 Theoretische Ergebnisse zum Lernen bei Konzeptverschiebungen

Das Thema der Konzeptverschiebung ist in den letzten Jahren auch in der algorithmischen Lerntheorie bearbeitet worden. So haben beispielsweise Helmbold und Long ([Helmbold und Long, 1991], [Helmbold und Long, 1994]) und Kuh, Petsche und Rivest [Kuh et al., 1991] verschiedene Bedingungen untersucht, unter denen das effiziente Lernen sich verändernder Konzepte möglich ist. Sie gehen dabei von der Beobachtung aus, daß das Lernen sich verändernder Konzepte unmöglich ist, wenn man die Art der Konzeptverschiebungen nicht an irgendwelche Restriktionen bindet. Als Extremfall kann man sich eine zufällig zwischen der konstant Eins liefernden Funktion und der konstant Null liefernden Funktion springende Konzeptverschiebung vorstellen, die kein Algorithmus in mehr als 50% der Fälle korrekt vorhersagen kann. Ausgehend von dieser Beobachtung, betrachten sie verschiedene Einschränkungen des Ausmaßes (*Extent of Drift*) und der Frequenz von Konzeptverschiebungen (*Rate of Drift*).

Helmbold und Long ([Helmbold und Long, 1991], [Helmbold und Long, 1994]) nehmen eine permanente, aber sehr leichte Konzeptverschiebung an (Concept Drift), die möglicherweise mit jedem neuen Beispiel stattfindet. Das Ausmaß Δ der Konzeptverschiebung messen sie als den relativen Fehler zweier aufeinanderfolgender Konzepte, also als die Wahrscheinlichkeit, daß zwei aufeinanderfolgende Konzepte in ihrer Klassifikation eines zufällig gezogenen Beispiels nicht übereinstimmen. Helmbold und Long bewerten Algorithmen danach, welches Ausmaß an Konzeptverschiebung sie tolerieren können, wenn die Wahrscheinlichkeit für einen Vorhersagefehler höchstens ϵ sein soll und ihre Hypothesensprache $\mathcal{H}$ eine VC-Dimension d hat[1]. Zu ihren Hauptergebnissen gehört folgendes:

- Eine *obere Schranke für das maximal tolerierbare Ausmaß an Konzeptverschiebungen* für die Konzeptklassen der achsenparallelen Halbräume und Hyperrechtecke im n-dimensionalen Raum ($\mathcal{R}^n$):
  Kein Algorithmus, egal für welches n und $\epsilon < \frac{1}{12}$, kann Konzeptverschiebungen mit
  $$\Delta > c_1 \cdot \frac{\epsilon^2}{n} \tag{4.7}$$
  tolerieren, wobei c_1 eine Konstante ist.

- Einen allgemein einsetzbaren, aber von der Laufzeit ineffizienten, *Algorithmus*, der Konzeptverschiebungen bis zu
  $$\Delta \leq c_2 \cdot \frac{\epsilon^2}{d \cdot \ln \frac{1}{\epsilon}} \tag{4.8}$$
  tolerieren kann, wobei c_2 eine Konstante ist.

- Einen zweiten allgemein einsetzbaren *Algorithmus*, für den zwar nur bewiesen ist, daß er Konzeptverschiebungen bis zu
  $$\Delta \leq c_3 \cdot \frac{\epsilon^2}{d^2 \cdot \ln \frac{1}{\epsilon}} \tag{4.9}$$
  toleriert, der aber effizient ist, d. h. in Polynomialzeit lernt, wenn eine Hypothese, die den Fehler auf einer gegebenen Stichprobe näherungsweise minimiert, effizient gefunden werden kann. Dabei ist c_3 wiederum eine Konstante.

[1]Siehe Abschnitt 3.6 für eine Erläuterung des Begriffs der VC-Dimension.

Außerdem zeigen Helmbold und Long, daß es für ein Lernverfahren ausreicht, nur eine feste Anzahl der zuletzt gesehenen Beispiele zu berücksichtigen, also ein Zeitfenster fester Größe über dem Strom der Beispiele zu verwenden. Ihr Analyse führt zu einer groben Abschätzung der Fenstergröße, die für eine Konzeptverfolgung benötigt wird. Für den ersten oben genannten Algorithmus zeigen sie beispielsweise, daß unter den oben angegebenen Voraussetzungen eine Fenstergröße von

$$m = c_4 \cdot \frac{d}{\epsilon} \ln \frac{1}{\epsilon} \tag{4.10}$$

zur Konzeptverfolgung ausreicht, wobei c_4 eine Konstante ist. Hierzu ist anzumerken, daß dieser Algorithmus von Helmbold und Long seine aktuelle Hypothese bei Erhalt jedes neuen Beispiels neu aus den Daten des gesamten Fensters generiert. Es wird also für jedes Beispiel gelernt, aber nicht inkrementell, sondern mit einem Batch-Lernverfahren, daß die gesamte Trainingsmenge im aktuellen Fenster berücksichtigt.

Im Hinblick auf die praktische Anwendbarkeit dieser Ergebnisse für das Informationsfiltern bei sich verändernden Konzepten ergibt sich hier einerseits das Problem, daß es in den meisten Anwendungen aus Laufzeitgründen nicht praktikabel sein dürfte, bei Erhalt jedes einzelnen Beispiels den kompletten Lernvorgang zu wiederholen. Andererseits, und dies ist der wohl kritischere Punkt, läßt sich in wahrscheinlich keiner Anwendung eine derartige Beschränkung der Ausprägung von Konzeptverschiebungen garantieren. Beispielsweise wird kein Leser einer für ihn persönlich zusammengestellten elektronischen Zeitung sich von seinem Informationsfilter vorschreiben lassen, wie schnell er zwischen verschiedenen Themen wechseln darf und wie verschieden diese Themen sein dürfen, und trotzdem wird er eine möglichst schnelle Anpassung des Systems an seine Interessen wünschen. Unter diesen Gesichtspunkten haben die beschriebenen theoretischen Ergebnissen kaum einen praktischen Nutzen für das in dieser Arbeit adressierte Problem.

Auch Kuh et al. [Kuh et al., 1991] gehen von einem Batch-Lern-Szenario aus, bei dem eine Hypthose für jedes einzelne Beispiel auf den Daten des gesamten Fensters neu berechnet wird. Sie führen den Begriff der *PAC-Verfolgbarkeit* einer Sequenz von Konzepten, also eines sich verändernden Konzeptes, als eine Erweiterung von Valiants Paradigma der *PAC-Lernbarkeit* ein (*Probably Approximately Correct Learning* = wahrscheinlich annähernd korrektes Lernen, siehe [Valiant, 1984] oder für eine kurze Einführung [Morik, 1997]). Eine Begriffsklasse C ist *PAC-lernbar* durch einen Hypothesenraum H, wenn es einen Algorithmus gibt, der für jedes Zielkonzept $c \in C$ für eine beliebige, aber feste Wahrscheinlichkeitsverteilung über der Beispielmenge mit einer Wahrscheinlichkeit von mindestens $1 - \delta$ eine Hypothese $h \in H$ findet, deren Fehler nicht größer als ϵ ist, und dessen Laufzeit durch ein Polynom in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, der Länge der Konzeptbeschreibungen und der Länge der Beispielbeschreibungen beschränkt ist. Hierbei gilt $0 \leq \epsilon \leq \frac{1}{2}$ und $0 \leq \delta \leq \frac{1}{2}$.

Gemäß der Definition von Kuh et al. ist ein Konzeptsequenzenraum C' *PAC-verfolgbar* durch einen Hypothesenraum H, wenn es einen Algorithmus gibt, der für jede Konzeptsequenz $c' \in C'$ für eine beliebige, aber feste Wahrscheinlichkeitsverteilung über der Beispielmenge, deren korrekte Klassifizierung sich zu einem Zeitpunkt t durch das Konzept c_t aus der Sequenz c' ergibt, für alle ausreichend großen t und einen gemäß einer uniformen Verteilung zwischen 1 und t zufällig gewählten Zeitpunkt t' mit einer Wahrscheinlichkeit von mindestens $1 - \delta$ eine Hypothese $h \in H$ findet, deren Wahrscheinlichkeit für einen Fehler gegenüber $c_{t'}$ auf einem zufällig gezogenen Beispiel nicht größer als ϵ ist.

Während Helmbold und Long die tolerierbare *Ausprägung der Konzeptverschiebung (Extent of Drift)* begrenzt haben, versuchen Kuh et al. eine maximal von einem Lernverfahren handhabbare *Konzeptverschiebungsrate (Rate of Drift)* zu bestimmen, also eine maximal zulässige Häufigkeit von Konzeptwechseln. Eine Verschiebungsrate von höchstens λ bedeutet dabei, daß das Konzept im Durchschnitt für mindestens $\frac{1}{\lambda}$ Zeitschritte bzw. Beispiele stabil ist. Das Hauptergebnis von Kuh et al. ist eine *untere Schranke für die zulässige Verschiebungsrate*. Eine Sequenz PAC-lernbarer Konzepte ist PAC-verfolgbar, wenn für die Konzeptverschiebungsrate λ folgendes gilt:

$$\lambda \; < \; \frac{\delta}{2} \; \cdot \; m\left(\epsilon, \frac{\delta}{2}\right) \tag{4.11}$$

Dieses Ergebnis setzt eine minimale feste Fenstergröße von $w(\epsilon, \delta) = m(\epsilon, \frac{\delta}{2})$ voraus, wobei $m(\epsilon, \delta)$ von der allgemeinen Schranke für die Anzahl für eine garantierte PAC-Lernbarkeit notwendiger Beispiele abgeleitet ist (siehe [Blumer et al., 1989] oder auch [Morik, 1997]):

$$m(\epsilon, \delta) \; \geq \; \max\left(\frac{4}{\epsilon}\log_2\frac{2}{\delta}, \; \frac{8d}{\epsilon}\log_2\frac{13}{\epsilon}\right) \tag{4.12}$$

Dabei ist d die VC-Dimension des Hypthosenraums H ($VCdim(H)$). Die Ähnlichkeit mit der von Helmbold und Long hergeleiteten minimalen Fenster- bzw. Stichprobengröße (Gleichung (4.10)) ist offensichtlich.

Das bereits in der Erläuterung der theoretischen Ergebnisse von Helmbold und Long oben Gesagte gilt auch hier. In realen Informationsfilteraufgaben ist wahrscheinlich weder das ständige Neulernen einer Hypothese bei jedem neuen Beispiel praktikabel, noch wird man eine Höchstausprägung oder eine Höchstfrequenz von Konzeptverschiebungen garantieren können, was für die Gültigkeit der theoretischen Resultate notwendig wäre. Auch die von den theoretischen Ergebnissen vorgeschriebene relativ große minimale Fenstergröße würde in einer realen Anwendung sicherlich nicht zu einer besonders schnellen Anpassungsfähigkeit des Systems führen. Oder anders formuliert, mit einer kleineren Fensterrate lassen sich in der Praxis wahrscheinlich meistens bessere Ergebnisse erzielen, auch wenn die garantierbare Fehlerrate deutlich höher liegt.

4.2.2 Die FLORA-Familie von Lernverfahren

Die Algorithmen der FLORA-Familie [Widmer und Kubat, 1996] sind Online-Lernverfahren für binäre Klassifikationsprobleme (o. B. d. A. mit den Klassen + und −). Die FLORA-Algorithmen können sich an Veränderungen des Zielkonzeptes anpassen und einen Nutzen aus wiederkehrenden Kontexten ziehen. Diese Algorithmen verwalten ein Zeitfenster dynamischer Größe auf dem Strom der Beispiele und eine Menge aktuell vertrauenswürdiger Hypothesen. Außerdem speichern sie alte Konzeptbeschreibungen, um sie wiederverwenden zu können, wenn ein bereits gesehener Kontext wiederauftaucht. Beide Prozeße, also die Fensterverwaltung und das Wiedereinsetzen alter Hypothesen, werden von einer Heuristik gesteuert, die ständig das Systemverhalten beobachtet.

Während der erste FLORA-Algorithmus [Kubat, 1989] noch ein Zeitfenster fester Größe verwaltete, bei dem beim Eintreffen eines neuen klassifizierten Beispiels das jeweils älteste aus dem Fenster entfernt wurde, benutzen die hier näher beschriebenen Verfahren FLORA2 bis FLORA4 ein heuristisch in der Größe an die aktuelle Situation angepaßtes Fenster, daß allerdings ebenso die Beispiele aus der jüngsten Vergangenheit enthält. Sowohl das Hinzufügen von Beispielen zum als auch das Löschen von Beispielen vom aktuellen Fenster bewirken Modifikationen der Konzeptbeschreibung (Hypothese), um diese mit den Beispielen im Fenster konsistent zu halten. FLORA3 speichert Konzepte für eine spätere Wiederverwendung und testet ihre Nützlichkeit, wenn eine Kontextverschiebung wahrgenommen wird. FLORA4 baut auf FLORA2 und FLORA3 auf und ist speziell auf Robustheit gegenüber Rauschen in den Daten ausgelegt, das es mit Hilfe von Konfidenzintervallen von echten Konzeptverschiebungen zu trennen versucht.

Die Algorithmen der FLORA-Familie verwenden eine einfache propositionale Darstellung, die auf Attribut-Wert-Tests ohne Negation basiert. Diese Tests sind in ihrer Form deutlich eingeschränkter als beispielsweise die in Abschnitt 3.7 beschriebenen Selektoren, da sie nur den Gleichheitsoperator als Vergleichsoperator verwenden und immer nur auf die Gleichheit mit einem einzelnen Attributwert testen (nicht etwa auf die Gleichheit mit einem Wert aus einer Disjunktion von Werten). Ein *Deskriptor* ist eine Konjunktion solcher Attribut-Wert-Paare (Tests). Auch die Beispiele werden als Konjunktionen von Attribut-Wert-Paaren dargestellt.

Die Konzeptbeschreibung wird in Form von drei Mengen von Deskriptoren gespeichert. Die Menge *ADES (Accepted DEScriptors)* enthält Deskriptoren, die nur positive Beispiele abdecken und kann wie die beiden anderen Mengen als eine logische Formel in disjunktiver Normalform (DNF) interpretiert werden. Die Menge *NDES (Negative DEScriptors)* umfaßt Deskriptoren, die nur negative Beispiele abdecken, und wird verwendet, um eine Übergeneralisierung von ADES zu vermeiden. Die Menge *PDES (Potential DEScriptors)* enthält Deskriptoren, die sowohl positive als auch negative Beispiele abdecken, also zum aktuellen Zeitpunkt zu generell sind, aber vielleicht später relevant werden.

Für jeden Deskriptor werden zwei Zähler verwaltet, die angeben, wieviele positive und wieviele negative Beispiele dieser Deskriptor im aktuellen Fenster abdeckt. Diese Zähler werden mit jedem Hinzufügen eines Beispiels zum und jedem Löschen eines Beispiels vom Fenster aktualisiert. Sie werden für die Entscheidung benutzt, wann ein Deskriptor von einer der drei Deskriptorenmengen in eine andere verschoben werden muß. Deskriptoren, die weder ein positives noch ein negatives Beispiel im aktuellen Fenster abdecken, werden ganz von den Hypothesen gelöscht. Änderungen des Fensters können den Inhalt der Deskriptorenmengen in folgender Weise beeinflußen:

- Das *Hinzufügen eines positiven Beispiels* zum Fenster kann zur Folge haben, daß dieses Beispiel als Deskriptor in ADES aufgenommen wird, bestehende Deskriptoren in ADES verstärkt oder so generalisiert werden, daß sie das neue Beispiel abdecken, und/oder das bestehende Deskriptoren von NDES zu PDES verschoben werden.

- In analoger Weise führt das *Hinzufügen eines negativen Beispiels* zur Aufnahme als neuer Deskriptor in NDES oder zur Verstärkung oder Generalisierung bestehender Deskriptoren in NDES, oder zur Verschiebung bestehender Deskriptoren von ADES zu PDES.

- Wenn ein Beispiel aus dem Fenster entfernt wird, kann das zur Abschwächung bestehender Deskriptoren führen (d. h. die entsprechenden Zähler werden reduziert) oder sogar zur Entfernung aus der aktuellen Konzeptbeschreibung (falls die Zähler auf Null gesunken sind) oder zu einer Verschiebung von PDES zu ADES oder NDES (falls dies das letzte falsch abgedeckte Beispiel war).

Eine Generalisierung eines Deskriptors in ADES (bzw. NDES) erfolgt nur, wenn ein neues positives (bzw. negatives) Beispiel von keinem Deskriptor in dieser Menge abgedeckt wird. Wenn weder eine Abdeckung des Beispiels gegeben ist, noch eine Generalisierung zum Erreichen der Abdeckung möglich ist, ohne daß der generalisierte Deskriptor einen Deskriptor in einer der beiden anderen Deskriptorenmengen subsumiert, so wird das neue Beispiel in ADES (bzw. PDES) als neuer Deskriptor und potentieller Ausgangspunkt späterer Generalisierungen aufgenommen. Der einzige eingesetzte Generalisierungsoperator ist das Entfernen von Attribut-Wert-Paaren aus der Konjunktion eines Deskriptors (*Dropping Condition Rule* [Michalski, 1983]). Es gibt keinen Spezialisierungsoperator. Durch die bei Generalisierungen durchgeführten Subsumptionstest sind die drei Deskriptorenmengen konsistent und redundanz-frei.

Das Verfahren *FLORA2* ([Widmer und Kubat, 1992], [Widmer und Kubat, 1996]) versucht die Größe seines Fensters während des Lernprozeßes dem aktuellen Grad der Konzeptverschiebung und dem momentanen Lernzustand anzupassen. Bei dem Auftreten einer Konzeptverschiebung sollte das Fenster verkleinert werden, um eine möglichst schnelle Anpassung an die neue Datenlage zu erreichen. Wenn das Konzept stabil erscheint, soll die Fenstergröße fest bleiben, und sonst soll es langsam wachsen, bis eine stabile Konzeptbeschreibung generiert werden kann.

Variablen- und Parameterbenennung:

N	:	Anzahl von ADES abgedeckter positiver Beispiele;
S	:	Größe von ADES in Anzahl von Literalen (Attributtests);
Acc	:	aktuelle Accuracy (beobachtet über den letzten Beispielen);
$\|W\|$	:	aktuelle Fenstergröße;
lc	:	Schwellwert für niedrige Abdeckung von ADES (benutzerdefinierter Parameter);
hc	:	Schwellwert für hohe Abdeckung von ADES (benutzerdefinierter Parameter);
p	:	Schwellwert für akzeptable Accuracy (benutzerdefinierter Parameter);

```
Prozedur BestimmeAnzahlZuVergessenderBeispiele (lc, hc, p)
  Falls (N/S < lc) oder
        ((Acc < p) und Abnehmend(Acc))          /* Drift vermutet: */
    dann Anzahl := 0.2 · |W|;                   /*   verkleinere Fenster um 20%. */
  sonst falls (N/S > 2 · hc) und (Acc > p)      /* Situation sehr stabil: */
    dann Anzahl := 2;                           /*   verkleinere Fenster um 1. */
  sonst falls (N/S > hc) und (Acc > p)          /* Situation stabil genug: */
    dann Anzahl := 1;                           /*   lasse Fenstergröße fest. */
    sonst Anzahl := 0;                          /* Vergrößere Fenster um 1. */
  Return Anzahl;
```

Abbildung 4.1: Die Heuristik der FLORA-Algorithmen zur Fensteranpassung (im Original *how_many_to_forget*, siehe [Widmer und Kubat, 1996]). Bei der Anzahl zu vergessender Beispiele ist zu beachten, daß direkt vor Anwendung der Heuristik ein neues Beispiel zum Fenster hinzugefügt wurde. Deswegen entspricht eine Anzahl von einem zu vergessenden Beispiel dem Beibehalten der Fenstergröße.

Zur *Erkennung von Konzeptverschiebungen* verwendet FLORA2 zwei *Indikatoren*: die Accuracy des Systems (über einer festen Anzahl vergangener Klassifikationen) und eine syntaktische Eigenschaft der entstehenden Hypothesen. Die grundlegende Annahme ist, daß ein deutlicher Abfall der Accuracy oder ein sehr plötzliches starkes Ansteigen der Anzahl der Deskriptoren in ADES ein Signal für eine Konzeptverschiebung sein können. Letzteres wird über die *Abdeckung* der Deskriptorenmenge ADES ermittelt, die sich aus dem Verhältnis der von den Deskriptoren in ADES abgedeckten Beispiele zur Größe der Deskriptorenmenge gemessen in der Anzahl der Literale (Attributtests) aller Deskriptoren der Menge ergibt. Da die beiden Indikatoren Accuracy und Abdeckung sehr stark von der Charakteristik der Lernaufgabe abhängen, muß die in Abbildung 4.1 dargestellte Fensteranpassungsheuristik von FLORA2 über drei vom Benutzer einzustellende Parameter an die jeweilige Anwendung anpaßt werden: einen Schwellwert für eine akzeptable Accuracy (p) sowie zwei Schwellwerte für eine hohe bzw. niedrige Abdeckung (hc bzw. lc).

Wenn die Fensteranpassungsheuristik von FLORA2 eine Konzeptverschiebung erkannt zu haben glaubt, reduziert sie die Größe des Fensters um 20%. Wenn eine sehr stabile Situation vorliegt, wird die Fenstergröße um Eins reduziert, um nicht unnötig viele Beispiele zu speichern. Beim Vorliegen einer ausreichend stabilen Hypothese bleibt die Fenstergröße unverändert. Falls keine der drei vorgenannten Situationen gegeben ist, geht die Heuristik davon aus, daß mehr Daten für das Erreichen einer stabilen Situation benötigt werden, und vergrößert das Fenster inkrementell um jeweils ein Beispiel, indem das jeweils älteste Beispiel nicht „vergessen" wird.

Das Verfahren *FLORA3* ([Widmer und Kubat, 1993], [Widmer und Kubat, 1996]) baut auf FLORA2 auf und verwendet dessen Heuristik zur Fensteranpassung. Zusätzlich erlaubt FLORA3 das *Erkennen wiederkehrender Kontexte* und die *Wiederverwendung alter Hypothesen*. Dieser Ansatz läßt sich dann einsetzen, wenn es eine endliche Anzahl versteckter Kontexte gibt, die wiederkehren können. Dies kann zyklisch geschehen wie beispielsweise bei den vier Jahreszeiten oder in ungeordneter Weise. Die Wiederverwendung von Hypothesen aus wiederkehrenden Kontexten kann dem System zu schnellerer Konvergenz verhelfen, wenn ein gleicher oder ähnlicher Kontext schon einmal erschienen ist.

Der Mechanismus der Wiederwendung alter Konzeptbeschreibungen ist eng mit der Fensteranpassungsheuristik verzahnt. Wenn die aktuelle Hypothese gemäß der Fensteranpassungsheuristik stabil ist, wird die aktuelle Konzeptbeschreibung für eine etwaige spätere Wiederverwendung gespeichert, außer wenn bereits vorher schon eine identische Menge von ADES Deskriptoren gespeichert wurde. Wenn die Heuristik zur Fensteranpassung eine Konzeptverschiebung zu erkennen glaubt und die Fenstergröße verkleinert, wird versucht eine auf die aktuelle Situation passende Konzeptbeschreibungen aus dem Speicher alter ADES Deskriptorenmengen zu finden, sie anhand der aktuell im Fenster befindlichen Daten anzupassen und eventuell anstelle der aktuellen Deskriptoren in ADES einzusetzen, wenn sie geeigneter als die aktuelle Konzeptbeschreibung erscheint. Da ADES Deskriptormengen gemäß ihrer Konstruktion immer alle positiven und keine negativen Beispiele abdecken, erfolgt die Bewertung der Konzeptbeschreibungen über ihre Komplexität, d. h. eine kompaktere Beschreibung wird besser bewertet als eine komplexere.

Im allgemeinen ist es beim inkrementellen Lernen schwierig, zwischen echten Konzeptverschiebungen und kleineren Unregelmäßigkeiten zu unterscheiden, die auf Rauschen in den Trainingsdaten zurückzuführen sind. Verfahren, die so entworfen sind, daß sie besonders schnell auf Konzeptveränderungen reagieren, überreagieren bei verrauschten Daten leicht und zeigen ein unstabiles Verhalten. Dies geschieht auch bei FLORA2 und FLORA3, weil sie stets streng auf die Konsistenz der Hypothesen mit den aktuellen Daten achten. Ein ideales Lernverfahren sollte *Stabilität und Robustheit gegenüber verrauschten Daten* mit einer guten Anpassungsfähigkeit an Konzeptveränderungen kombinieren, zwei etwas gegensätzliche Anforderungen, zwischen denen es einen Kompromiß zu finden gilt. Eben dies versucht das Verfahren *FLORA4* ([Widmer, 1994], [Widmer und Kubat, 1996]), das die Fensteranpassungsheuristik von FLORA2 und den Wiederverwendungsmechanismus für Konzeptbeschreibungen wiederkehrender Kontexte von FLORA3 übernimmt, aber die strenge Konsistenzforderung gegen ein weniger scharfes Zuverlässigkeitsmaß für die Vorhersagekraft der Deskriptoren ersetzt.

Hierzu beobachtet FLORA4 die Accuracy jedes einzelnen Deskriptors im Zeitablauf und berechnet für ein vom Benutzer vorgegebenes Konfidenzniveau ein Konfidenzintervall für diese Genauigkeitsabschätzung auf den Daten des aktuellen Zeitfensters. Die Entscheidungen darüber, wann ein Deskriptor von einer Deskriptorenmenge in eine andere zu verschieben oder ganz zu entfernen ist, werden anhand dieser Konfidenzintervalle und der beobachteten relativen Klassenhäufigkeiten getroffen. Ein Deskriptor bleibt so lange in ADES, wie seine Accuracy mit hoher Konfidenz über der relativen Häufigkeit der von ihm vorhergesagten Klasse liegt. Die Verwaltung der Deskriptorenmenge erfolgt nun wie folgt:

- Ein Deskriptor bleibt solange in ADES, wie der untere Endpunkt seines Accuracy-Konfidenzintervalls über dem oberen Endpunkt des Konfidenzintervalls der relativen Häufigkeit positiver Beispiele liegt. Deskriptoren aus PDES, die diese Bedingung erfüllen, werden nach ADES verschoben.

- Deskriptoren aus ADES, deren Konfidenzintervall sich mit dem Intervall der relativen Häufigkeit positiver Beispiele überlappt, werden nach PDES verschoben.

- Deskriptoren, deren obere Konfidenzintervallsgrenze unter der unteren Konfidenzintervallsgrenze der relativen Häufigkeit positiver Beispiele befindet, werden verworfen.

- Deskriptoren in NDES bleiben so lange dort, wie sie in analoger Weise zu den ADES-Deskriptoren akzeptable Vorhersagen für negative Beispiele machen. Wenn sie dies nicht mehr tun, werden sie nicht wie bei FLORA2 und FLORA3 in PDES aufgenommen, sondern verworfen.

Generalisierungen in ADES und NDES dürfen jetzt auch negative bzw. positive Beispiele abdecken, so lange ihre Accuracy dies zuläßt. PDES beinhaltet nun alternative Generalisierungen die zur Zeit unzuverlässig sind, weil sie zu viele negative Beispiele abdecken oder eine insgesamt zu geringe Anzahl von Beispielen abdecken (und deswegen sehr große Konfidenzintervalle haben). In den übrigen Aspekten entspricht FLORA4 dem Verfahren FLORA3.

Die Algorithmen FLORA2 bis FLORA4 wie in [Widmer und Kubat, 1996] beschrieben sind nur für symbolische Attribute anwendbar, aber in [Kubat und Widmer, 1995] findet sich eine Erweiterung von FLORA4, die auch numerische Attributwerte handhaben kann. Das erweiterte FLORA4 verwendet als einfachen Generalisierungsoperator Michalski's *Closing Interval Rule* [Michalski, 1983] und approximiert numerische Konzepte entsprechend durch achsenparallele Hyper-Rechtecke. In dieser Variante wäre FLORA4 wahrscheinlich auch gut für Anwendungen in der Textklassifikation einsetzbar.

4.2.3 Das Lernverfahren FRANN

Das FRANN-Verfahren [Kubat und Widmer, 1995] kann sowohl numerische als auch symbolische Attribute handhaben, wobei symbolische Attribute dazu in numerische überführt werden. Dieses Verfahren benutzt wie die Verfahren FLORA2 bis FLORA4 eine Heuristik zur dynamischen Größenanpassung des Zeitfensters auf den Daten. Allerdings aktualisiert FRANN seine Hypothese nicht für jedes einzelne Beispiel, sondern es kommen in jedem Zeitschritt N neue Beispiele hinzu. Als Indikatoren für die Bestimmung des Zeitpunktes und der Art der Fensteranpassung werden die Accuracy auf den letzten M Beispielen und die aktuelle Fenstergröße herangezogen. FRANN sucht mit Hilfe einer Hill-Climbing-Suche die jeweils beste Teilmenge der Daten aus dem aktuellen Zeitfenster heraus und erzeugt mit ihrer Hilfe ein Netzwerk von radialen Basisfunktionen (*Radial Basis Function Network*, siehe z.B. [Mitchell, 1997]). Dieses neuronale Netz wird dann als Klassifikator eingesetzt. FRANN eignet sich besonders zum Lernen bei numerischen Attributen und bei Klassen mit nicht-linearen Grenzen.

4.2.4 Fensterverwaltung für Lerner statischer Konzepte

Fensterverwaltungsansätze mit Fenstern fester Größe sind nicht nur für Online-Lernverfahren eingesetzt worden, sondern auch schon zusammen mit Lernverfahren für statische Konzepte, die so in dynamischen Domänen eingesetzt werden konnten. Mitchell et al. [Mitchell et al., 1994] beschreiben Versuchsergebnisse mit einem persönlichen Kalenderassistenzsystem, *Calendar APprentice (CAP)*, das aus den Daten vergangener Termine bei Angabe von Art und Anwesenden eines zukünftigen Termins den Ort, die Dauer, den Wochentag und die Startzeit des zukünftigen Termins vorhersagen soll. Zum Lernen haben sie das Entscheidungsbauminduktionsverfahren ID3 (siehe auch Abschnitt 3.8) mit einem Zeitfenster fester Größe eingesetzt, wobei sie eine günstige Größe des Fensters empirisch ermittelt hatten. Ein anderes Beispiel ist der Einsatz eines Neuronale Netzes (siehe z.B. [Mitchell, 1997]) zur Vorhersage der Kreditwürdigkeit potentieller Kunden, das mit Hilfe des Backpropagation-Algorithmus und eines Zeitfensters auf den Daten bisheriger Kunden trainiert wurde [Taylor et al., 1997].

4.3 Online-Lernen mit Gewichtung der Beispiele

Der in diesem Abschnitt vorgestellte adaptive Nächster-Nachbar-Klassifikator ([Kunisch, 1996], [Taylor et al., 1997]) ist eine dynamische Variante des in Abschnitt 3.4 beschriebenen k-NN-Klassifikators (mit $k = 1$), bei der die gespeicherten Beispiele initial mit einem Gewicht von Eins versehen werden, das im Laufe der Zeit gemäß einer *Alterungsfunktion* abnimmt und unter bestimmten Umständen bei korrekten bzw. falschen Vorhersagen des Klassifikators erhöht bzw. verringert wird. Die Alterung der Beispiele erfolgt durch eine Multiplikation ihrer Gewichte mit einem Faktor λ in jedem Zeitschritt ($0 < \lambda < 1$). Die Anpassung der Gewichte der Beispiele gemäß ihres Nutzens für die Lösung der Klassifikationsausfgabe geschieht additiv:

- Wenn ein neues Beispiel von seinem nächsten Nachbarn korrekt klassifiziert wird, aber von seinem zweit-nächsten Nachbarn falsch klassifiziert werden würde, so wird das Gewicht des nächsten Nachbarn um eine Konstante γ erhöht.
- Wenn ein neues Beispiel von seinem nächsten Nachbarn falsch klassifiziert wird, aber von seinem zweit-nächsten Nachbarn korrekt klassifiziert werden würde, so wird das Gewicht des nächsten Nachbarn um eine Konstante ϵ verringert. Sinkt das Gewicht des nächsten Nachbarn unter Null, wird dieser nächste Nachbar aus der Trainingsmenge entfernt.

Auf diese Weise spiegelt sich im Gewicht einer Instanz neben ihrem Alter auch ihre Nützlichkeit für den Klassifikationsprozeß wieder.

Die Gewichtung der bekannten Beispiele hat im Fall des beschriebenen Nächster-Nachbar-Verfahrens eine gewisse Ähnlichkeit mit der in Abschnitt 4.1 beschriebenen Gewichtung von Hypothesen(teilen) bei einigen Online-Verfahren, da beim Nächster-Nachbar-Klassifikator die Beispiele praktisch Teil der Hypothese sind. Wie bei STAGGER und Winnow erfolgt also praktisch eine Aufwertung für die Klassifikation günstiger und eine Abwertung für die Klassifkation ungünstiger Hypothesenbestandteile. Und wie beim adaptiven TFIDF-Klassifikator von Balabanovic, Shoham und Yun erfolgt zusätzlich eine Abwertung des alten Hypothesenanteils.

Der Ansatz der Datengewichtung entsprechend dem Alter und eventuell auch der gemäß der Nützlichkeit der Daten ließe sich auch auf andere Lernverfahren anwenden, die eigentlich auf das Lernen statischer Konzepte ausgelegt sind, insofern sie eine Gewichtung der einzelnen Trainingsbeispiele zulassen. Bei einem solchen Einsatz, würde der Unterschied zwischen einer Gewichtung der Daten und einer Gewichtung von Hypothesen(teilen) deutlicher. Die Verwendung gewichteter Daten über einen längeren Zeitraum ohne ein zusätzliches Zeitfenster und ohne Entfernen von Beispielen mit einem Gewicht, das unter einem gewissen Schwellwert liegt, hat den Nachteil, daß der Datenverwaltungsaufwand und je nach Klassifikationsverfahren auch der Lern- und/oder Klassifikationsaufwand erheblich steigen.

4.4 Online-Lernen mit Kontexterkennung

In einer dynamischen Domäne auftretende Kontextveränderungen sind eine potentielle Quelle von Konzeptverschiebungen und somit kann die Entdeckung von Kontextänderungen in manchen Fällen äquivalent zur Erkennung von Konzeptverschiebungen sein. Die in Abschnitt 4.2.2 beschriebenen Verfahren FLORA3 und FLORA4 verfügen über einen Mechanismus zum Erkennen von versteckten Kontexten. Manchmal ist der Kontext allerdings nicht vollständig vor dem Lernverfahren verborgen, sondern anhand von Indikatorvariablen erkennbar. Die in diesem Abschnitt kurz beschriebenen Verfahren METAL(B) und METAL(IB) ([Widmer, 1996b], [Widmer, 1996a]) versuchen solche Kontextindikatoren automatisch zu finden. Sie benutzen eine Zwei-Ebenen-Architektur, bei der auf der Basisebene ein Lernverfahren online neue Beispiele klassifiziert und auf einer höheren Ebene (*Meta-Ebene*) ein Algorithmus versucht, Attribute zu identifizieren, die die Erkennung von Kontexten erlauben (*Meta-Attribute*), um so das Verhalten des Systems bei Kontextänderungen zu verbessern. Das zugrunde liegende Lernverfahren ist dabei ein naiver Bayes'scher Klassifikator *(B)* bzw. ein instanz-basiertes Verfahren *(IB)* ([Widmer, 1996b], [Widmer, 1996a]). Wie bei den FLORA-Verfahren wird ein Zeitfenster auf den Daten verwaltet.

Die Definition des Begriffs *kontextueller Attribute*, also solcher Attribute, die die Erkennung von Kontextänderungen erlauben, basiert dabei auf der Definition des Begriffs vorhersagekräftiger Attribute. Ein Attribut F_i gilt als *vorhersagekräftig*, wenn es zwischen dem Vorliegen eines seiner Werte v_{ij} und dem Vorliegen einer bestimmten Klasse C_k eine signifikante Korrelation gibt, d. h. wenn $\Pr(C_k|F_i = v_{ij})$ signifikant von der a priori Wahrscheinlichkeit $\Pr(C_k)$ der Klasse C_k abweicht. Ein Attribut F_i gilt als *kontextuell*, wenn es zwischen dem Vorliegen eines seiner Werte v_{ij} und der Vorhersagekraft eines Attributes F_k eine signifikante Korrelation gibt, d. h. wenn $\Pr(F_k = v_{kl} \text{ ist vorhersagekräftig} \mid F_i = v_{ij})$ signifikant von $\Pr(F_k = v_{kl}$ ist vorhersagekräftig) abweicht. Zur Feststellung einer signifikanten Abweichung wird auf beiden Ebenen ein χ^2-Test verwendet.

Die im aktuellen Zeitfenster kontextuellen Attribute werden zur Identifikation des aktuellen Kontextes herangezogen. Das Lernverfahren auf der Basisebene verwendet nur die Beispiele aus dem Zeitfenster für die Klassifikation eines neuen Beispiels, die die gleichen Werte für die kontextuellen Attribute haben wie das neue Beispiel. Es verwendet also nur Trainingsbeispiele aus dem gleichen Kontext. Ob die METAL-Verfahren für die Textklassifikation einsetzbar sind, hängt sicherlich sehr stark von der konkreten Anwendung ab, da vermutlich bei vielen Textkategorisierungsproblemen nicht mit kontextuellen Attribute zu rechnen ist. Es ist außerdem fraglich, ob sich durch eine Erkennung kontextueller Attribute wirklich ein Vorteil ergibt, denn viele Lernverfahren ohne eine Erkennung solcher Attribute würden wahrscheinlich kontextuelle Attribute genauso wie die vorhersagekräftigen Attribute konjunktiv in ihre Hypothesen einbauen und dadurch wahrscheinlich bei der Klassifikation ein vergleichbares Resultat erzielen.

Kapitel 5

Verwaltung von Zeitfenstern

Wenn man beim Lernen von Konzepten, die sich im Zeitablauf verändern können, kein inkrementelles Lernverfahren verwendet, das ohne eine Speicherung bereits gesehener Beispiele auskommt (siehe z.B. Abschnitt 4.1), so muß man eine Menge von Trainingsbeispielen verwalten, anhand derer das eingesetzte Lernverfahren sein Klassifikationsmodell aktualisieren oder neu lernen kann. Bei jeder Konzeptverschiebung ergibt sich allerdings das Problem, daß die Daten, die aus der Zeit vor dieser Verschiebung stammen, für die aktuelle Situation nicht länger repräsentativ sind. Wie in Kapitel 4 beschrieben, kann ein Verfahren diese nicht länger repräsentativen Beispiele entweder durch eine Datengewichtung in ihrer Bedeutung abwerten oder mit Hilfe eines Zeitfensters dafür sorgen, daß diese Beispiele schließlich irgendwann aus der Trainingsmenge entfernt werden und die Qualität der Trainingsmenge nicht auf Dauer, sondern allenfalls vorübergehend beeinträchtigen. Offensichtlich hat die Art dieser Datenverwaltung einen großen Einfluß auf die Aktualität der Daten, die dem Lernverfahren zur Verfügung stehen, und somit auf die Adaptivität des Lernverfahrens an Konzeptverschiebungen.

Dieses Kapitel beschäftigt sich mit der Verwaltung von Zeitfenstern auf einem Strom von Trainingsdaten für Lernverfahren für die Textklassifikation bei sich verändernden Konzepten. Es wird ein *Batch-Lern-Szenario* angenommen, bei dem in jedem Zeitschritt N neue Beispiele (1 Batch) aus dem Dokumentenstrom zu verarbeiten sind. Die einfachste Form von Zeitfenstern hat eine feste Größe (*„Fixed Size“*), die im gesamten Zeitablauf unverändert bleibt, und bei der die jeweils ältesten Beispiele entfernt werden, wenn neue Beispiele zum Fenster hinzugefügt werden. Ein Spezialfall der Verwaltung von Fenstern fester Größe ist die Verwendung einer Fenstergröße von nur einem Batch, d. h. das Klassifikationsmodell für den aktuellen Batch wird nur auf den Daten des direkt vorhergenden Batches gelernt. Dieser Ansatz hat also kein „Gedächtnis“, das über die Länge eines Batches hinausreicht, und wird deshalb im folgenden auch als *„No Memory“*-Ansatz bezeichnet.

Ein zentrales Problem bei der Verwendung eines Fensters fester Größe ist die *Wahl der „richtigen“ Fenstergröße*. Dies läßt sich am besten an den

Folgen einer unangemessenen Fenstergröße erläutern. Während ein zu kleines Fenster dem Lernverfahren keine ausreichende Anzahl von Beispielen zur Verfügung stellt, um eine stabile und ausreichend generelle Konzeptbeschreibung zu finden, verzögert ein zu großes Fenster die Reaktionsfähigkeit des Lernverfahrens, was insbesondere dann ein großer Nachteil ist, wenn sehr häufig Konzeptverschiebungen auftreten oder die Veränderung sehr abrupt und stark ausgeprägt ist. Bei der Wahl einer festen Fenstergröße muß man also einen Kompromiß zwischen einer schnellen Anpassungsfähigkeit bei Konzeptverschiebungen einerseits und der Stabiliät und besseren Lernergebnissen in verschiebungsfreien Phasen andererseits finden.

Die Grundidee einer *adaptiven Fensterverwaltung* ist es, die Größe des Zeitfensters während des Lernprozeßes dem aktuellen Grad der Konzeptverschiebung und der aktuellen Performanz des Systems anzupassen (*„Adaptive Size“*, siehe auch Abschnitt 4.2). Das Ziel ist es, den Vorteil kleiner Fenstergrößen, also die hohe Adaptivität bei Konzeptverschiebungen, mit den Vorteilen großer Fenstergrößen, also die Stabilität und die besseren Lernergebnisse bei längere Zeit stabilen Konzepten, zu verbinden und die jeweiligen Nachteile zu vermeiden. Beim Auftreten einer Konzeptverschiebung sollte das Fenster verkleinert werden, um eine möglichst schnelle Anpassung an die neue Datenlage zu erreichen. Wenn keine Konzeptverschiebung vorliegt, sollte das Fenster auf eine Größe wachsen, die das Erzielen guter Generalisierungen und somit stabiler Lernergebnisse erlaubt.

Von zentraler Bedeutung für eine effiziente adaptive Fensterverwaltung ist die zuverlässige Erkennung von Konzeptverschiebungen. Deswegen ist die Wahl geeigneter *Indikatoren für das Erkennen von Konzeptverschiebungen* wichtig, die es ermöglichen, Unregelmässigkeiten in den Daten (Rauschen) von echten Konzeptverschiebungen zu unterscheiden. In Abschnitt 5.1 werden solche Indikatoren für Textklassifikationsprobleme vorgestellt. Diese Indikatoren bilden die Basis der in Abschnitt 5.2 vorgestellten adaptiven Zeitfensterverwaltung für Textklassifikationsaufgaben mit sich verändernden Konzepten.

5.1 Indikatoren für die Erkennung von Konzeptverschiebungen bei der Textklassifikation

Um angemessen auf eine Konzeptverschiebung reagieren zu können, ist es für eine adaptive Zeitfensterverwaltung wichtig, den Zeitpunkt und möglichst auch den Grad der Verschiebung zuverlässig erkennen zu können. Für die Erkennung von Konzeptverschiebungen lassen sich verschiedene *Indikatoren* einsetzen, von denen hier eine Auswahl vorgestellt wird. Weil die Wahl einer geeigneten Menge von Indikatoren in der Regel domänenabhängig ist, liegt der Schwerpunkt hier auf Indikatoren für den Einsatz in Textklassifikationsanwendungen.

Man kann diese Indikatoren in folgende Kategorien gliedern:

- *Performanzmaße*: Der Indikator ist ein Maß für die aktuelle Performanz des Klassifikationssystems (siehe Abschnitt 2.8 für eine Beschreibung diverser Performanzmaße).

- *Eigenschaften des Klassifikationsmodells*: Der Indikator beschreibt eine Eigenschaft der aktuellen Hypothese, bei einer Regelmenge also beispielsweise deren Komplexität.

- *Eigenschaften der Daten*: Der Indikator beschreibt eine Eigenschaft der aktuellen Trainingsdaten wie beispielsweise die aktuelle Klassenverteilung, also die relativen Häufigkeiten der Klassen.

Die Lernverfahren FLORA2 bis FLORA4 (Abschnitt 4.2.2) und FRANN (Abschnitt 4.2.3) verwenden das Performanzmaß Accuracy, also die aktuelle Klassifikationsgenauigkeit des Klassifikators (siehe auch Abschnitt 2.8), als einen ihrer Indikatoren. Die FLORA-Verfahren betrachten außerdem die Abdeckung ihrer aktuellen Konzeptbeschreibung, die sich aus dem Verhältnis der Anzahl der abgedeckten positiven Beispiele zur Anzahl der Literale in der Konzeptbeschreibung ergibt. Während Indikatoren in Form von Performanzmaßen auf sehr verschiedenartige Lernverfahren anwendbar sind, sind Indikatoren, die wie die Abdeckung Eigenschaften des Klassifikationsmodells beschreiben, immer auf eine bestimmte Hypothesensprache bezogen und somit nur sehr begrenzt auf andere Lernverfahren übertragbar. Weil die in diesem Kapitel vorgestellte adaptive Fensterverwaltung für möglichst viele Textklassifikationsverfahren einsetzbar sein soll und in den in dieser Arbeit beschriebenen Experimenten (Kapitel 6) auch zusammen mit sehr verschiedenartigen Verfahren eingesetzt wird, wird bei ihr auf den Einsatz von Indikatoren verzichtet, die Eigenschaften des Klassifikationsmodells beschreiben.

Neben der Klassenverteilung sind auch andere Indikatoren vorstellbar, die *Eigenschaften der aktuell vorliegenden Daten* beschreiben und damit unabhängig vom eingesetzten Lernverfahren anwendbar sind. So schlägt beispielsweise Allan [Allan, 1996] vor, den Zeitpunkt und eventuell auch den Grad einer Konzeptverschiebung an einer beobachteten *Veränderung der als relevant klassifizierten Dokumente* zu erkennen, die sich mit Hilfe der Bildung von Clustern von Dokumenten feststellen lassen könnte.

Ein anderer Indikator könnten die gemäß eines Attributselektionskriteriums (siehe Abschnitt 2.6) für die Klassikation am besten geeigneten Attribute sein. Wenn beispielsweise plötzlich neue Attribute unter den *besten n Attributen* auftauchen, könnte das ein Signal dafür sein, daß sich die aktuelle Hypothese eventuell verbessern lassen könnte, wenn sie diese Attribute berücksichtigen würde. Während Indikatoren in Form von Performanzmaßen und Eigenschaften des Klassifikationsmodells die Erkennung von *Verschlechterungen* der Perfomanz des Systems bzw. der Struktur der Hypothese erlauben und dazu dienen sollen, derartigen Verschlechterungen möglichst schnell zu beheben, bieten sie keine Möglichkeit, eine Veränderung zu erkennen, die keine Verschlechterung dieser Kriterien mit sich bringt, stattdesen aber das *Potential einer Leistungssteigerung* des Systems bietet. Eine Beobachtung der Eigenschaften der aktuell verfügbaren Daten kann die Möglichkeit bieten, eventuell auch solche Potentiale zu erkennen. Wenn beispielsweise Dokumente eines sehr jungen Forschungsgebietes als relevant betrachtet werden, so sind sie vielleicht anfangs schwer von Dokumenten verwandter Gebiete zu trennen. Sobald sich aber in diesem Forschungsgebiet ein eigenes Vokabular gebildet hat, könnte dieses dazu beitragen, die Dokumente dieses Gebietes besser von denen anderer Gebiete zu trennen. In diesem Fall würde das neue Vokabular sich unter den für die Klassifikation besonders relevanten Attributen etablieren.

Das in diesem Kapitel vorgestellte Verfahren zur Verwaltung der Fenstergröße setzt allerdings nur Performanzmaße als Indikatoren ein, weil sie wahrscheinlich die zuverlässigsten Indikatoren für Konzeptverschiebungen sind. Wie für die Verfahren der FLORA-Familie und FRANN wird hier die *Accuracy* des aktuellen Klassifikationsmodells als Indikator eingesetzt. Da bei Textklassifikationsaufgaben in der Regel eine kleine Anzahl für eine Kategorie relevanter Dokumente einer im Verhältnis sehr viel größeren Zahl nicht relevanter Dokumente gegenübersteht, ist die Accuracy eines Textklassifikationssystems alleine zur Bewertung des Systems nur sehr bedingt geeignet, weil eine Default-Regel, die alle Dokumente als irrelevant klassifiziert, bereits eine sehr hohe Accuracy erzielt (siehe hierzu auch Abschnitt 2.8). Aus dem gleichen Grund erscheint die aktuelle Accuracy eines Systems als einziger Indikator für Konzeptverschiebungen nicht ausreichend. Eine Änderung dessen, was relevant ist, betrifft nur einen recht kleinen Teil der Dokumente und macht sich deshalb wahrscheinlich in der Accuracy auch nicht so stark bemerkbar. Aus diesem Grund werden hier die in Abschnitt 2.8 beschriebenen im Information Retrieval gebräuchlichen Maße *Precision* und *Recall* als weitere Indikatoren eingesetzt, da sie eine Beobachtung der Performanz der Vorhersage auf der kleineren, meist wichtigeren Klasse der relevanten Dokumente erlauben.

5.2 Entwurf einer adaptiven Zeitfensterverwaltung für Textklassifikationsprobleme

Wie die Fensteranpassungsheuristik der FLORA-Algorithmen (Abschnitt 4.2.2) versucht die in diesem Abschnitt vorgestellte Heuristik, die Größe des Zeitfensters auf den Trainingsdaten dem aktuellen Grad der Konzeptverschiebung anzupassen. Allerdings setzt das hier vorgestellte Verfahren zum Teil andere Indikatoren zur Erkennung von Konzeptverschiebungen ein als FLORA (siehe Abschnitt 5.1) und reagiert anders auf einen Teil der erkannten Situationen.

Wenn der Wert eines der beobachteten Indikatoren Accuracy, Recall und Precision auf eine abrupte, starke Konzeptverschiebung (Concept Shift) hindeutet, wird das Fenster radikal auf die Größe eines Batches verkleinert, die hier als untere Schranke für die Fenstergröße angesehen wird, um alle nicht länger repräsentativen Daten so schnell wie möglich zu verwerfen und so dem Lernverfahren eine schnellstmögliche Anpassung an die Veränderung zu erlauben. Wenn eine weniger ausgeprägte Konzeptverschiebung (Concept Drift) vermutet wird, wird die Fenstergröße ebenfalls verkleinert, allerdings weniger radikal, da die alten Daten zu einem großen Teil offensichtlich noch repräsentativ sind und die Datenbasis für das Lernen nicht unnötig stark beschnitten werden soll. Hier soll ein Kompromiß zwischen einer schnellen Adaptivität und stabilen Lernergebnissen erzielt werden. Wenn das Zielkonzept stabil erscheint, wird das Fenster vergrößert, um die Datenbasis für das Lernverfahren zu vergrößern. Solange keine Konzeptverschiebung vorzuliegen scheint, werden alle gesehenen Daten gespeichert. Es wird also stets versucht, eine möglichst große Datenbasis bereitzuhalten, die möglichst keine Konzeptverschiebung enthält. Dies geht auf die Beobachtung zurück, daß bei den meisten Textklassifikationsaufgaben mit statischen Konzepten, die Performanz mit der Größe der Trainingsmenge zunimmt. Während einer Phase ohne erkennbare Konzeptverschiebung wird ein quasi statisches Konzept vermutet und deswegen wie beschrieben verfahren. Eine obere Schranke für die Fenstergröße ist in der hier vorgestellten Fassung der Heuristik nicht vorgesehen, kann aber natürlich, wenn es die Anwendung aus Speicherplatz- oder Laufzeitgründen erfordert, leicht eingebaut werden.

Im Gegensatz zur hier vorgestellten Heuristik vergrößert die Heuristik der FLORA-Algorithmen ihr Fenster nur, wenn die Konzeptbeschreibung nicht stabil genug erscheint. Wenn die Hypothese sehr stabil erscheint, verkleinert sie das Fenster sogar. Ein weiterer Unterschied zur Heuristik der FLORA-Verfahren besteht in der Art und Weise, wie über das Vorliegen einer Konzeptverschiebung entschieden wird. Bei den FLORA-Verfahren definiert der Benutzer absolute Schwellwerte für die Werte der Indikatoren und verändert diese Werte im späteren Zeitablaufs des Systems nicht. Da diese Werte sehr domänenabhängig sind, müssen vom Benutzer experimentell günstige Werte ermittelt werden [Widmer und Kubat, 1996].

Bei der hier vorgestellten Fensteranpassungsheuristik wird versucht, die Parameterwahl weniger domänenabhängig zu gestalten und die Schwellwerte im Zeitablauf dynamisch anzupassen. Hierzu werden die Werte der Indikatoren im Zeitablauf betrachtet und die Schwellwerte, bei deren Unterschreitung eine Konzeptverschiebung angenommen wird, als untere Grenze eines Konfidenzintervalls um den Mittelwert des jeweiligen Indikators auf den letzten M Batches berechnet. Die Weite des Konfidenzintervalls ergibt sich dabei als Vielfaches des Standardfehlers des jeweiligen Indikators über den letzten M Batches. Der entsprechende Faktor für das Konfidenzniveau, hier mit α bezeichnet, ist vom Benutzer anzugeben[1].

[1] In allen Experimenten zu dieser Arbeit wurden die Werte der Indikatoren über die letzten $M = 10$ Batches gemittelt und $\alpha = 5.0$ verwendet, um eine sehr hohe Konfidenz dafür zu erreichen, daß es sich bei einer als Konzeptverschiebung erkannten Situation auch wirklich um eine Konzeptverschiebung handelt. Liegen weniger als M alte Werte des Indikators vor, wird entsprechend über weniger Werte gemittelt.

Variablen- und Parameterbenennung:

M : Anzahl der Batches für Mittelwertbildung (benutzerdef. Parameter);
α : Konfidenzniveau für Verschiebungserkennung (benutzerdef. Parameter);
β : Höchstanteil eines Indikatorwertes an seinem Vorgängerwert bei einem Concept Shift (benutzerdefinierter Parameter);
γ : Fensterreduktionsrate bei Concept Drift (benutzerdefinierter Paramter);
$|W_t|$: aktuelle Fenstergröße (Ist-Größe);
$|W_{t+1}|$: neue Fenstergröße (neue Soll-Größe);
$|B|$: Größe eines Batches (in Anzahl von Dokumenten);
Acc_t : aktueller Accuracy-Wert;
Acc_{t-1} : letzter Accuracy-Wert;
$Avg_M(Acc)$: mittlere Accuracy auf den letzten M Batches;
$StdErr_M(Acc)$: Standardfehler der Accuracy auf den letzten M Batches;
Rec_t, Rec_{t-1}, $Avg_M(Rec)$, $StdErr_M(Rec)$: entsprechende Recall-Werte;
$Prec_t$, $Prec_{t-1}$, $Avg_M(Prec)$, $StdErr_M(Prec)$: entsprechende Precision-Werte;

Prozedur `BestimmeNeueFenstergröße` ($|W_t|$, M, α, β, γ)
Falls $((Acc_t < Avg_M(Acc) - \alpha \cdot StdErr_M(Acc))$ **und** $(Acc_t < \beta \cdot Acc_{t-1}))$ **oder**
$((Rec_t < Avg_M(Rec) - \alpha \cdot StdErr_M(Rec))$ **und** $(Rec_t < \beta \cdot Rec_{t-1}))$ **oder**
$((Prec_t < Avg_M(Prec) - \alpha \cdot StdErr_M(Prec))$ **und** $(Prec_t < \beta \cdot Prec_{t-1}))$
dann /* Concept Shift vermutet: reduziere Fenster auf Batchgröße */
$|W_{t+1}| := |B|$;

sonst falls $(Acc_t < Avg_M(Acc) - \alpha \cdot StdErr_M(Acc))$ **oder**
$(Rec_t < Avg_M(Rec) - \alpha \cdot StdErr_M(Rec))$ **oder**
$(Rec_t < Avg_M(Prec) - \alpha \cdot StdErr_M(Prec))$
dann /* Concept Drift vermutet: reduziere Fenstergröße um $\gamma \cdot 100\%$ */
$|W_{t+1}| := \max(|B|,\ |W_t| - \gamma \cdot |W_t|)$;

sonst /* Stabiles Konzept vermutet: vergrößere Fenster um einen Batch */
$|W_{t+1}| := |W_t| + |B|$;

Return $|W_{t+1}|$;

Abbildung 5.1: Fensteranpassungsheuristik für Textkategorisierungsprobleme. Die Anzahl der ältesten Beispiele, die nach Hinzufügen des aktuellen Batches aus dem Zeitfenster entfernt werden müssen, berechnet sich als $|B| - (|W_{t+1}| - |W_t|)$.

Um in einer als Konzeptverschiebung erkannten Situation zu entscheiden, ob es sich um eine starke, abrupte Verschiebung (Concept Shift) oder um eine eher kleinere oder über einen längeren Zeitraum erfolgende Verschiebung (Concept Drift) handelt, werden die Werte der Indikatoren des aktuellen Batches mit den Werten des jeweils letzten Batches verglichen. Wenn einer von ihnen um mehr als einen benutzerdefinierten Faktor β unter seinem Vorgängerwert liegt, wird das Vorliegen eines Concept Shifts angenommen, anderenfalls das Vorliegen eines Concept Drifts[2].

Abbildung 5.1 zeigt die beschriebene Fensteranpassungsheuristik. Wenn einer der Indikatoren Accuracy, Recall oder Precision auf einen Concept Shift hindeutet, wird die Fenstergröße auf ein Minimum reduziert, auf die Größe

[2] In allen Experimenten zu dieser Arbeit wurde $\beta = 0.5$ verwendet.

eines Batches. Wenn statt dessen nur ein Concept Drift vermutet wird, wird die Fenstergröße um einen benutzerdefinierten Faktor γ, die Fensterreduzierungsrate, verkleinert (solange die Fenstergröße dabei das Minimum, also die Größe eines Batches, nicht unterschreitet)[3]. Anderenfalls wird eine Situation ohne Konzeptverschiebung angenommen und das Fenster um die Beispiele des aktuellen Batches erweitert, ohne alte Beispiele zu entfernen. Für eine initiale Anzahl M_0 von Batches wird keine Anpassung der Fenstergröße vorgenommen, damit die Mittelwerte und Standardfehler der Indikatoren eine zuverlässige Schätzung erreichen können[4]. Die initiale Fenstergröße wird vom Benutzer vorgegeben[5].

[3] In allen Experimenten zu dieser Arbeit wurde $\gamma = 0.5$ verwendet.

[4] In allen Experimenten zu dieser Arbeit wurde $M_0 = 5$ verwendet.

[5] In allen Experimenten zu dieser Arbeit wurde die initiale Fenstergröße $|W_0|$ auf drei Batches gesetzt.

Kapitel 6

Experimente

In den im folgenden beschriebenen Experimenten wird die Performanz der in Kapitel 3 vorgestellten Lernverfahren in Kombination mit verschiedenen Fensterverwaltungsansätzen und ohne den Einsatz eines Zeitfensters verglichen. Bei einem Einsatz ohne Zeitfenster lernen die Verfahren stets auf allen bekannten Trainingsdaten, also sozusagen mit „vollem Gedächtnis" (*Full Memory*). Anhand zweier simulierter Szenarien auf realen Textdaten soll untersucht werden, ob die Verwendung von Zeitfenstern gegenüber dem Lernen auf allen bekannten Daten zu einer Verbesserung der Lernergebnisse führt. Außerdem soll festgestellt werden, ob die Erkennung von Konzeptverschiebungen bei der adaptiven Fensterverwaltung funktioniert, und ob die durchgeführten Anpassungen der Fenstergröße beim adaptiven Fensterverwaltungsansatz zu einer Performanzsteigerung der Lernverfahren gegenüber den Fensterverwaltungsansätzen mit fester Fenstergröße führen.

Bei dem eingesetzten Datensatz handelt es sich um eine Teilmenge des Datensatzes der *Text REtrieval Conference (TREC)*. Dieser Datensatz enthält englischsprachige Wirtschaftsnachrichtentexte aus verschiedenen Quellen. Diese Texte sind in der Regel einer oder mehreren Kategorien zugeordnet. Für die in dieser Arbeit beschriebenen Experimente wurden die Texte aus den Kategorien 1, 3, 4, 5 und 6 verwendet. Die Tabelle 6.1 zeigt die Titel dieser Kategorien sowie die Anzahlen der in ihnen enthaltenen Dokumente. Für die in diesem Kapitel beschriebenen Experimente werden zwei verschiedene Konzeptverschiebungsverläufe simuliert. Hierzu werden die Daten zufällig auf zwanzig gleich große Batches verteilt, die jeweils 130 Dokumenten enthalten[1]. Dabei werden die Dokumente der einzelnen Kategorien möglichst gleichmäßig auf die 20 Batches verteilt.

Im ersten Szenario, im folgenden als *Szenario A* bezeichnet, soll ein Benutzerinteresse simuliert werden, das zuerst die Dokumente der Kategorie 1 (Antitrust Cases Pending) für relevant hält und die Texte aller übrigen Kategorien für irrelevant, und das dann abrupt wechselt (Concept Shift) und

[1]Von den insgesamt 2608 Dokumenten bleiben also in jedem Versuch 8 zufällig ausgewählte Dokumente unberücksichtigt.

Kategorie	Name der Kategorie	Anzahl zugeordneter Dokumente
1	Antitrust Cases Pending	400
3	Joint Ventures	842
4	Debt Rescheduling	355
5	Dumping Charges	483
6	Third World Debt Relief	528
	Insgesamt	2608

Tabelle 6.1: In den Experimenten verwendete Kategorien des TREC-Datensatzes und die Anzahlen der ihnen zugeordneten Dokumente.

Kate-	Relevanz der Kategorien in den einzelnen Batches																			
gorie	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tabelle 6.2: Relevanz der Kategorien im Zeitablauf für Konzeptverschiebungsszenario A (abrupte Verschiebung in Batch 10). Diese Tabelle beschreibt für Dokumente aus einer bestimmten Kategorie, mit welcher Wahrscheinlichkeit sie zu einem bestimmten Zeitpunkt (Batch) als relevant gelten.

ab dem 10. Batch die Dokumente der Kategorie 3 (Joint Ventures) für relevant hält und die der übrigen Kategorien für irrelevant. In Tabelle 6.2 ist dies über die Wahrscheinlichkeit beschrieben, daß ein Dokument aus einer bestimmten Kategorie zu einem bestimmten Zeitpunkt (Batch) für das Benutzerinteresse relevant ist. Die Klassen 4, 5 und 6 sind also zu keinem Zeitpunkt relevant.

Im zweiten Szenario, im folgenden als *Szenario B* bezeichnet, soll ebenfalls ein Benutzerinteresse simuliert werden, das zuerst die Dokumente der Kategorie 1 (Antitrust Cases Pending) für relevant hält und die Texte aller übrigen Kategorien für irrelevant und später die Texte der Kategorie 3 (Joint Ventures) für relevant und die aller übrigen Kategorien für irrelevant. In diesem Szenario soll die Interessenverschiebung allerdings schrittweise von Batch 8 bis Batch 12 erfolgen (Concept Drift). Auch hier sind die Klassen 4, 5 und 6 zu keinem Zeitpunkt relevant. Tabelle 6.3 zeigt die entsprechenden Wahrscheinlichkeiten für die Relevanz eines Dokumentes einer Kategorie zu einem bestimmten Zeitpunkt (Batch).

Zur Durchführung der Experimente wurde eine Testumgebung entworfen und implementiert, die die zeitliche Abfolge des Stroms der Dokumente sowie das jeweilige Benutzerinteresse mit seinen Verschiebungen simuliert, Schnittstellen zu den eingesetzten Lernverfahren bietet und deren Per-

Kate-	Relevanz der Kategorien in den einzelnen Batches																			
gorie	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	0.6	0.4	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.4	0.6	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tabelle 6.3: Relevanz der Kategorien im Zeitablauf für Konzeptverschiebungsszenario B (Verschiebung in kleinen Schritten von Batch 8 bis Batch 12). Diese Tabelle beschreibt für Dokumente aus einer bestimmten Kategorie, mit welcher Wahrscheinlichkeit sie zu einem bestimmten Zeitpunkt (Batch) als relevant gelten.

formanz auswertet (*Dynamical Classification Environment (DyCE)*, siehe Abschnitt 6.1). Ohne das Vorliegen einer solchen Testumgebung wäre die Evaluation der in dieser Arbeit vorgestellten Ansätze zur Fensterverwaltung nicht möglich. Im Anschluß an die Darstellung der Testumgebung werden der weitere experimentelle Aufbau (Abschnitt 6.2) sowie die erzielten Ergebnisse der Experimente für die beiden geschilderten Szenarien beschrieben (Abschnitte 6.3 und 6.4).

6.1 Die dynamische Klassifikationsumgebung DyCE

Das Vorhandensein einer Testumgebung, die es erlaubt, Dokumentenströme mit Konzeptverschiebungen zu simulieren und die verschiedenen vorgestellten Fensterverwaltungmechanismen zusammen mit verschiedenen Lernverfahren auf diese Dokumentenströmen anzuwenden, ist eine unumgängliche Voraussetzung für die im folgenden beschriebenen Experimente und für die Evaluation der verschiedenen Ansätze zur Fensterverwaltung. Deswegen wurde im Rahmen dieser Arbeit eine dynamische Klassifikationsumgebung *(Dynamic Classification Environment (DyCE))* implementiert, die diesen Anforderungen gerecht wird. In diesem Abschnitt werden folgende Aspekte von DyCE kurz angesprochen:

- Datenvorverarbeitung: Textnormalisierung.
- Konfiguration: Dokumentenströme, Konzeptverschiebungsverläufe, Zeitfensterverwaltung und Lernverfahren.
- Experimentverwaltung.
- Ablauf eines Experiments.

Vor der Durchführung von Experimenten werden die Texte einmalig einer Datenvorverarbeitung unterzogen (*Textnormalisierung*, siehe auch Abschnitt 2.5). Eventuell in den Texten vorhandene SGML-Tags werden ebenso entfernt wie Interpunktions- und Sonderzeichen. Die verbleibenden

Zeichenketten (Token) werden in Kleinbuchstaben umgewandelt. Token, die nur aus Ziffern bestehen werden entfernt. In Token, die neben Ziffern auch andere Zeichen enthalten, werden alle Ziffern in Einsen transformiert. Eine Stammform- und eine Grundformreduktion erfolgen nicht.

Die *Konfiguration der Dokumentenströme und der zu simulierenden Konzeptverschiebungsverläufe* erfolgt getrennt von der Konfiguration der Lernverfahren und der Zeitfensterverwaltung. Die Dokumente für die Experimente können auf beliebig viele Verzeichnisse verteilt sein, die die Dokumente als ASCII-Dateien enthalten. Jedes solches Verzeichnis wird in DyCE als Quelle eines eigenen Datenstroms (Stream) angesehen. Für jeden simulierten Zeitablauf innerhalb eines Experimentes werden die Dokumente aus diesen Quellen zufällig, aber möglichst gleichmäßig auf die Batches verteilt, deren Anzahl der Benutzer vorgibt. Für jedes Experimente werden in der Regel mehrere solcher Zeitabläufe simuliert und die Ergebnisse ihrer Auswertungen gemittelt. Für die in diesem Kapitel beschriebenen Experimente wurde über jeweils vier simulierte Zeitabläufe gemittelt. In ähnlicher Form wie in den Tabellen 6.2 und 6.3 kann der Benutzer spezifieren, mit welcher Wahrscheinlichkeit ein Dokument aus einem bestimmten Dokumentenstrom zu einem bestimmten Zeitpunkt (Batch) relevant ist. Gemäß diesen Wahrscheinlichkeiten erfolgt dann während der Simulation des Zeitablaufs die Zuweisung der Dokumente zu den Klassen relevant bzw. nicht relevant.

Verbunden mit der *Konfiguration der Lernverfahren und der Zeitfensterverwaltung* ist eine *Experimentverwaltung*, die es erlaubt, Serien von Experimenten automatisch ablaufen zu lassen. Für jedes in DyCE verfügbare Lernverfahren können für jeden seiner Parameter mehrere zu testende Werte angegeben werden, die dann in allen Kombinationen mit den Werten der anderen Parameter des Verfahrens getestet werden. Gleiches gilt für die zur Verfügung stehenden Formen der Trainingsdatenverwaltung, die neben dem Lernen auf allen bekannten Daten das Lernen mit fester oder adaptiver Fenstergröße umfassen. Bei entsprechender Konfiguration kann man also alle Kombinationen von bestimmten Datenverwaltungsformen, Lernverfahren und Parametern dieser Verfahren automatisiert testen.

Beim *Ablauf eines Experiments*, d. h. der Anwendung eines Lernverfahrens mit einem Parametersatz und einer Datenverwaltungsform, werden für jeden Zeitpunkt (Batch) jedes simulierten Zeitablaufs folgende Schritte durchlaufen:

- Einrichtung der aktuellen Trainings- und Testmenge für den aktuellen Batch gemäß der gewählten Form der Datenverwaltung.
- Erzeugen der Konfigurationsdatei für das gewählte Lernverfahren und Aufruf der Textklassifikationsumgebung *TCat* [Joachims, 1996] für den Lernschritt auf der Trainingsmenge und die Klassifikation der Testmenge.
- Vergleich der vorhergesagten und der „echten“ Klassifikation der Testdokumente und Berechnung der Performanzmaße für den aktuellen Batch.

Die Ergebnisse werden anschließend über alle simulierten Zeitabläufe eines Experimentes gemittelt, so daß man Durchschnittswerte für jeden Batch und gemittelt über alle Batches erhält.

Die Textkategorisierungsumgebung TCat umfaßt eine Reihe von Lernverfahren für die Textklassifikation bei statischen Konzepten, unter anderem alle hier verwendeten Lernverfahren (siehe Kapitel 3 bzw. den nächsten Abschnitt). Das Verfahren CN2 (Abschnitt 3.7) und die hier eingesetzte Variante des Rocchio-Algorithmus mit automatischer Schwellwert-Ermittlung durch v-Fold-Cross-Validation (Abschnitt 3.1) wurden im Rahmen dieser Arbeit in TCat aufgenommen.

6.2 Aufbau der Experimente

Die hier beschriebenen Experimente erfolgen nach dem Prinzip des Batch-Lernens, d. h. die Lernverfahren erzeugen bei Erhalt eines neuen Batches von Trainingsdaten stets ein neues Klassifikationsmodell. Folgenden *Datenverwaltungsansätze* werden in allen Kombinationen mit den anschließend aufgelisteten Lernverfahren eingesetzt:

- *Lernen mit „vollem Gedächtnis“*, im folgenden als *„Full Memory“*-Ansatz bezeichnet: Das Lernverfahren lernt sein Klassikationsmodell stets auf allen bekannten Beispielen aus der Vergangenheit, kann alte Beispiele also nicht „vergessen“.

- *Lernen „ohne Gedächtnis“*, im folgenden als *„No Memory“*-Ansatz bezeichnet: Das Lernverfahren lernt sein Klassifikationsmodell immer nur auf dem zuletzt gesehenen Batch. Dies entspricht der Verwendung eines Zeitfensters mit der festen Größe eines Batches.

- *Lernen mit einem Fenster fester Größe*: Hier wird ein Fenster der Größe von drei Batches verwendet. Dieser Ansatz wird im folgenden als *„Fixed Size“*-Ansatz bezeichnet.

- *Lernen mit adaptiver Fenstergröße*: Bei diesem Ansatz wird das in Kapitel 5 entworfene Verfahren zur automatischen Anpassungen der Fenstergröße an die aktuelle Situation eingesetzt. Diese Variante der Datenverwaltung wird hier als *„Adaptive Size“*-Ansatz bezeichnet.

Bei der adaptiven Fensterverwaltung wird die initiale Fenstergröße auf die Größe dreier Batches gesetzt ($|W_0| := 3 \cdot |B|$) und die Anzahl der initialen Batches zur Etablierung eines stabilen Mittelwertes und Standardfehlers für die beobachteten Indikatoren auf fünf gesetzt ($M_0 := 5$). Die Mittelwertbildung für die Kontrolle der Indikatorwerte mittelt hier über die Indikatorwerte der letzten $M := 10$ Batches (falls bereits soviele Batches vorliegen). Das Konfidenzniveau für die Erkennung von Konzeptverschiebungen wird auf $\alpha = 5.0$ gesetzt, der Parameter β wird auf 0.5 gesetzt und die Fensterreduktionsrate γ

wird ebenfalls auf den Wert 0.5 gesetzt. Diese Werte sind relativ willkürlich gesetzt und nicht durch Vorexperimente bestimmt oder in irgendeiner anderen Weise optimiert worden. Der hohe Wert von α soll bewirken, daß Konzeptverschiebungen nur bei entsprechend hoher Konfidenz vorhergesagt werden.

Die in der folgenden Liste der eingesetzten *Lernverfahren* angegebenen Parameterwerte wurden in Vorexperimenten mit einer anderen Klassifikationsaufgabe auf den TREC-Daten als günstig befunden und für die hier beschriebenen Experimente nicht weiter optimiert:

- *Rocchio-Algorithmus*: Es wird $\alpha := 1.0$ und $\beta := 1.0$ benutzt (siehe Abschnitt 3.1). Hier wird die Variante des Algorithmus angewandt, die über v-fold-Cross-Validation einen möglichst guten Schwellwert zu finden versucht. Es wird $v := 4$ verwendet. Alle Attribute werden zum Lernen herangezogen, d. h. es erfolgt keine Attributauswahl.

- *Naiver Bayescher Klassifikator* aus Abschnitt 3.2: Alle Attribute werden zum Lernen herangezogen.

- *PrTFIDF-Algorithmus* (Abschnitt 3.3): Alle Attribute werden zum Lernen herangezogen.

- *k-Nächste-Nachbarn-Verfahren* (Abschnitt 3.4): Hier wird ein distanzgewichterer Klassifikator mit $k := 5$ eingesetzt. Alle Attribute werden zum Lernen herangezogen.

- *Winnow-Algorithmus* (Abschnitt 3.5): Hier wird bei einer Lernrate von $\gamma := 1.1$ über 40 Iterationen gelernt. Es wird auf den jeweils 1000 besten Attributen gemäß des Information Gain Kriteriums (Abschnitt 2.6) gelernt.

- *Support-Vektor-Maschine (SVM)* (Abschnitt 3.6): Hier wird eine SVM mit polynomiellem Kernel und einem Polynomgrad von Eins eingesetzt. Alle Attribute werden zum Lernen herangezogen.

- *CN2* (Abschnitt 3.7): CN2 wird auf den gemäß Information Gain besten 1000 Attributen mit $MaxStar := 10$ zur Erzeugung ungeordneter Regelmengen eingesetzt. Das Konfidenzniveau wird auf den Default-Wert Null gesetzt.

- *C4.5* (Abschnitt 3.8): C4.5 wird hier mit den Default-Parametern [Quinlan, 1993] zur Erzeugung geordneter Regelmengen eingesetzt, die wie in Abschnitt 3.8 beschrieben durch Pruning vereinfacht werden. Es wird auf den jeweils 1000 besten Attributen gemäß des Information Gain Kriteriums gelernt.

In den Experimenten, deren Ergebnisse in den nächsten beiden Abschnitten beschrieben werden, werden alle Kombinationen aus den hier beschriebenen Lernverfahren und Datenverwaltungsansätzen auf die beiden in der Einleitung dieses Kapitels beschriebenen Konzeptverschiebungsszenarien angewandt.

Die Ergebnisse werden jeweils über vier Läufe gemittelt. Da bei Erhalt des ersten Bachtes (Batch 0) noch kein Klassifikationsmodell vorliegt, werden nur Ergebnisse für die Batches 1 bis 19 angegeben.

6.3 Ergebnisse für Szenario A (Concept Shift)

Die in diesem Abschnitt beschriebenen Ergebnisse beziehen sich auf das Konzeptverschiebungsszenario A (siehe Tabelle 6.2), bei dem in Batch 10 eine abrupte Konzeptverschiebung (Concept Shift) von TREC-Kategorie 1 zu TREC-Kategorie 3 auftritt. Tabelle 6.4 vergleicht alle Kombinationen von Lernverfahren und Datenverwaltungsansätzen anhand der Performanzmaße Accuracy, Recall und Precision. Die Ergebnisse sind über vier Versuche mit jeweils 20 Batches gemittelt.

Neben den absoluten Ergebnissen vergleicht die Tabelle in ihren drei rechten Spalten jeweils ein Paar von Verfahren. Die Spalte „(2) vs. (1)“ gibt den Vorteil von Datenverwaltungsansatz (2) (No Memory) gegenüber dem Ansatz (1) (Full Memory) an, also die Differenz der Performanzmaße dieser beiden Verfahren. In entsprechender Weise vergleichen die beiden letzten Spalten den Ansatz Adaptive Size mit den beiden Ansätzen mit fester Fenstergröße, also mit No Memory und Fixed Size.

Aus der Spalte „(2) vs. (1)“ läßt sich ablesen, daß es für alle eingesetzten Lernverfahren deutlich besser ist, den einfachen Fensteransatz No Memory einzusetzen als auf allen bekannten Beispielen zu lernen (Full Memory). Die durchschnittliche Performanzsteigerung beträgt bei der Accuracy 11.78%, bei Recall und Precision, den in der Textklassifikation meist wichtigeren Maßen sogar 25.76% bzw. 23.63%. Also führt bei diesem Szenario bereits der Einsatz einer einfachen Fensterverwaltungen zu einer erheblichen Performanzsteigerung.

Nun stellt sich die Frage, ob man diese Performanzsteigerung durch den Einsatz der adaptiven Fensterverwaltung noch erhöhen kann. Für das *Szenario A* zeigt sich in der Tat, daß sich noch eine leichte Steigerung erreichen läßt (siehe Spalten „(4) vs. (2)“ und „(4) vs. (3)“ in der Tabelle 6.4). Gemittelt über alle Verfahren, liegt die Performanzsteigerung durch den Einsatz der adaptiven Fensterverwaltung gegenüber der besten getesteten Datenverwaltung mit Fenster fester Größe bei 1.59% in der Accuracy und deutlicheren 4.45% bzw. 2.35% bei Recall bzw. Precision. Bei Accuracy-Werten um die 90%, wie sie hier erreicht werden, stellt aber auch eine (absolute) Verbesserung um „nur“ 1.59 Prozentpunkte bei der Accuracy bereits eine relative Verringerung der Fehlerrate um über 10% dar.

Die durch den Einsatz der adaptiven Fensterverwaltung bei diesem Szenario erzielte geringe durchschnittliche Performanzsteigerung ist differenzierter zu betrachten. Während einige Verfahren wie CN2, C4.5, die SVM oder Winnow

	Full Memory (1)	No Memory (2)	Fixed Size (3)	Adaptive Size (4)	(2) vs. (1)	(4) vs. (2)	(4) vs. (3)
Rocchio							
Accuracy	75.95%	84.63%	87.93%	89.38%	+08.86%	+03.65%	+01.45%
Recall	49.77%	93.59%	87.41%	91.74%	+43.82%	–01.15%	+04.33%
Precision	48.64%	61.77%	70.43%	72.91%	+13.13%	+11.14%	+02.48%
Naive Bayes							
Accuracy	81.96%	93.59%	91.97%	93.97%	+11.63%	+00.38%	+02.00%
Recall	68.51%	86.96%	84.67%	88.18%	+18.45%	+01.22%	+03.51%
Precision	67.63%	87.32%	84.69%	87.68%	+19.69%	+00.36%	+02.99%
PrTFIDF							
Accuracy	80.67%	88.18%	87.44%	88.87%	+07.51%	+00.69%	+01.43%
Recall	85.33%	93.49%	93.31%	94.19%	+08.16%	+00.70%	+00.88%
Precision	56.78%	67.66%	66.21%	64.42%	+10.88%	–03.24%	–01.79%
k-NN							
Accuracy	79.32%	91.26%	90.14%	92.33%	+11.94%	+01.07%	+02.19%
Recall	49.82%	76.60%	74.45%	80.74%	+26.78%	+04.14%	+06.29%
Precision	63.34%	87.14%	84.29%	87.02%	+23.80%	–00.12%	+02.73%
Winnow							
Accuracy	74.48%	89.94%	89.15%	91.64%	+15.46%	+01.70%	+02.49%
Recall	41.44%	70.09%	70.77%	78.33%	+28.65%	+08.24%	+07.56%
Precision	48.46%	83.12%	82.03%	85.95%	+34.66%	+02.83%	+03.92%
SVM							
Accuracy	79.48%	92.64%	91.80%	94.48%	+13.16%	+01.84%	+02.68%
Recall	51.03%	74.24%	77.11%	83.95%	+23.21%	+09.71%	+06.84%
Precision	64.65%	91.27%	87.32%	91.49%	+26.62%	+00.22%	+04.17%
CN2							
Accuracy	77.72%	90.50%	90.16%	92.45%	+12.78%	+01.95%	+02.29%
Recall	41.20%	68.45%	69.68%	76.74%	+27.25%	+08.29%	+07.06%
Precision	56.49%	85.89%	85.37%	89.56%	+29.40%	+03.67%	+04.19%
C4.5							
Accuracy	78.49%	91.40%	90.29%	92.83%	+12.91%	+01.43%	+02.54%
Recall	49.22%	79.02%	76.49%	83.47%	+29.80%	+04.45%	+06.98%
Precision	51.24%	82.10%	81.84%	86.03%	+30.86%	+03.97%	+04.19%
Mittelwert							
Accuracy					+11.78%	+01.59%	+02.13%
Recall					+25.76%	+04.45%	+05.43%
Precision					+23.63%	+02.35%	+02.86%

Tabelle 6.4: Accuracy, Recall und Precision der Lernverfahren in Kombination mit allen Datenverwaltungsansätzen gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario A.

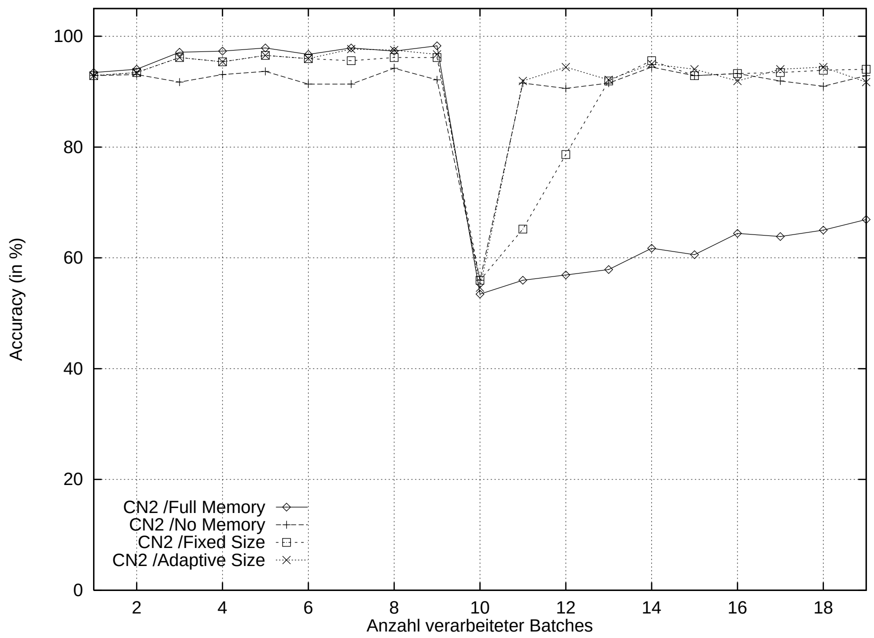

Abbildung 6.1: Accuracy des Lernverfahrens CN2 in Kombination mit den verschiedenen Datenverwaltungsansätzen gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario A.

recht deutlich von der adaptiven Fenstergröße profitieren, zeigen sich bei anderen Verfahren wie PrTFIDF oder dem naiven Bayes'schen Klassifikator keine signifikaten Verbesserungen. Bei PrTFIDF sinkt die Precision sogar um über drei Prozentpunkte.

Während Tabelle 6.4 zwar Performanzsteigerungen erkennen läßt (oder teilweise eben auch nicht), vermag sie die Frage, ob die Indikatoren der adaptiven Fensterverwaltung die Konzeptverschiebung zuverlässig erkannt haben, und ob sie sich durch Unregelmäßigkeiten in den Daten nicht haben täuschen lassen, nicht direkt beantworten. Um hier zu einer Antwort zu kommen, bietet es sich an, die Indikatoren und die Fenstergröße mit ihren Anpassungen im Zeitablauf zu betrachten. Genau diesen detailierteren Einblick bieten die Abbildungen 6.1 bis 6.4 am Beispiel des Lernverfahrens CN2 in Kombination mit allen Datenverwaltungsansätzen.

Die Abbildungen 6.1 bis 6.3 mit den Accuracy-, Recall- und Precision-Werten von CN2 mit den verschiedenen Datenverwaltungsansätzen lassen eine ganze Reihe von Dingen erkennen. Erstens eignen sich offensichtlich alle drei Performanzmaße in diesem Szenario als Indikatoren, denn sie alle zeigen einen deutlichen Performanzeinbruch beim Auftreten des Concept Shifts in Batch 10. Anhand der Abbildungen sieht man aber auch, daß Recall und Precision einen viel ausgeprägteren Einbruch aufweisen als die Accuracy. Dies erklärt sich aus der Art der Konzeptverschiebung. Die relevanten Dokumente,

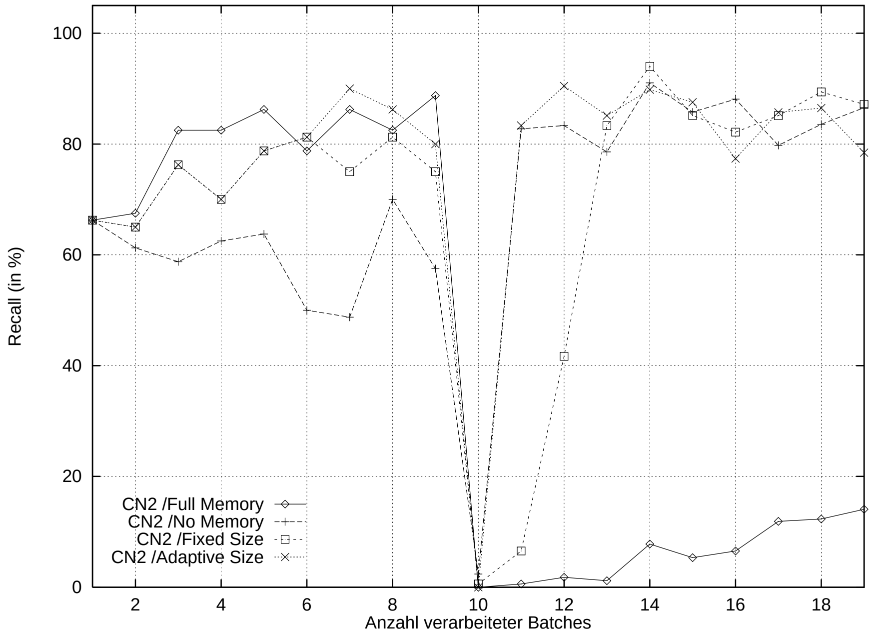

Abbildung 6.2: Recall des Lernverfahrens CN2 in Kombination mit den verschiedenen Datenverwaltungsansätzen gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario A.

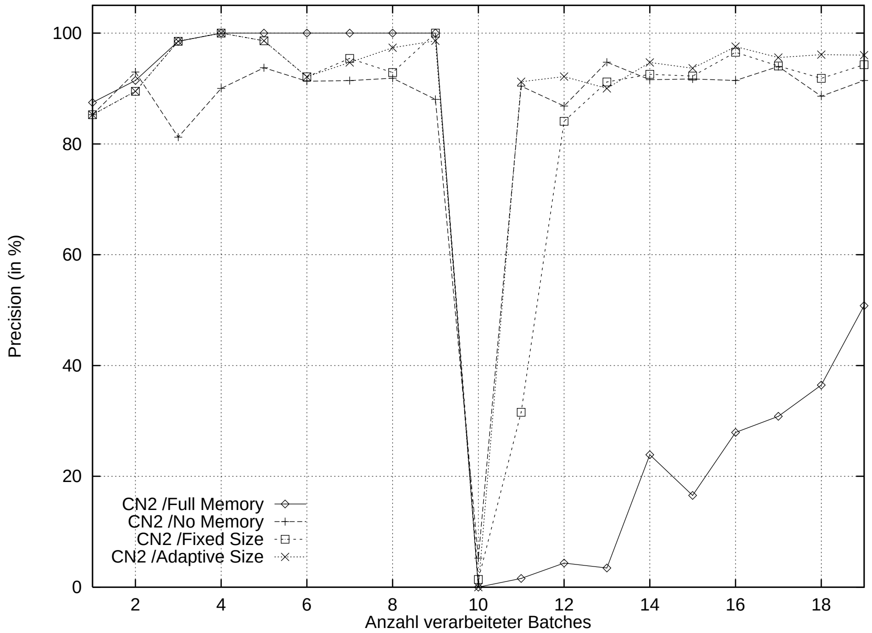

Abbildung 6.3: Precision des Lernverfahrens CN2 in Kombination mit den verschiedenen Datenverwaltungsansätzen gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario A.

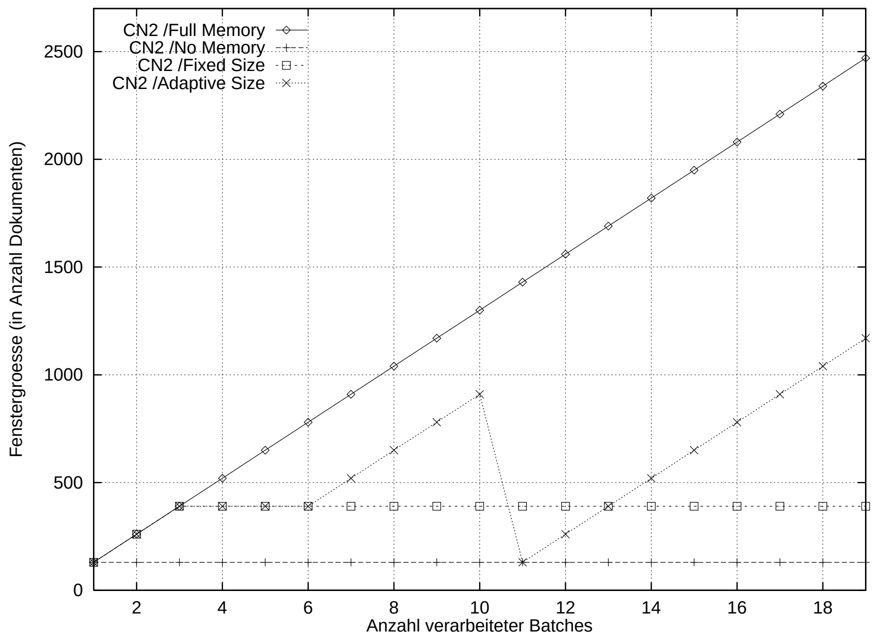

Abbildung 6.4: Fenstergröße für das Lernverfahren CN2 bei Einsatz der verschiedenen Datenverwaltungsansätze gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario A.

deren Klassifikation von Recall und Precision beobachtet wird, verändern sich bei dem Concept Shift radikal, während sich die Masse der nicht-relevanten Dokumente, die von der Accuracy ebenso erfaßt werden wie die relevanten, nicht verändert.

Eine zweite Beobachtung, die man anhand dieser Abbildungen machen kann, ist das typische Verhalten der verschiedenen Datenverwaltungsansätze im Verhältnis zueinander. Vor der Konzeptverschiebung zeigt sich der Full Memory Ansatz mit seiner großen Datenbasis als besonders leistungsstark, aber von seinem Leistungseinbruch beim Auftreten der Konzeptverschiebung erholt er sich nur äußerst langsam, weil die alte Datenbasis noch relativ lange ein relatives hohes Gewicht behält. Der Fixed Size Ansatz zeigt in Phasen mit stabilem Zielkonzept auch ein recht gutes Verhalten, benötigt im Falle einer Konzeptverschiebung aber auch mehrere Batches, um seine Leistung wieder zu erreichen. Der No Memory Ansatz bietet die maximale Flexibilität und erholt sich nach nur einem Batch vom Performanzeinbruch, der durch die Konzeptverschiebung auftritt. Allerdings zeigt er sich bei stabilem Konzept deutlich instabiler und weniger leistungsstark als die anderen Ansätze. Dem Adaptive Size Ansatz gelingt es in Verbindung mit CN2 in diesem Szenario offensichtlich, sowohl bei stabilem Konzept eine hohe Performanz und große Stabilität zu zeigen, als auch sich sehr schnell an die aufgetretene Konzeptverschiebung anzupassen.

Zusammenfassend läßt sich zu diesen drei Abbildungen also sagen, daß man klar erkennt, daß die Fensterverwaltungsansätze dem Lernen auf allen Beispielen in diesem Szenario klar überlegen sind, und daß es der adaptiven Fensterverwaltung hier gelingt, die Vorteile verschiedener Fenstergröße zu kombinieren.

Abbildung 6.4 zeigt die Fenstergröße der verschiedenen Datenverwaltungsansätze am Beispiel von CN2 im Zeitablauf. Das Fenster des Full Memory Ansatzes wächst linear in der Anzahl gesehener Beispiele. Das Fenster des No Memory Ansatzes ist stest genau einen Batch groß und das Fenster des Fixed Size Ansatzes wächst bis zu einer Größe von drei Batches und behält diese Größe dann bei. Die Fensterverwaltung des adaptiven Ansatzes zeigt hier genau ihr gewünschtes Verhalten. Das Fenster wächst zuerst bis zu seiner intialen Größe von drei Batches (Benutzervorgabe) und behält diese bis zum Ende der initialen Schwellwertermittlungsphase (fünf initiale Batches, ebenfalls eine Benutzervorgabe). Ab dem sechsten Batch wird die adaptive Fensterkontrolle dann aktiv und läßt das Fenster wachsen, da keine Konzeptverschiebung zu erkennen ist. Beim Eintreten der abrupten Konzeptverschiebung (Concept Shift) setzt die Fensterverwaltungsheuristik die Größe des Fensters auf einen Batch zurück. Da die Situation dann wieder stabil ist, wächst das Fenster direkt nach der Verschiebung wieder an.

Dieses erwünschte Verhalten der Fensteranpassung hat sich in Tabelle 6.4 in einer gegenüber den anderen Fensterverwaltungsansätzen erhöhten Performanz niedergeschlagen. Doch woran liegt es, daß sich dieser Effekt bei manchen der anderen Verfahren, wie beispielsweise PrTFIDF, nicht oder nur sehr schwach zeigt ? Sind die Indikatoren vielleicht doch nicht ausreichend, um die Konzeptverschiebung für alle Lernverfahren zuverlässig zu erkennen ?

Ein Blick auf die Abbildungen 6.5 und 6.6 zeigt, daß dies nicht der Fall ist. Zum einen zeigen die Indikatoren die Konzeptverschiebung durch einen sehr deutlichen Einbruch an, Recall und Precision wiederum stärker als die Accuracy, und zum anderen zeigt ein Blick auf die Fenstergröße von PrTFIDF zusammen mit dem Adaptive Size Ansatz, daß auch die Fenstergröße in nahezu der gewünschten Weise angepaßt wird. Also funktionieren die Indikatoren und die Fensteranpassung auch bei PrTFIDF, auch wenn sich dies nicht in einer Performanzsteigerung niederschlägt.

6.4 Ergebnisse für Szenario B (Concept Drift)

Die in diesem Abschnitt beschriebenen Ergebnisse beziehen sich auf das Konzeptverschiebungsszenario B (siehe Tabelle 6.3), bei dem von Batch 8 bis Batch 12 eine schrittweise Konzeptverschiebung (Concept Drift) von TREC-Kategorie 1 zu TREC-Kategorie 3 auftritt. Tabelle 6.4 vergleicht alle Kombinationen von Lernverfahren und Datenverwaltungsansätzen anhand der

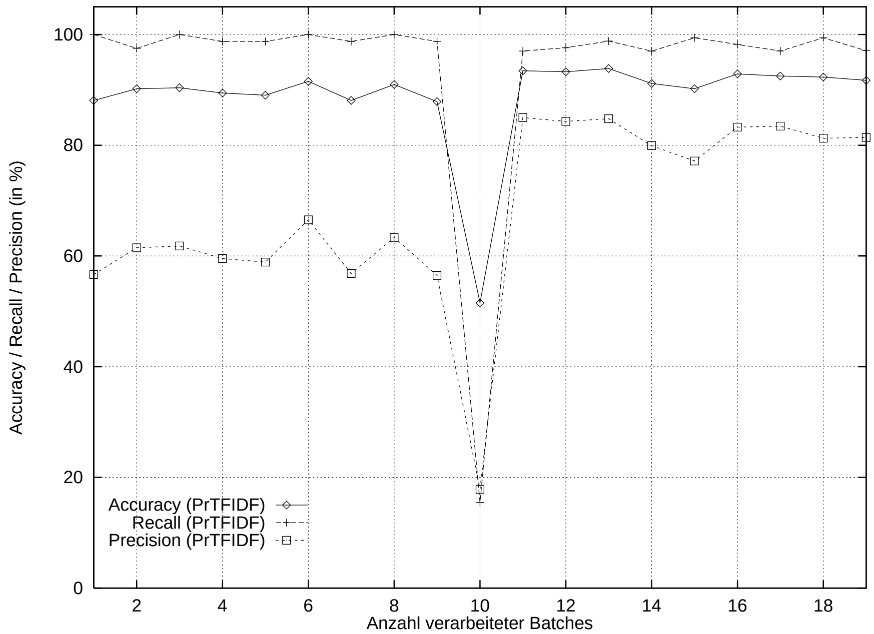

Abbildung 6.5: Die Performanzmaße Accuracy, Recall und Precision für den Adaptive Size Fensterverwaltungsansatz in Kombination mit dem Lernverfahren PrTFIDF gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario A.

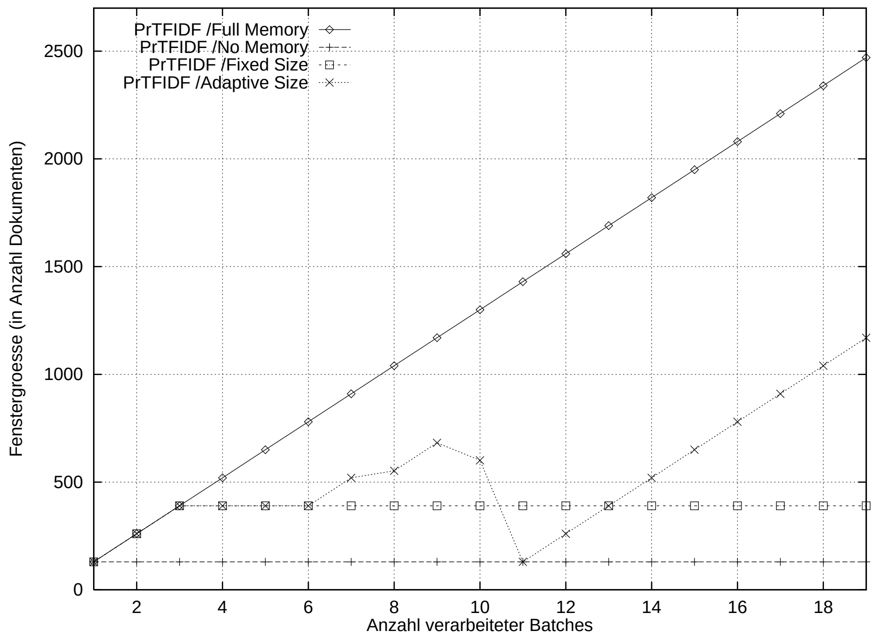

Abbildung 6.6: Fenstergröße für das Lernverfahren PrTFIDF bei Einsatz der verschiedenen Datenverwaltungsansätze gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario A.

Performanzmaße Accuracy, Recall und Precision. Die Ergebnisse sind wieder über vier Versuche mit jeweils 20 Batches gemittelt.

Wie in Szenario A zeigen auch in Szenario B alle Lernverfahren eine deutliche Performanzsteigerung, wenn sie den einfachen Fensteransatz No Memory anstelle der Full Memory Methode, also dem Lernen auf allen bekannten Beispielen, einsetzen (siehe Spalte „(2) vs. (1)" in Tabelle 6.5). Die Accuracy verbessert sich im Durchschnitt um über 10%, der Recall um über 22% und die Precision um über 21%.

Die durchschnittliche Performanzsteigerung, die sich durch den Einsatz der adaptiven Fensterverwaltung anstelle des besten getesteten Fensters fester Größe ergibt, ist hier allerdings geringer als in Szenario A (siehe die Spalten „(4) vs. (2)" und „(4) vs. (3)" in Tabelle 6.5). Die Accuracy erhöht sich um durschnittlich 0.34%, während Recall und Precision im Durchschnitt 1.34% bzw. 0.87% zulegen. Wenn man die einzelnen Verfahren differenziert betrachtet, so zeigen einige von ihnen kleine Verschlechterungen oder kleine Verbesserungen, die nicht signifikant erscheinen (siehe z.B. PrTFIDF). Bei anderen scheint sich der Einsatz der Fensteranpassung aber auch in diesem Szenario bezahlt zu machen. So verzeichnen beispielsweise CN2, C4.5 und Rocchio auch in diesem Szenario eine erkennbare Performanzsteigerung durch den Einsatz der adaptiven Fensterverwaltung.

Wenn auch für die meisten Verfahren die Hoffnung auf eine große Performanzsteigerung durch den Einsatz der adaptiven Fensterverwaltung für Szenario B nicht bestätigt wird, so zeigt sich bei näherer Betrachtung der Performanz und der Fenstergröße von CN2 in Kombination mit den verschiedenen Datenverwaltungsansätzen im Zeitablauf, daß die Indikatoren zur Erkennung von Konzeptverschiebungen relativ gut funktionieren. Dabei zeigen auch hier Recall (Abbildung 6.8) und Precision (Abbildung 6.9) die Verschiebung viel deutlicher an als die Accuracy (Abbildung 6.7).

Auch die Adaptive Fensterverwaltung arbeitet wieder recht gut mit CN2 zusammen, wie neben der Performanz von CN2 auch seine in Abbildung 6.10 im Zeitverlauf dargestellte Fenstergröße zeigt. Der Concept Drift wird bereits in Batch 9 als solcher erkannt und die Fenstergröße entsprechend reduziert. Die Reduzierung der Fenstergröße hält bis zum Batch 11 bzw. 12 an[2], bei dem die Verschiebung abgeschlossen ist. An der Tatsache, daß die Fenstergröße nicht in einem Schritt auf die Größe eines Batches reduziert wird, sieht man, daß der Concept Drift nicht fälschlicherweise als Concept Shift erkannt wurde.

Wie bereits in Szenario A, gehört PrTFIDF zu den Verfahren, die keinen nachweisbaren Nutzen aus der Adaptivität des Zeitfensters zu ziehen scheinen

[2]Da die Fenstergröße zwischen Batch 11 und 12 im Durchschnitt fast unverändert bleibt, steigt sie offensichtlich bereits in einem Teil der vier simulierten Zeitabläufe in Batch 12 wieder an, während sie bei den übrigen noch reduziert wird, denn die Fensteranpassungsheuristik kennt kein Festhalten der Fenstergröße, sondern nur Vergrößerungen und Verkleinerungen.

	Full Memory (1)	No Memory (2)	Fixed Size (3)	Adaptive Size (4)	(2) vs. (1)	(4) vs. (2)	(4) vs. (3)
Rocchio							
Accuracy	76.28%	84.44%	87.70%	89.23%	+08.16%	+04.79%	+01.53%
Recall	47.64%	89.27%	83.35%	86.12%	+41.63%	−03.15%	+02.77%
Precision	50.89%	62.01%	70.61%	73.92%	+11.12%	+11.91%	+03.31%
Naive Bayes							
Accuracy	82.60%	92.45%	92.29%	92.73%	+09.85%	+00.28%	+00.44%
Recall	66.78%	82.93%	82.94%	84.87%	+16.15%	+01.94%	+01.93%
Precision	68.61%	86.32%	85.10%	85.93%	+17.71%	−00.39%	+00.83%
PrTFIDF							
Accuracy	80.94%	87.11%	87.02%	86.63%	+06.17%	−00.58%	−00.39%
Recall	83.98%	93.84%	92.74%	94.54%	+09.86%	+00.70%	−01.80%
Precision	57.18%	65.93%	65.69%	64.99%	+08.75%	−00.94%	−00.70%
k-NN							
Accuracy	79.10%	90.51%	90.03%	89.92%	+11.41%	−00.59%	−00.41%
Recall	47.90%	71.99%	72.23%	72.13%	+24.09%	+00.14%	−00.10%
Precision	61.85%	85.73%	84.20%	84.43%	+23.88%	−01.30%	+00.23%
Winnow							
Accuracy	74.48%	87.14%	87.43%	87.29%	+12.66%	+00.15%	−00.14%
Recall	37.88%	61.02%	64.40%	65.50%	+23.14%	+04.48%	+01.10%
Precision	47.89%	77.31%	77.32%	77.83%	+29.42%	+00.52%	+00.50%
SVM							
Accuracy	78.89%	91.04%	91.43%	91.97%	+12.15%	+00.83%	+00.54%
Recall	50.14%	67.41%	73.35%	74.93%	+17.27%	+07.52%	+01.58%
Precision	65.10%	88.84%	86.62%	88.11%	+23.74%	−00.73%	+01.49%
CN2							
Accuracy	77.85%	88.63%	88.92%	89.39%	+10.78%	+01.65%	+00.47%
Recall	38.86%	61.83%	62.98%	65.34%	+22.97%	+03.51%	+02.36%
Precision	56.03%	81.89%	85.24%	85.49%	+25.86%	+03.60%	+00.25%
C4.5							
Accuracy	78.42%	88.63%	89.07%	89.75%	+10.21%	+01.12%	+00.68%
Recall	43.97%	68.46%	68.56%	71.45%	+24.49%	+02.99%	+02.89%
Precision	49.84%	78.73%	80.50%	81.53%	+28.89%	+02.80%	+01.03%
Mittelwert							
Accuracy					+10.17%	+00.96%	+00.34%
Recall					+22.51%	+02.26%	+01.34%
Precision					+21.17%	+01.93%	+00.87%

Tabelle 6.5: Accuracy, Recall und Precision der Lernverfahren in Kombination mit allen Datenverwaltungsansätzen gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario B.

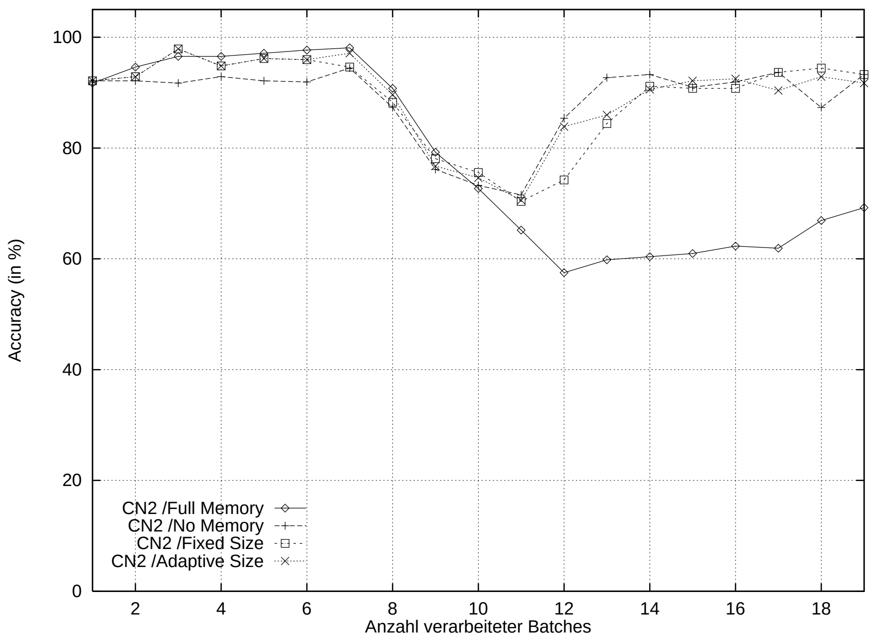

Abbildung 6.7: Accuracy des Lernverfahrens CN2 in Kombination mit den verschiedenen Datenverwaltungsansätzen gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario B.

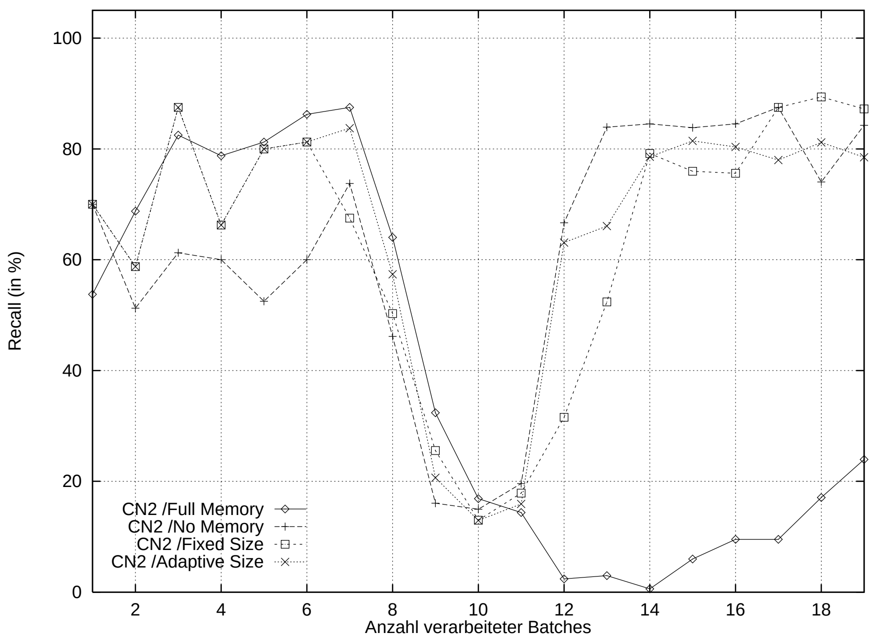

Abbildung 6.8: Recall des Lernverfahrens CN2 in Kombination mit den verschiedenen Datenverwaltungsansätzen gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario B.

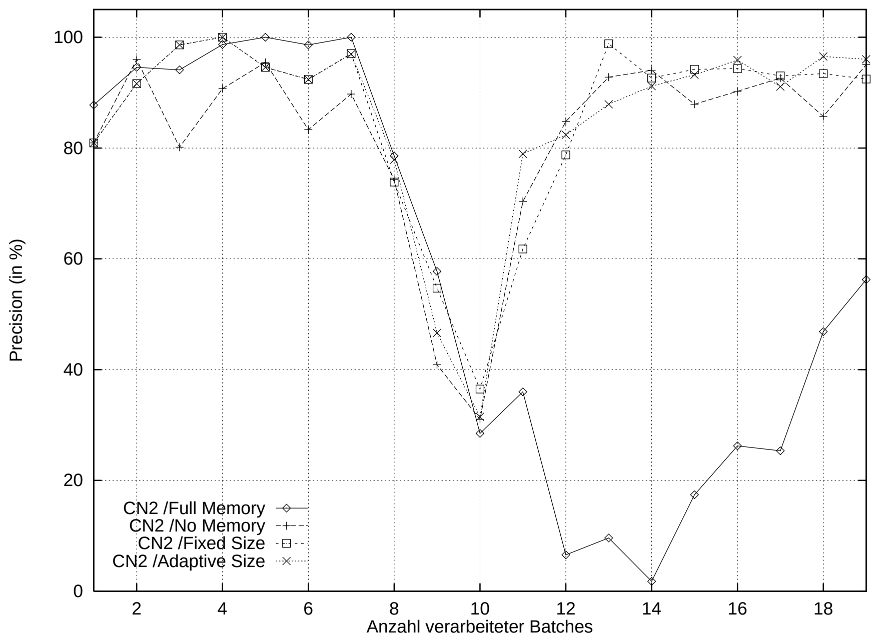

Abbildung 6.9: Precision des Lernverfahrens CN2 in Kombination mit den verschiedenen Datenverwaltungsansätzen gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario B.

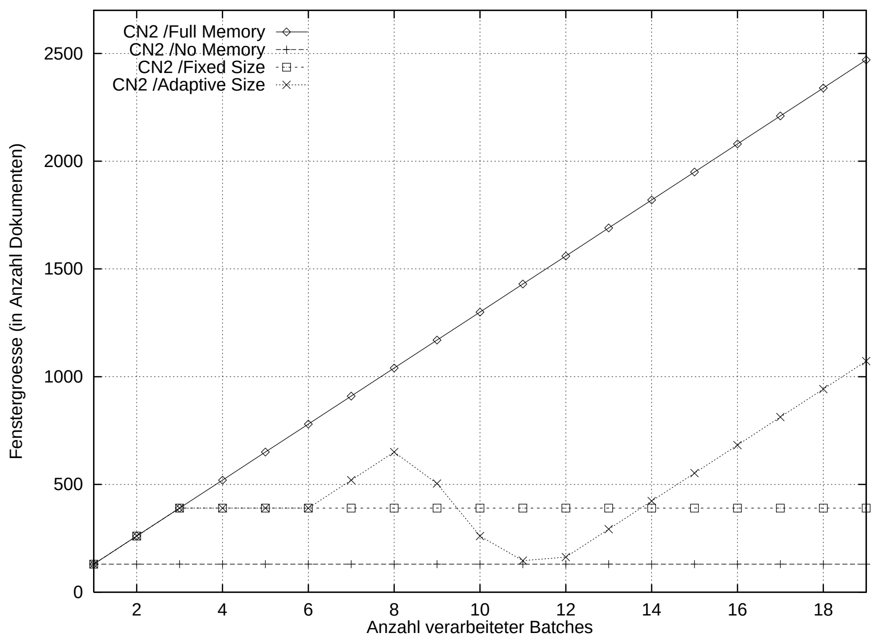

Abbildung 6.10: Fenstergröße für das Lernverfahren CN2 bei Einsatz der verschiedenen Datenverwaltungsansätze gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario B.

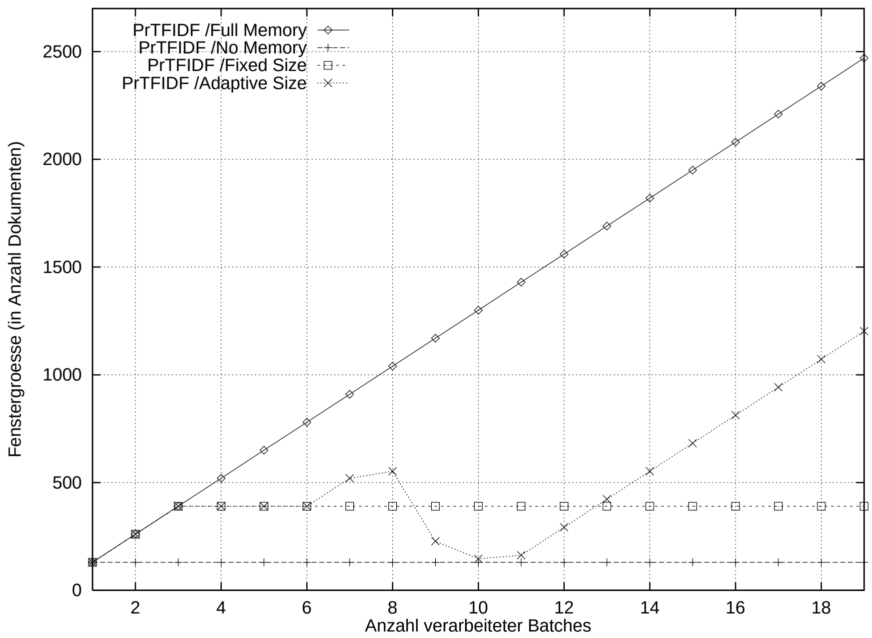

Abbildung 6.11: Fenstergröße für das Lernverfahren PrTFIDF bei Einsatz der verschiedenen Datenverwaltungsansätze gemittelt über vier Versuche mit jeweils 20 Batches gemäß Szenario B.

(siehe Tabelle 6.5). Doch wie auch schon bei Szenario A liegt dies nicht daran, daß die adaptive Fensterverwaltung bei PrTFIDF nicht wie gewünscht funktioniert, denn wie Abbildung 6.11 zeigt, funktioniert sie annähernd so wie bei CN2, das mit Hilfe der adaptiven Fensterverwaltung eine erkennbare Verbesserung seiner Performanz erzielen kann. Die Fensterverwaltung paßt sich also bei beiden Verfahren wie gewünscht der aufgetretenen Konzeptverschiebung an, aber nur eines der beiden Verfahren erreicht dadurch wie erhofft eine Performanzsteigerung.

Kapitel 7

Ergebnisse und Schlußfolgerungen

In diesem Kapitel werden noch einmal die wesentlichen Ergebnisse dieser Arbeit zusammengefaßt, die in der Aufgabenstellung in Abschnitt 1.1 formulierten Fragen anhand dieser Ergebnisse beantwortet und einige Schlußfolgerungen gezogen. In der Einleitung zu dieser Arbeit wurden das Erkennen von Konzeptveränderungen und eine geeignete Datenverwaltung als zentrale Voraussetzungen eines adaptiven Textklassifikators zum Lernen bei Informationsfilteraufgaben mit sich verändernden Konzepten identifiziert. Das Erkennen des Zeitpunktes und der Ausprägung einer Konzeptverschiebung ist für die Entscheidung erforderlich, wann das Klassifikationsmodell aktualisiert oder neu gelernt werden muß. Die Datenverwaltung sollte für das aktuelle Zielkonzept nicht länger repräsentative Dokumente aus der Trainingsmenge entfernen und dennoch eine Trainingsmenge günstiger Größe behalten können.

In dieser Arbeit wurden Indikatoren zur Erkennung von Konzeptverschiebungen bei Textklassifikationsproblem mit sich verändernden Konzepten untersucht und in einer adaptiven Zeitfensterverwaltung für derartige Textklassifikationssysteme eingesetzt. Anhand von Experimenten auf realen Textdaten wurde dieser Fensterverwaltungsansatz mit anderen Datenverwaltungsansätzen verglichen und die Funktionsfähigkeit der Indikatoren zur Erkennung von Konzeptverschiebungen untersucht.

Die Fragen aus der Aufgabenstellung lassen sich anhand der hier beschriebenen Experimente und Überlegungen wie folgt beantworten:

1. *Werden Konzeptverschiebungen mit den gewählten Kriterien erkannt ?*
   Es wurde in den Experimenten zu dieser Arbeit gezeigt, daß sich die gewählten Indikatoren Accuracy, Recall und Precision sehr gut zur zuverlässigen Erkennung von Konzeptverschiebungen eignen. Insbesondere die Indikatoren Recall und Precision zeigen in ihrem zeitlichen Verlauf sehr deutlich, wenn sich eine Verschiebung im Benutzerinteresse ergibt.

2. *Führt die Verwendung von Zeitfenstern auf den Daten bei Textklassifikationsproblemen mit sich verändernden Konzepten zu einer Performanzsteigerung gegenüber dem Lernen auf allen bekannten Beispielen ?*
   Die Experimente zu dieser Arbeit haben gezeigt, daß sich bereits durch den Einsatz einer sehr einfachen Zeitfensterverwaltung sehr deutliche Performanzsteigerungen gegenüber dem Lernen auf allen bekannten Beispielen erreichen lassen, wenn in der Textklassifikationsanwendung Konzeptverschiebungen auftreten.

3. *Läßt sich durch den Einsatz der vorgestellten adaptiven Fensterverwaltung eine weitergehende Verbesserung erreichen als mit einfacheren Ansätzen mit Fenstern fester Größe ?*
   Diese Frage läßt sich nach den Experimenten in dieser Arbeit nicht allgemein bejahen. Für einige der eingesetzten Verfahren führte der Einsatz der adaptiven Fensterverwaltung zu keiner signifikanten Verbesserung bzw. verursachte sogar leichte Verschlechterungen. Bei einigen anderen Verfahren ließen sich allerdings erkennbare Performanzsteigerungen feststellen.

In Anbetracht der Tatsache, daß in dieser Arbeit nur Experimente mit zwei relativ einfachen Szenarien auf nur einem Textdatensatz durchgeführt wurden, erscheinen weitere Experimente sinnvoll, um zu untersuchen, wie zuverlässig die in dieser Arbeit eingesetzten Indikatoren auf anderen Domänen und in anderen Situationen funktionieren. Mit Hilfe weiterer Experimente ließe sich auch feststellen, ob sich mit der adaptiven Zeitfensterverwaltung für die Verfahren, für die sich diese Form der Datenverwaltung in den Experimenten zu dieser Arbeit als vorteilhaft erwiesen hat, auf anderen Domänen und in anderen Szenarien ebenfalls Performanzsteigerungen erreichen lassen.

Die hier vorgestellte adaptive Fensterverwaltung hat den Nachteil, daß sie nur auf einer Heuristik beruht und eine große Anzahl vom Benutzer einzustellender Parameter aufweist (auf deren Optimierung in dieser Arbeit verzichtet wurde). Entgegen ursprünglichen Erwartungen bei der Wahl der Indikatoren, läßt sich dieses Fensterverwaltungsverfahren auch nicht auf beliebige Lernverfahren anwenden, sondern scheint nur mit einem Teil der Verfahren gut zusammenzuarbeiten.

Aber auch, wenn die vorgestellte Zeitfensterverwaltungen in den Experimenten zu dieser Arbeit im Zusammenspiel mit einem Teil der Verfahren keine Vorteile brachte, hat sie einigen Lernverfahren doch zu erkennbaren Performanzsteigerungen verholfen, wie am Beispiel von CN2 sehr gut zu sehen war. Zudem erscheinen die in dieser Arbeit eingesetzten Indikatoren zum Erkennen von Konzeptverschiebungen durchaus vielversprechend.

Kapitel 8

Ausblick

Mit der Indentifikation zuverlässiger Indikatoren für die Erkennug von Konzeptverschiebungen bei Textklassifikationsproblemen und der Vorstellung einer auf diesen Indikatoren beruhenden adaptiven Zeitfensterverwaltung sind zwei der zentralen Voraussetzungen für die Realisierung eines adaptiven Textklassifikators gegeben. Ein solcher Klassifikator könnte die vorgestellten und vielleicht auch weitere Indikatoren dazu verwenden, sein Klassifikationsmodell möglichst immer nur dann anzupassen, wenn eine Konzeptverschiebung vorzuliegen scheint. Außerdem könnte er anhand der vermuteten Art der Konzeptverschiebung entscheiden, ob es günstiger erscheint, ein Klassifikationsmodell neu zu lernen oder das bestehende Modell zu aktualisieren.

Wenn man sich einen regelbasierten Klassifikator vorstellt, dessen Regeln für Menschen verständlich sein sollen und eventuell von diesen verifiziert werden, ist es sinnvoll, die Häufigkeit und den Umfang von Veränderungen des Klassifikationsmodells so gering wie möglich zu halten. Die zuverlässige Erkennung von Konzeptverschiebungen böte eine Möglichkeit dazu.

Die Untersuchung weiterer zuverlässiger Indikatoren für Konzeptverschiebung wie beispielsweise die Beobachtung der gemäß eines Attributselektionskriteriums besten Attribute oder die Beobachtung von Eigenschaften der als relevant klassifizierten Dokumenten wie beispielsweise die Zugehörigkeit zu bestimmten Dokument-Clustern bieten sicherlich ein interessantes Feld für weitere Untersuchungen.

Anwendungen für effiziente adaptive Textklassifikatoren gibt es bereits heute schon, z.B. beim Nachrichtenfiltern oder der Kategorisierung von E-Mail. Ein weiteres denkbares Feld ist die gezielte Verteilung von Werbung oder Information gemäß dem Push-Prinzip, bei dem der Informarionsanbieter Profile potentieller Kunde finden will. Weitere Anwendungen werden sicher hinzukommen, wenn die weltweite Informationsverbreitung und die elektronische Kommunikation weiter in dem bisherigen Maße zunehmen.

Danksagungen

Ganz herzlich bedanken möchte ich mich bei Herrn Prof. Dr. Gholamreza Nakhaeizadeh, Frau Dr. Ingrid Renz, Herrn Dr. Thomas Bayer, Herrn Georg Veltmann sowie bei den Mitarbeitern der Abteilungen F3S/E und F3M/T der Daimler-Benz AG, Forschung und Technik, Ulm, für die Unterstützung dieser Arbeit und viele wertvolle Anregungen.

Mein besonderer Dank gilt Frau Prof. Dr. Katharina Morik und Herrn Thorsten Joachims für die Betreuung meiner Diplomarbeit sowie für viele aufschlußreiche und anregende Diskussionen. Außerdem bin ich Thorsten Joachims auch dankbar für die Bereitstellung des Textkategorisierungssystems TCat für die in dieser Arbeit beschriebenen Experimente.

Weiterhin möchte ich meinen Eltern und allen, die mich in vielfältiger Weise beim Verfassen dieser Arbeit unterstützt haben, meinen herzlichen Dank aussprechen.

Literaturverzeichnis

[Allan, 1996] Allan, J. (1996). Incremental Relevance Feedback for Information Filtering. In *Proceedings of SIGIR '96*, Zürich, Schweiz.

[Apté und Damerau, 1994] Apté, C. und Damerau, F. (1994). Automated Learning of Decision Rules for Text Categorization. *ACM Transactions on Information Systems*, 12(3):233–251.

[Balabanovic, 1997] Balabanovic, M. (1997). An Adaptive Web Page Recommendation Service. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, USA.

[Balabanovic und Shoham, 1995] Balabanovic, M. und Shoham, Y. (1995). Learning Information Retrieval Agents: Experiments with Automated Web Browsing. In *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, Stanford, CA. AAAI Press.

[Balabanovic et al., 1997] Balabanovic, M., Shoham, Y., und Yun, Y. (1997). An Adaptive Agent for Automated Web Browsing. Technical Report CS-TN-97-52, Department of Computer Science, Stanford University, Stanford, CA, USA.

[Blum, 1997] Blum, A. (1997). Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain. *Machine Learning*, 26:5–23.

[Blumer et al., 1989] Blumer, A., Ehrenfeucht, A., Haussler, D., und Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis Dimension. *Journal of the ACM*, 36(4):929–965.

[Cheeseman et al., 1988] Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., und Freeman, D. (1988). AutoClass: A Bayesian Classification System. In *Proceedings of the Fifth International Conference on Machine Learning (ICML)*, Seiten 54–56.

[Clark und Boswell, 1991] Clark, P. und Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. In Kodratoff, Y., Hrsg., *Machine Learning – Proceedings of the Fifth European Conference (EWSL '91)*, Seiten 151–163, Berlin, Germany. Springer.

[Clark und Niblett, 1989] Clark, P. und Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning*, 3(4):261–283. Kluwer, Netherlands.

[Cohen, 1995a] Cohen, W. (1995a). Learning to Classify English Text with ILP Methods. In *Advances in Inductive Logic Programming*. IOS Press.

[Cohen, 1995b] Cohen, W. (1995b). Text Categorization and Relational Learning. In *International Conference on Machine Learning (ICML)*, Seiten 124–132, Lake Tahoe.

[Cohen, 1996] Cohen, W. W. (1996). Learning Rules that Classify E-Mail. In *Proceedings of the 1996 AAAI Spring Symposium on Machine Learning in Information Access (MLIA '96)*, Stanford, CA.

[Cortes und Vapnik, 1995] Cortes, C. und Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20:273–297.

[Deerwester et al., 1990] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., und Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407.

[Fuhr, 1989] Fuhr, N. (1989). Models for Retrieval with Probabilistic Indexing. *Information Processing and Management*, 25(1):55–72.

[Fuhr et al., 1991] Fuhr, N., Hartmann, S., Lustig, G., Schwantner, M., Tzeras, K., und Knorz, G. (1991). AIR/X - a Rule-Based Multistage Indexing System for Large Subject Fields. In *RIAO*, Seiten 606–623.

[Fuhr und Knorz, 1984] Fuhr, N. und Knorz, G. (1984). Retrieval Test Evaluation of a Rule Based Automatic Indexing (AIR/PHYS). In van Rijsbergen, C., Hrsg., *Research and Development in Information Retrieval: Proceedings of the Third Joint BCS and ACM Symposium*, Seiten 391–408. Cambridge University Press.

[Graf, 1997] Graf, I. (1997). Adaption von Klassifikatoren mit Support-Vektoren. Diplomarbeit, Abteilung Informationstechnik, Universität Ulm.

[Helmbold und Long, 1991] Helmbold, D. P. und Long, P. M. (1991). Tracking Drifting Concepts Using Random Examples. In Valiant, L. und Warmuth, M., Hrsg., *Proceedings of the Fourth Annual Workshop on Computational Learning Theory (COLT '91), Santa Cruz, CA, USA*, Seiten 13–23, San Mateo, CA, USA. Morgan Kaufmann Publishers.

[Helmbold und Long, 1994] Helmbold, D. P. und Long, P. M. (1994). Tracking Drifting Concepts By Minimizing Disagreements. *Machine Learning*, 14:27–45.

[Joachims, 1996] Joachims, T. (1996). Einsatz eines intelligenten, lernenden Agenten für das World Wide Web. Diplomarbeit, Fachbereich Informatik, Universität Dortmund.

[Joachims, 1997a] Joachims, T. (1997a). A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In *Proceedings of the 1997 International Conference on Machine Learning (ICML '97)*.

[Joachims, 1997b] Joachims, T. (1997b). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. Technical Report 23, Universität Dortmund, LS VIII.

[Joachims et al., 1997] Joachims, T., Freitag, D., und Mitchell, T. (1997). WebWatcher: A Tour Guide for the World Wide Web. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

[Kalbfleish, 1979] Kalbfleish, J. (1979). *Probability and Statistical Inference*, Jgg. 2. Springer, New York, NY, USA.

[Klinkenberg, 1996] Klinkenberg, R. (1996). Rule Set Quality Measures For Inductive Learning Algorithms. Masters thesis, Department of Computer Science, University of Missouri-Rolla, Rolla, MO, USA.

[Klinkenberg und St. Clair, 1996] Klinkenberg, R. und St. Clair, D. (1996). Rule Set Quality Measures For Inductive Learning Algorithms. In Dagli, C. H., Akay, M., Chen, C. L. P., Fernández, B. R., und Ghosh, J., Hrsg., *Intelligent Engineering Systems Through Artificial Neural Networks – Smart Engineering Systems: Neural Networks, Fuzzy Logic, and Evolutionary Programming, Proceedings of the Artificial Neural Networks In Engineering (ANNIE '96) conference*, Jgg. 6, Seiten 161–168. ASME Press, New York, USA.

[Kubat, 1989] Kubat, M. (1989). Floating Approximation in Time-Varying Knowledge Bases. *Pattern Recognition Letters*, 10:223–227.

[Kubat und Widmer, 1995] Kubat, M. und Widmer, G. (1995). Adapting to Drift in Continuous Domains. In Lavrac, N. und Wrobel, S., Hrsg., *Proceedings of the 8th European Conference on Machine Learning (ECML-95)*, Jgg. 912 aus *Lecture Notes in Artificial Intelligence*, Seiten 307–310, Berlin, Germany. Springer. (Extended abstract).

[Kuh et al., 1991] Kuh, A., Petsche, T., und Rivest, R. (1991). Learning Time-varying Concepts. In *Advances in Neural Information Processing Systems*, Jgg. 3, Seiten 183–189, San Mateo, CA, USA. Morgan Kaufmann.

[Kunisch, 1996] Kunisch, G. (1996). Anpassung und Evaluierung statistischer Lernverfahren zur Behandlung dynamischer Aspekte in Data Mining. Diplomarbeit, Fakultät für Mathematik und Wirtschaftswissenschaften, Universität Ulm, Ulm, Germany.

[Lang, 1995] Lang, K. (1995). NewsWeeder: Learning to Filter Netnews. In *Proceedings of the 1995 International Conference on Machine Learning (ICML '95)*.

[Lebowitz, 1987] Lebowitz, M. (1987). Experiments with Incremental Concept Formation: UNIMEM. *Machine Learning*, 2:103–138.

[Lewis und Ringuette, 1994] Lewis, D. und Ringuette, M. (1994). A Comparison of Two Learning Algorithms for Text Classification. In *Third Annual Symposium on Document Analysis and Information Retrieval*, Seiten 81–93.

[Lewis, 1992] Lewis, D. D. (1992). An Evaluation of Phrasal and Clustered Representations on a Text Categorization Task. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seiten 37–50.

[Littlestone, 1988] Littlestone, N. (1988). Learning Quickly When Irrelevant Attributes Abound: A New Linear-treshold Algorithm. *Machine Learning*, 2:285–318. Kluwer Academic Publishers, Boston, MA, USA.

[Littlestone, 1991] Littlestone, N. (1991). Redundant Noisy Attributes, Attribute Errors, and Linear-treshold Learning Using Winnow. In Valiant, L. und Warmuth, M., Hrsg., *Proceedings of the Fourth Annual Workshop on Computational Learning Theory (COLT '91), Santa Cruz, CA, USA*, Seiten 147–156, San Mateo, CA, USA. Morgan Kaufmann Publishers.

[Littlestone und Warmuth, 1994] Littlestone, N. und Warmuth, M. K. (1994). The Weighted Majority Algorithm. *Information and Computation*, 108(2):212–261.

[Michalski, 1983] Michalski, R. S. (1983). A Theory and Methodology of Inductive Learning. In Michalski, R. S., Carbonell, J. G., und Mitchell, T. M., Hrsg., *Machine Learning — An Artificial Intelligence Approach*, Jgg. 1, Kapitel 4, Seiten 83–135. Morgan Kaufmann, Palo Alto, CA.

[Michalski et al., 1986] Michalski, R. S., Mozetic, I., J., H., und Lavrac, N. (1986). The Multipurpose Incremental Learning System AQ15 And Its Testing Application To Three Medical Domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Jgg. 2, Seiten 1041–1045. Morgan Kaufmann.

[Michie et al., 1994] Michie, D., Spiegelhalter, D. J., und Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, New York u.a.

[Mitchell et al., 1994] Mitchell, T., Caruana, R., Freitag, D., McDermott, J., und Zabowski, D. (1994). Experience with a Learning Personal Assistant. *Communications of the ACM (CACM)*, 37(7):81–91.

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

[Morik, 1997] Morik, K. (1997). *Maschinelles Lernen, Skript zur Vorlesung im Wintersemester 1997/98.* Lehrstuhl VIII, Fachbereich Informatik, Universersität Dortmund.

[Moulinier und Ganascia, 1996] Moulinier, I. und Ganascia, J. (1996). Applying an Existing Machine Learning Algorithm to Text Categorization. In Wermter, S., Riloff, E., und Scheler, G., Hrsg., *Connectionist, statistical, and* *symbolic approaches to learning for natural language processing*. Springer-Verlag.

[Moulinier et al., 1996] Moulinier, I., Raskinis, G., und Ganascia, J. (1996). Text Categorization: A Symbolic Approach. In *Annual Symposium on Document Analysis and Information Retrieval (SDAIR)*.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1(1):81–106.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Series in Machine Learning. Morgan Kaufmann, San Mateo, CA, USA.

[Quinlan und Cameron-Jones, 1993] Quinlan, R. und Cameron-Jones, R. (1993). FOIL: A Midterm Report. In *European Conference on Machine Learning (ECML)*.

[Raghavan et al., 1989] Raghavan, V., Bollmann, P., und Jung, G. (1989). A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. *ACM Transactions on Information Systems*, 7(3):205–229.

[Rocchio Jr., 1971] Rocchio Jr., J. J. (1971). Relevance Feedback in Information Retrieval. In Salton, G., Hrsg., *The SMART Retrieval System: Experiments in Automatic Document Processing*, Seiten 313–323. Prentice-Hall, Englewood Cliffs, NJ, USA.

[Rosenblatt, 1962] Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan Books, New York.

[Salton, 1971] Salton, G. (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, Englewood Cliffs, NJ, USA.

[Salton, 1991] Salton, G. (1991). Developments in Automatic Text Retrieval. *Science*, 253:974–979.

[Salton und Buckley, 1988] Salton, G. und Buckley, C. (1988). Term Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523.

[Schewe, 1997] Schewe, S. (1997). Automatische Kategorisierung von Volltexten unter Anwendung von NLP-Techniken. Diplomarbeit, Fachbereich Informatik, Universität Dortmund.

[Schlimmer und Granger Jr., 1986] Schlimmer, J. C. und Granger Jr., R. H. (1986). Incremental Learning from Noisy Data. *Machine Learning*, 1(3):317–354.

[Taylor et al., 1997] Taylor, C., Nakhaeizadeh, G., und Lanquillon, C. (1997). Structural Change and Classification. In Nakhaeizadeh, G., Bruha, I., und

Taylor, C., Hrsg., *Workshop Notes on Dynamically Changing Domains: Theory Revision and Context Dependence Issues, 9th European Conference on Machine Learning (ECML '97), Prague, Czech Republic*, Seiten 67–78.

[Valiant, 1984] Valiant, L. G. (1984). A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142.

[Vapnik, 1982] Vapnik, V. (1982). *Estimation of Dependencies Based on Empirical Data*. Springer Series in Statistics. Springer-Verlag.

[Vapnik, 1995] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer, New York, NY, USA.

[Veltmann, 1997] Veltmann, G. (1997). Einsatz eines Multiagentensystems zur Erstellung eines persönlichen Pressespiegels. Diplomarbeit, Fachbereich Informatik, Universität Dortmund.

[Wang et al., 1992] Wang, Z., Wong, S., und Yao, Y. (1992). An Analysis of Vector Space Models Based on Computational Geometry. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seiten 152–160.

[Widmer, 1994] Widmer, G. (1994). Combining Robustness and Flexibility in Learning Drifting Concepts. In Cohn, A. G., Hrsg., *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI '94), Amsterdam, The Netherlands, August 8-12, 1994*, Seiten 468–472, Chichester, UK. John Wiley & Sons.

[Widmer, 1996a] Widmer, G. (1996a). On-line Metalearning in Changing Contexts: MetaL(B) and MetaL(IB). In Michalski, R. S. und Wnek, J., Hrsg., *Proceedings of the Third International Workshop on Multistrategy Learning (MSL-96)*, Seiten 217–228. Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, USA.

[Widmer, 1996b] Widmer, G. (1996b). Recognition and Exploitation of Conceptual Clues via Incremental Meta-Learning. In Saitta, L., Hrsg., *Proceedings of the Thirteenth International Conference on Machine Learning (ICML '96), Bary, Italy*, Seiten 525–533, San Francisco, CA, USA. Morgan Kaufmann Publishers. Extended version as Report TR-96-01, Austrian Research Institute for Artificial Intelligence, Vienna Austria.

[Widmer und Kubat, 1992] Widmer, G. und Kubat, M. (1992). Learning Flexible Concepts from Streams of Examples: FLORA2. In Neumann, B., Hrsg., *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI '92), Vienna, Austria, August 3-7, 1992*, Seiten 463–467. John Wiley & Sons, Chichester.

[Widmer und Kubat, 1993] Widmer, G. und Kubat, M. (1993). Effective Learning in Dynamic Enviroments by Explicit Context Tracking. In Brazdil, P. B., Hrsg., *Machine Learning: ECML '93, Proceedings of the Sixth European Conference on Machine Learning, Vienna, Austria, April 1993*, Lecture

Notes in Artificial Intelligence, No. 667, Seiten 227–243, Berlin, Germany. Springer Verlag.

[Widmer und Kubat, 1996] Widmer, G. und Kubat, M. (1996). Learning in the Presence of Context Drift and Hidden Contexts. *Machine Learning*, 23:69–101.

[Yang, 1997] Yang, Y. (1997). An Evaluation of Statistical Approaches to Text Categorization. Technical Report CMU-CS-97-127, Carnegie Mellon University.

[Yang und Pedersen, 1997] Yang, Y. und Pedersen, J. O. (1997). A Comparative Study on Feature Selection in Text Categorization. In *Proceedings of the 1997 International Conference on Machine Learning (ICML '97).*