Enhancing Text Classification to Improve Information Filtering

Carsten Lanquillon
Hochschule Heilbronn
46 PUBLICATIONS **231** CITATIONS

Enhancing Text Classification to Improve Information Filtering

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dipl.-Inform. Carsten Lanquillon
geb. am 15. August 1972 in Braunschweig

Gutachter:
Prof. Dr. Rudolf Kruse
Prof. Dr. Dietmar Rösner
Prof. Dr. Gholamreza Nakhaeizadeh

Promotionskolloquium:
Magdeburg, den 07. Dezember 2001

Acknowledgements

This dissertation is based on research that I have carried out as a Ph.D. student in the Data Mining Solutions Department at DaimlerChrysler AG, Research & Technology in Ulm, Germany. During this time, I had the chance to spend a semester as a research visitor at the Center for Automated Learning and Discovery, School of Computer Science, Carnegie Mellon University, Pittsburgh, which greatly influenced my research.

This work would not have been realized without the help of many people. It is not possible to enumerate all the people who influenced it. Nevertheless, I would like to express my gratitude to some people in particular.

First of all, I want to thank my thesis supervisor and first referee Prof. Dr. Rudolf Kruse and my co-referees Prof. Dr. Dietmar Rösner and Prof. Dr. Gholamreza Nakhaeizadeh for their support, advice, and encouragement. Prof. Dr. Gholamreza Nakhaeizadeh, head of the Data Mining Solutions research department, not only gave me the freedom to carry out interesting research in the fields of text classification and information filtering but also supported my stay at Carnegie Mellon University.

I am also grateful to my colleagues at the Data Mining Solutions research department, who have helped me in many ways during my time as a post-graduate student. I would especially like to thank my tutor Dr. Ingrid Renz for her support, Georg Oppenberg for introducing me to the information filtering activities at DaimlerChrysler, and Christian Köpf for his help while I was writing my dissertation. My gratitude also goes to the late Prof. Dr. Jürgen Schürmann for some helpful discussions and comments on my research.

Parts of this dissertation, in particular the work on semi-supervised learning, are based on research that I carried out at Carnegie Mellon University. I would therefore like to express my gratitude to all the people I have met and talked to during my stay and work there, especially to Prof. Dr. Tom Mitchell, head of the Center for Automated Learning and Discovery, and the members of his text learning group. The semi-supervised learning framework described in this work is based on Kamal Nigam's seminal work on combining labeled and unlabeled data. For this reason, special thanks go to Kamal Nigam for providing his results and source code, which enabled me to replicate and enhance parts of his research. I also thank Sean Slattery for introducing me to the text classification system rainbow, based on which all my experiments were carried out. Finally, I like to thank Prof. Dr. Amine Bensaid for many fruitful discussions during our joint work in the text learning group.

Last but not least, my thanks go to my parents for their support, care, and patience.

Contents

Kurzfassung

Im heutigen Informationszeitalter steht der Mensch einer Flut von Informationen gegenüber. Informationsfilterung hat zum Ziel, die Informationslast seiner Anwender bezüglich ihrer Interessengebiete zu reduzieren. Dabei werden nicht relevante Dokumente eines Stroms von Informationen entfernt, so dass den Anwendern nur relevante Dokumente präsentiert werden. Wir beschränken uns hier auf Textdokumente und behandeln Informationsfilterung als ein binäres Textklassifikationsproblem, das sich mit Hilfe überwachter Lernverfahren lösen lässt. Anhand von Beispielen mit bekannten Klassenzugehörigkeiten lernen diese Verfahren Klassifikatoren, die dann für die Klassifikation neuer Dokumente verwendet werden.

In realen Anwendungen von Informationsfiltern treffen wir auf drei Probleme. Erstens benötigt man zum Lernen effektiver Klassifikatoren in der Regel eine große Menge klassifizierter Beispiele. Für komplexe Textklassifikationsaufgaben wird die Bereitstellung dieser Lernbeispiele schnell zu einem sehr kostspieligen und auch unüberwindbaren Problem, weil sie von Menschen gelesen und klassifiziert werden müssen. Zweitens setzen viele gängige Lernverfahren homogene Klassen voraus. Die beim Informationsfiltern zugrundeliegenden Klassen sind jedoch oft heterogen. Das dritte Problem liegt in der Annahme, dass die beim Lernen verwendeten Beispiele und die zu klassifizierenden Daten von derselben Quelle stammen. Dokumentquellen können sich aber mit der Zeit ändern, so dass mit dynamischen Aspekten umgegangen werden muss.

In dieser Dissertation wird untersucht, wie stark eine zu kleine Menge an Lernbeispielen, heterogene Klassen sowie sich mit der Zeit verändernde Datenquellen die Klassifikationsleistung beim Informationsfiltern beeinflussen. Für die dabei beobachteten Probleme entwickeln wir geeignete Lösungen. Insbesondere reduzieren wir die Menge benötigter Lernbeispiele durch die Verwendung halbüberwachter Lernalgorithmen. Diese lernen anhand weniger klassifizierter Beispiele und einer größeren Menge nicht klassifizierter Beispiele, die meist sehr kostengünstig zur Verfügung stehen. Weiterhin untersuchen wir Lösungsansätze zum Erlernen heterogener Klassen. Um gezielt mit den dynamischen Aspekten beim Information Filtering umgehen zu können, verwenden wir Methoden der statistischen Qualitätskontrolle. Dadurch versuchen wir, Veränderungen in Informationsströmen ohne zusätzlichen Benutzeraufwand zu erkennen, um dann die Anwender zu benachrichtigen, dass die verwendeten Filter anzupassen sind. Empirische Auswertungen zeigen, dass die in dieser Arbeit vorgestellten Ideen zur Lösung der beim Informationsfiltern beobachteten Probleme beitragen können.

Abstract

Today's information age overloads users with a flood of information. The objective of information filtering is to reduce the users' information load with respect to their areas of interest. The filter is supposed to remove any non-relevant documents from an incoming stream, so that only relevant documents are presented to users. Taking into account only textual documents, we view information filtering as a binary text classification problem. Classification problems can be solved by applying supervised learning algorithms. These learn from a set of examples with known class labels classifiers that predict the class labels of new documents.

In practice, we typically face three problems. First of all, accurately learning classifiers often necessitates a large number of labeled training examples. And, for complex text classification tasks, providing these may easily become prohibitive because training documents are commonly hand-labeled by humans. Second, some of the most frequently applied learning algorithms for constructing text classifiers require homogeneous class definitions. Yet, the two relevance classes in information filtering are often heterogenous. And third, supervised learning is based on the assumption that both the training documents and the new documents to be classified come from the same source. In a long-term application, however, document sources tend to change over time and we have to deal with dynamic aspects.

In this dissertation, we evaluate the extent to which the three problems outlined above impact classification performance. In addition, we present some solutions to remedy the difficulties observed. In particular, we try to reduce the need for labeled training examples by using semi-supervised learning algorithms. These learn from a small number of labeled documents in addition to a large number of unlabeled documents, which are often inexpensive and readily available in large quantities. Subsequently, we investigate the severity of problems inherent to dealing with heterogeneous classes. In order to cope with the dynamic nature of the information filtering task, we employ quality control methods which enable detection of changes in document streams without expensive user feedback and which alert users that their filtering system requires adaptation to these changes. Experimental evaluations performed on several real-world text collections show that the ideas developed in this dissertation are indeed useful to alleviate some of the difficulties encountered in information filtering.

Chapter 1

Introduction

1.1 Motivation

The promise of the information age entails making information available to people any time, any place, and in any form. Realizing such a promise depends on innovations in areas that impact the creation of information services and their communication infrastructures. However, this realization can easily become a mixed blessing without methods to filter and control the potentially unlimited flux of information from sources to their receiving end-users.[1]

The proliferation of information available on the World Wide Web, corporate intranets and databases, electronic news wires, and other media is overwhelming. However, while the amount of information available to us is continually increasing, our ability to digest this information remains constant. Moreover, the creation and dissemination of information is supported by a growing number of tools, while we are only insufficiently supported in processing and assimilating this information. We are often experiencing *information overload* when we fail to follow critical news, events, and trends.[2] Common effects of this are, for example, attention deficit disorder, increased cardiovascular stress, weakened vision, confusion, and impaired judgement.[3]

At this point, the application of tools that automatically assist a user in organizing and managing this *information glut* should come into consideration.[4] For instance, these tools could filter interesting or relevant from non-relevant information according to a specified user interest. Information could be summarized and visually presented in such a way that it is more easily accessible. Also, identifying hidden knowledge within the available data is a crucial issue. Goals and ideas which are common in the field of knowledge discovery and data mining must be applied to little or even unstructured data, such as text or multimedia data.

[1]From Loeb and Terry (1992), p. 27.

[2]See Sheridan and Ferrell (1974), pp. 133–134, for an early reference to information overload.

[3]See Shenk (1997), pp. 37–38.

[4]Information overload is also referred to as information glut, see Shenk (1997), for example.

1.2 Problem Description

The objective of *information filtering* is to reduce the users' information load with respect to their areas of interest. The filter is supposed to remove any non-relevant documents from an incoming stream, so that only relevant documents are presented to users. Therefore, information filtering can be described as a binary classification problem.[5] In this work, we limit the space of possible documents to textual documents and regard information filtering as a specific instance of text classification. Note that, in information filtering, documents are generally classified immediately, online, as they arrive. Instead, documents could be processed and presented in batches. This would allow ranking of documents according to their estimated degree of relevance, which is often misleadingly referred to as *information routing*.[6]

Classification problems can be solved by applying supervised learning algorithms. These learn from a set of examples with known class labels classifiers that predict the class labels of new, previously unseen observations. Such learning algorithms are studied in depth in the machine learning community. By *machine learning*, we understand the study of computer algorithms that improve automatically through experience, based on concepts from artificial intelligence, probability theory, statistics, information theory, and other disciplines.[7] A variety of text learning algorithms has been studied and compared in the literature.

There are several assumptions that must hold in order to achieve reasonable results when applying supervised learning algorithms to the task of text classification. We identify three major assumptions that tend to be violated in practice:

1. *Availability Assumption.* In order to learn reasonably accurate classifiers, we must be provided with enough labeled training examples. For text classification tasks, this usually requires a person to read numerous documents and to decide on the class label to be given to each of these documents: a tedious and extremely time consuming process. And for complex learning tasks, providing sufficiently large sets of labeled training examples easily becomes prohibitive. Thus, we generally cannot assume that we are provided with enough labeled training examples.

2. *Homogeneity Assumption.* Depending on the domain, the two relevance classes may be very broad. For example, when filtering news stories, there are typically several different topics which a user either likes or dislikes. Hence, the classes defined by the filtering task tend to be heterogeneous, so that the associated documents may be difficult to classify. Moreover, some of the learning algorithms which are most commonly applied to solve text classification problems require that classes be homogenous.

3. *Stationarity Assumption.* Effectively using information filters, or text classifiers in general, requires that the distributions of training documents and new documents

[5]See Lewis (1997), p. 75.
[6]See Hull (1998), p. 45.
[7]For example, see Langley (1996) or Mitchell (1997) for a general introduction.

are the same, or at least similar.[8] Assuming that both the training documents and the new documents are generated by class-specific document sources, the application of classifiers that were learned from examples requires that the document sources be time invariant. Yet, even though this stationarity assumption may hold initially, it is likely to become invalid in a long-term application. For various reasons, both the topics and contents of new documents can be expected to change over time. The document sources are *dynamic*, or *non-stationary*, rather than stationary. With time, this may cause a classifier to become less effective than expected.

Under these circumstances, the primary task of designing information filtering systems which are to be employed in real-world settings can be described as follows:

The application goal of this dissertation is to provide techniques that

1. learn reasonably accurate information filtering systems and
2. maintain classification performance in a long-term application

while *minimizing the user effort required*.

Note that the constraint regarding the user effort takes the difficulty of providing labeled training examples into account. This affects both learning a classifier and trying to maintain its classification effectiveness in a long-term application. Furthermore, the possible heterogeneity of the relevance classes may hinder a learning algorithm from inducing an effective classifier.

There are other general assumptions in the supervised learning setting which we do not address in this dissertation. For instance, examples are commonly represented by a set of features that are assumed to be sufficient in order to distinguish among classes. Also, the training examples are assumed to be independent. We are not aware of applications where violations of these assumptions cause serious problems in the context of text classification.

1.3 Research Goals

With this application goal and the assumptions described above in mind, we derive the following three research challenges:

1. Explore methods which will reduce the need for labeled training documents in order to relax the *availability assumption*.

2. Evaluate the extent to which violations of the *homogeneity assumption* affect classification performance.

3. Provide methods which indicate that the *stationarity assumption* has been violated and which can thus help to maintain classification performance.

These challenges lead to three extensions of basic text classification which will be dealt with in the second part of this dissertation as described in the following overview.

[8]See Schürmann (1996), p. 309.

1.4 Overview

This dissertation is structured in three parts which comprise eight chapters altogether. The first part, Chapters 1 through 3, covers some preliminaries of this dissertation. The second part, Chapters 4 through 6, addresses text classification enhancements which deal with the research issues motivated and stated above. Finally, by providing an empirical evaluation and the conclusions, the third part discusses the key results and contributions of this work.

In Chapter 2, we formally define the task of information filtering and introduce possible filtering variants. Focusing on content-based filtering of textual documents, we identify information filtering as a special case of text classification. Having reviewed common measures for performance evaluation of information filtering systems, we move on to some historical background on the development of information filtering. Finally, we briefly review several well-known filtering projects before we compare related tasks.

Chapter 3 introduces text classification: we define the task of text classification and show its relationship to the task of information filtering. Our goal is to use machine learning techniques for automatic text classification. Hence, we identify two basic subtasks which we examine closely: representing textual documents in a way that is appropriate as input for machine learning algorithms and the application of these learning algorithms proper.

Supervised learning algorithms typically require enormous amounts of training data to learn reasonably accurate classifiers. Yet, in many text classification tasks, labeled training documents are expensive to obtain, while unlabeled documents are often readily available in large quantities. In Chapter 4 we develop a semi-supervised learning framework which allows us to learn from both labeled and unlabeled data to mitigate the violation of the *availability assumption*. Also, we provide a thorough empirical evaluation of the approach proposed.

Some of the learning algorithms which are commonly applied to automate text classification require that classes be homogeneous. In Chapter 5, we evaluate the extent to which violations of the *homogeneity assumption* affect classification performance of two frequently used classifiers. In addition, we briefly discuss approaches that could remedy this problem.

In order to cope with the violation of the *stationarity assumption* in a long-term application, we favor a quality control methodology that attempts to detect changes and adapts the applied information filter only if classification performance needs to be maintained. Therefore, our prime interest in Chapter 6 is to detect changes in a text stream either with little or with no user feedback.

In Chapter 7, we integrate the semi-supervised learning framework and the quality control component introduced in Chapters 4 and 6. The system resulting is evaluated based on a new text corpus which was not used previously during the process of this work.

Finally, Chapter 8 concludes with a summary of the main contributions of this dissertation. In addition, we describe possible issues for future work.

Chapter 2

Information Filtering

In this chapter, we formally define the task of information filtering and shed some light on possible filtering variants. We focus on content-based filtering of textual documents, which turns out to be a special case of text classification. Techniques involved in text classification are covered in detail in the following chapter. In addition, we describe a general framework for an information filtering system and review some measures for performance evaluation. We then give some historical background on the development of information filtering and briefly review several well-known filtering projects before we compare related tasks.

2.1 Definition and Terminology

The objective of information filtering is to reduce the users' information load with respect to their areas of interest. Information filtering is an information seeking process in which non-relevant documents from an incoming stream are rejected according to a specific long-term user interest in such a way that only the relevant documents are presented to the user. More precisely, we formally pose the filtering problem as follows.[1]

Definition 2.1.1 (Information Filtering)
Assume a space of documents $\mathcal{D}$. With respect to a specific long-term user interest, ψ, we define information filtering as a mapping, $f_\psi : \mathcal{D} \mapsto \{0, 1\}$, from the document space onto either zero or one, which corresponds to rejecting or accepting a document, respectively. Alternatively, information filtering can be defined as a mapping from the document space onto the unit interval, $f^\star_\psi : \mathcal{D} \mapsto [0, 1]$, such that $f^\star_\psi(d)$ reflects the relevance of a document d with respect to the user interest ψ. Note that this notation further requires thresholding of the relevance scores if the documents are to be explicitly rejected or accepted.

Information filtering is a process actively conducted by humans, with or without the assistance of a machine,[2] in order to cope with information overload. The goal of an *information filtering system* is to automate this process. Understanding information filtering

[1]See Mostafa *et al.* (1997) or Hull (1998) for similar definitions.
[2]See Oard (1997) p. 142.

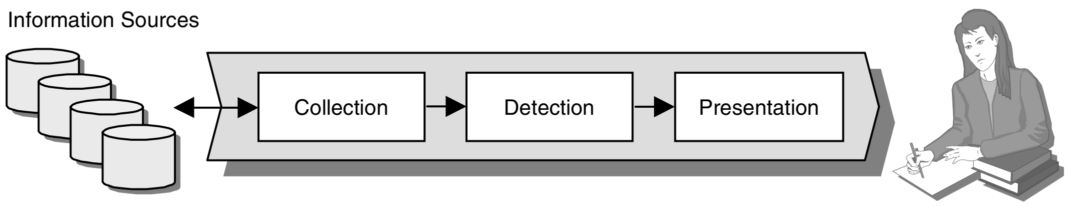


Figure 2.1: The three subtasks of an information seeking process (following Oard (1997), p. 142).

as an information seeking process leads to three subtasks for such a system: collecting documents, detecting relevant documents, and presenting the results to the user.[3] This subdivision is depicted in Figure 2.1.

It is arguable whether or not information collection should be part of the filtering system. This probably depends on whether we are considering filtering systems with an active or passive initiative of operation.[4] In early descriptions of information filtering systems, the passive collection of documents such as incoming electronic mail was commonly assumed.[5] However, due to an ever increasing amount of electronically accessible information, actively collecting documents is gaining in popularity. For instance, a collection of agents could be used to crawl the World Wide Web looking for relevant information.[6] In this dissertation, we do not further address the aspect of information collection. Instead, we assume a stream of documents as input to a filtering system. Detecting relevant documents from a stream of incoming documents is the core of any information filtering system. Essentially, this concerns finding a mapping, f_ψ or $f^\star_\psi$, to estimate the relevance of documents according to Definition 2.1.1. We will have more to say on this subtask in the sections that follow. The final subtask is responsible for presenting the documents selected according to their relevance to the user. As a filtering system is unlikely to be perfect, even if the remaining documents are predicted to be relevant, the means of presentation can further enhance the users' ability to identify what really meets their information needs. These issues involve the study of fields like human-computer interaction and information visualization, which we will not discuss in this dissertation. To summarize, we see an information filtering system with an emphasis on detecting relevant information as is presented in the following.

Definition 2.1.2 (Information Filtering System)
An information filtering system automates the process of information filtering with the objective to reduce information overload. It either generates or accepts a stream of documents, estimates the relevance of each document, and provides either only the relevant documents or a relevance score for each document for presentation to the user with respect to the user's interest.

There are several basic points that need to be put in concrete terms and in which specific information filtering systems may differ. In the following we analyze four specifications:

[3]See Oard (1997), p. 142.
[4]See Shapira *et al.* (1997).
[5]For example, see Denning (1982)
[6]For example, see Balabanovič (1997), or Pazzani and Billsus (1997).

the type of input to the system, i.e. the space of documents, requirements for the output, profile construction, and the concept of relevance. At this point, we do not consider the location of operation. Note that a filtering system can be located at the user's site, on special intermediate servers 'between' the information provider and the users, or directly at the information sources.[7] Also, we do not discuss any details concerning the architecture or implementation of an information filtering system.

Input

We assume that we are given a stream of documents as input. What is the nature of these documents? Generally, we can filter any type of information that can be represented digitally. On the one hand, this could be in the form of traditional textual documents, like electronic mail, Usenet newsgroups articles, or other online news. On the other hand, new media such as images, audio, video, and multimedia documents in general might be used. In this dissertation, we limit the input space of documents to *textual documents*. Hence, we always mean textual documents when talking about documents in subsequent chapters.

Output

Once a filter has estimated the relevance of documents from the incoming stream, the results have to be presented to the user. The presentation of information does not belong to the scope of this work. Note, however, that it may impose some requirements for the output of a filtering system. A primary issue is when and in what form the results are presented to the user. We focus on two common alternatives to show this. One way is to classify documents as they arrive, online, and present them directly to the user, if so decided. Hence, the decision whether to reject or accept a new document is made independent of other incoming documents. Alternatively, documents could be classified asynchronously and a personal report of interesting documents could be created at regular intervals. This batch processing allows ranking of documents according to their estimated degree of relevance, which is often misleadingly denoted as *information routing*.[8] We choose to classify documents online as they arrive because it is the more common variant of information filtering.

Profile Construction

Knowing what a user is actually interested in is a fundamental issue in any information seeking task. In information filtering, we are typically concerned with a long-term user interest. The compromised representation of this information need is usually referred to as the *user profile*. In the information retrieval and database community, this is also known

[7] See Shapira *et al.* (1997).

[8] This terminology is commonly used in the Text REtrieval Conference (TREC) environment. See Hull (1998), p. 45, for example.

as a *query*. However, since the user's interests may be manifold, the profile may actually be a set of queries. In this sense, we use the term query to refer to distinct parts of users' interests within their profile.

How do we construct a profile? Two main directions come into mind: human and machine design. First, users could hand-code their own filters, for instance by specifying some simple rules that satisfy their needs. Or, professional knowledge engineers could provide specific profiles. While the first approach is often particularly time-consuming and tedious, the second lacks the potential for personalization according to a user's own special needs. The second direction, which we follow in this dissertation, tries to avoid both of these imputations by automatically constructing profiles using machine learning techniques. Typically, these learn profiles from a set of training documents for which relevance judgements are given. Note that describing how a profile can be constructed does not yet specify how the profile will be represented. Matching a document to a user profile to obtain a measure of similarity corresponds to the relevance mapping described in Definition 2.1.1.

The Concept of Relevance

A crucial open question remaining is what is relevant or interesting for a user? In the literature, there are currently three filtering paradigms that differ in the criteria used to select relevant documents:[9]

- *Content-based filtering*, or *cognitive filtering*. This technique makes the decision upon the relevance of a document based solely on features that can be exploited from its content, whereby each user is assumed to operate independently. For textual documents, common features are n-grams, words, or phrases.[10] This paradigm is probably the best-known and most commonly applied technique. See Section 2.4 for some well-known research projects founded on the principle of content-based filtering.

- *Collaborative filtering*, or *social filtering*. This paradigm assumes that an effective way to find interesting or relevant documents for one user is to find other people who have similar interests.[11] Automating the 'word-of-mouth' metaphor, by which individuals recommend products or services to one another,[12] collaborative filtering can select documents for one user based on usage, preferences, annotations, or opinions of other users. This feedback can be given explicitly for a document by the users. However, they often tend to refrain from doing so. Rather than asking for explicit feedback, a system might use implicit feedback in order to remedy this.[13] For instance, simply observing how frequently a document is read could be used as a characteristic in analogy to the tendency of a often-read paper document.[14]

[9]See Malone *et al.* (1987), pp. 391.
[10]See Subsection 3.2.2 for more details.
[11]See Breese *et al.* (1998), p. 1.
[12]See Shardanand and Maes (1995).
[13]See Nichols (1997).
[14]See Hill *et al.* (1992) for the concept of 'read wear'.

An early collaborative filtering approach is the Information Tapestry project developed by Goldberg *et al.* (1992). The system filters electronic mails by exploiting other peoples' annotations. Another well-known example of this technique is the GroupLense project described by Resnick *et al.* (1994) and Konstan *et al.* (1997). This system has been implemented for filtering Usenet news postings. Similar to this is the music recommendation system Ringo by Shardanand and Maes (1995), which is the predecessor of the commercial Firefly system.

- *Economic filtering.* Economic filtering approaches are based on cost-benefit assessments which consider information as a commodity in economic markets.[15] Based on explicit or implicit pricing mechanisms, the decision on documents is made in such a way that cost and value are balanced. Note that the costs of a document need not correspond to monetary units. For instance, the decision to read or ignore a newspaper article may be based solely on its length, i.e. the expense of time required to peruse it.

While there is only very little work on economic filtering to be found in the literature as yet, a great deal of research has been done on content-based and collaborative filtering. The following outlines the relationship between the latter two filtering paradigms.

Depending on the application, one paradigm is likely to be more valuable than the other. For example, if the objective is to collect information on a certain topic, content-based filtering may be suitable. Yet, if the aim is to gather information in order to keep up to date with a certain community, collaborative filtering is more appropriate.[16] Obviously, a collaborative filtering system is good at identifying novelty because it is guided by humans.[17] However, this technique can only succeed when the users are not overloaded with information. If this should be so, content-based filtering could assist users in managing information overload as content-based filtering systems are designed to find relevant documents according to their similarity to documents that a user had found relevant before. So, the core technique tends to fail if something novel is to be found. A remedy could be to identify novelty by means of a change detection technique that would alert a user whenever it hypothesizes that there is a change in the input document stream that would require that the filtering system be adapted. More on this issue is found in Chapter 6. At this point, note that both content-based and collaborative filtering "can contribute to the other's effectiveness, potentially allowing an integrated system to achieve both reliability and serendipity."[18] Since humans and machines base their decisions on different features, a hybrid approach can be a promising solution.[19]

Because we intend to use supervised machine learning algorithms to construct user profiles, we must be provided with training examples. For content-based filtering, these are documents labeled either relevant or non-relevant by the user the filter is to be built for. Hence, giving feedback is for a user's own benefit. In the collaborative setting, in contrast,

[15]See Ferguson and Karakoulas (1996).

[16]See Sheth (1994), p. 14.

[17]See Oard (1997), p. 172.

[18]From Oard (1997), p. 172.

[19]For example, see Mock and Vemuri (1997), Delgado *et al.* (1998), or Shapira *et al.* (1999).

users are asked to give feedback not for their own but for the benefit of others. Obviously, somebody is required to 'consume' a document first. Fast and early readers will frequently give feedback while receiving little or no support in return.[20] In addition, providing expert rather than peer annotations for documents requires effort that has economic value. Hence, these annotations may not be freely available. Considering that costs of computing and communications resources are continously decreasing, content-based filtering can provide a competitive source of information on which further selections can be based. Thus, "the effectiveness and efficiency of content-based selection techniques have the potential to significantly influence the price of the annotations on which social filtering is based."[21] The bottleneck of collaborative filtering systems is to achieve a 'critical mass' of participants, which is necessary to obtain reliable recommendations. In a large, publicly accessible system, however, privacy can become a critical social issue. While protecting users' data in a content-based filtering system is possible especially if the filtering system is located at the user's site, it is much more complicated in a collaborative system since it builds on sharing annotations among different users. Note that the "tension between the desire for privacy and the benefit of free exchange of information may ultimately limit the applications to which social filtering can be applied."[22]

Delimitation of Thesis

Above, we have described several information filtering variants. In this dissertation, we focus on content-based filtering of textual documents. Therefore, we always refer to the content-based paradigm when talking about information filtering. Relevant documents are presented to the user independent of other incoming documents, based only on their degree of relevance. Our objective is to automatically learn user profiles by using machine learning techniques. Mapping textual documents onto a predefined set of categories—here accept and reject, or relevant and non-relevant, respectively—is commonly known as text classification.[23] We give a detailed introduction to the techniques involved in text classification in Chapter 3.

2.2 General Framework

As will be discussed in Section 2.5, information filtering is strongly related to the well established research area of information retrieval. Here, we describe a general information filtering model derived from information retrieval as depicted in Figure 2.2.

Recall Definition 2.1.1 and assume a document space $\mathcal{D}$ and a particular user with an information need ψ from a fictitious user interest space $\mathcal{N}$. The process of information filtering is a mapping $f_\psi : \mathcal{D} \mapsto \{0, 1\}$ where the values zero and one correspond to re-

[20]See Kilander *et al.* (1997) for a discussion of this issue.
[21]From Oard (1997), p. 172.
[22]From Oard (1997), p. 171.
[23]See Lewis (1997), p. 75.

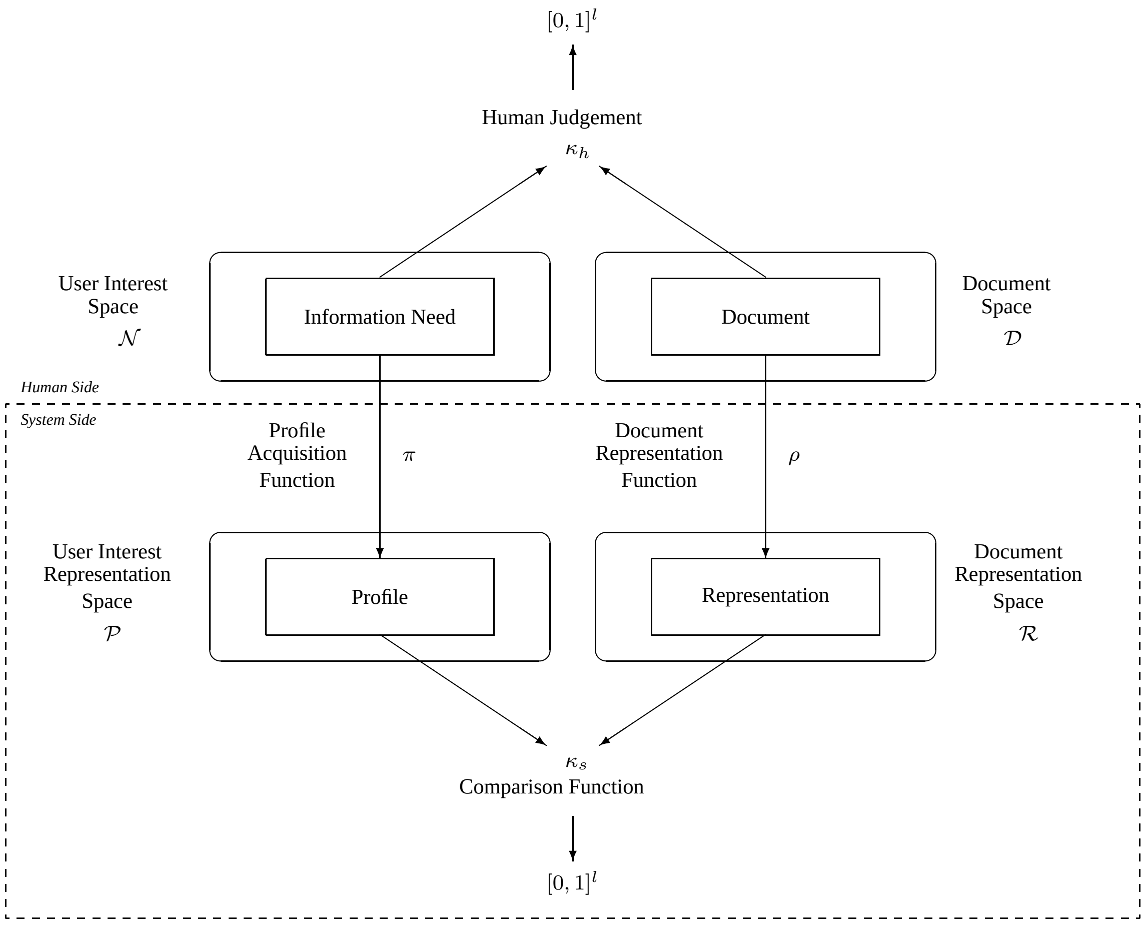


Figure 2.2: A general framework for an information filtering system (following Oard (1997), p. 160).

jecting and accepting a document, respectively.[24] Now, let us further analyze this process as it is conducted by a human without the assistance of a machine. The user tries to apply an information need to a set of documents with the objective of detecting some relevant documents. The user interest may be multifaceted and thus be based on different aspects. Let there be l different aspects that can be measured on l numeric scales. For instance, there could be a set of values reflecting the similarities between a document and different topics that a user likes or dislikes. We formalize the user's judgement of the relationships between the user's interest and a document as a comparison function $\kappa_h : \mathcal{N} \times \mathcal{D} \mapsto [0,1]^l$ (top of Figure 2.2). Based on the result of this comparison, the user can then decide, even though usually subconsciously, whether to reject or accept a document. This corresponds to a mapping $\tau_h : [0,1]^l \mapsto \{0,1\}$. In terms of Definition 2.1.1, we see that

$$f_\psi(d) = (\tau_h \circ \kappa_h)(\psi, d) \tag{2.1}$$

for a given information need $\psi \in \mathcal{N}$ and any document $d \in \mathcal{D}$.

[24]Note that we do not consider mapping documents onto a degree of relevance which would allow ranking of a set of documents according to their degree of relevance.

Every approach to automate this process must have four basic components:[25]

- A technique for representing documents, denoted as the document representation function $\rho : \mathcal{D} \mapsto \mathcal{R}$. This function maps a natural document onto its document representation, also known as a document surrogate, in the document representation space $\mathcal{R}$.
- A technique for representing the user's information need, denoted as the profile acquisition function $\pi : \mathcal{N} \mapsto \mathcal{P}$. This function maps a user interest onto a profile in the user interest or profile space $\mathcal{P}$. Note that a profile may consist of several queries, which may be represented in the same space as the documents. In this case, we have $\mathcal{P} = \mathcal{R}^q$, where q is the number of distinct queries that form the profile and correspond to topics that the user likes or dislikes.
- A technique for matching the representation of the user's information need against the document representations, denoted as the comparison function $\kappa_s : \mathcal{P} \times \mathcal{R} \mapsto [0, 1]^l$. Again, the range $[0, 1]^l$ reflects different aspects of the user interest which are measured on l numeric scales, similar to the range of the human judgement function κ_h. For example, in the case of q queries that form the profile, we could have $l = q$ similarities between a document and each of the queries.
- A technique for using the results of this comparison, denoted as the system decision function $\tau_s : [0, 1]^l \mapsto \{0, 1\}$. This function may be very similar to the human decision function τ_h. Yet, in reality, a human typically decides subconsciously, and the system decision function is a means of formalizing this action. For instance, if the profile consists of $l = q$ queries reflecting l topics which the user likes or dislikes, this function could simply respond with one if the corresponding document is most similar to a topic that the user likes. Otherwise, the function would respond with zero. Note that this component could be considered as a part of the presentation task in the information seeking process as depicted in Figure 2.1.

According to this very general model, the four system functions ρ, π, κ_s, and τ_s must be implemented in order to build the core of an information filtering system. Recall that we focus on content-based filtering of textual documents and thus see information filtering as a binary text classification problem. In Chapter 3, we will introduce text classification techniques that permit the construction of these functions.

An obvious objective for a filtering system is that the human decision τ_h based on the result of the human judgement function κ_h be equivalent to the system decision τ_s based on the result of the comparison function κ_s in the representation spaces:[26]

$$(\tau_h \circ \kappa_h)(\psi, d) = (\tau_s \circ \kappa_s)(\pi(\psi), \rho(d)) \qquad \forall \psi \in \mathcal{N}, \forall d \in \mathcal{D} \tag{2.2}$$

We see that formulating an objective function for an information filtering system is straightforward. However, evaluating a system's performance with respect to this objective is not trivial. We will discuss this issue in the following section.

[25] See Oard (1997), p. 159.

[26] See Oard (1997), p. 159, for a similar formulation.

2.3 Performance Evaluation

Having built an information filtering system, we would like to know how well it performs. Evaluating a system concerns measuring the ability of the system to satisfy the user.[27] Typically, this involves the system's *efficiency* and *effectiveness*. Effectiveness is a measure of the system's ability to accept relevant documents while at the same time rejecting non-relevant ones. "It is assumed that the more effective the system, the more it will satisfy the user."[28] However, only an effective system that does not put too much effort on the user can actually satisfy the user. This is where efficiency comes in. Efficiency is a measure of the performance in relation to the resources that are consumed to produce the filtering output. As a rule, there are computer resources such as computation time and more notably, in the context of supervised learning, the user effort for providing labeled training data. Because of continually increasing computing capabilities, we ignore efficiency issues related to computer resources for now. Yet, the effort that is demanded of the user will be a key issue in the chapters that follow because the goal of this dissertation is to provide techniques for learning an information filtering system and for maintaining classification effectiveness in a long-term application, while minimizing the required user effort. At this point, however, we focus on the system's effectiveness only.

How do we measure effectiveness? Recall that we view information filtering as a binary text classification problem. The classification decision is whether or not a document is relevant for a certain user. In the previous section, we saw that building a filtering system corresponds to constructing the functions ρ, π, κ_s, and τ_s so as to imitate a specific human judgement, κ_h, as depicted in Figure 2.2, together with the subsequent decision τ_h. This objective is based on the assumption that a document can unambiguously be considered relevant or non-relevant according to a certain information need. In reality this assumption does not always hold, because 'relevance' is a subjective notion: the judgements of different users about the relevance of particular documents may differ significantly.[29] And even the judgements of the same user given at different times may differ. Hence, assessing the effectiveness of an information filtering system is not as straightforward as many standard classification tasks in which each object to be classified has exactly one class label which is to be returned by a classifier. In filtering, the assignment of a class label depends on human judgement, so that effectiveness measures of a filtering system are not as objective as in standard classification tasks. Despite this fact, we will use standard measures from machine learning and information retrieval to evaluate the effectiveness of information filtering systems as described below. Nevertheless, note that the problem of contradictory human judgements can be expressed in a probabilistic framework by referring to the posterior probabilities of classes given a document rather than to the plain classification decision.[30]

Recall again that the classification decision in information filtering is whether or not a document is relevant for a particular user. This is a hard classification decision obtained

[27]See van Rijsbergen (1979), p. 145.
[28]From van Rijsbergen (1979), p. 145.
[29]See van Rijsbergen (1979), p. 146.
[30]See Subsection 3.3.4 (pp. 75 ff.) for a brief introduction of this probabilistic notation.

Relevant?	User says *yes*.	User says *no*.
System says *yes*.	a	b
System says *no*.	c	d

Table 2.1: Contingency table for $n = a + b + c + d$ relevance judgements.

by applying some decision function τ_s to the classification scores representing the relevance judgements for a particular document. For the purpose of evaluation, assume that a set of n documents has been classified by an information filtering system as either *reject* or *accept* and the true class labels are provided by the user. The relationship between the classification decisions and the true class labels can be summarized in a contingency table as shown in Table 2.1. For example, entry a is the number of documents that the information filter accepts for presentation to the user and that are, in fact, relevant. Note that the values given by the contingency table may vary depending on the decision function τ_s. It is often possible to provide a parameterized decision function, such as a threshold function, which depends on some free parameter of the system. Consequently, we can obtain an operating characteristic curve for an effectiveness measure as a function of this parameter instead of just a single effectiveness score. For the following definition of the common effectiveness measures, we assume that the decision function and the resulting contingency table are fixed.

Common performance measures in machine learning are *accuracy* and *error rate*, or *error* for short. While the error estimates the probability of misclassification, accuracy estimates the probability that a document is classified correctly, independent of the class label assigned. Note that the sum of error and accuracy is always one. In terms of the contingency table, they are defined for a non-zero total number of documents as[31]

$$error = \frac{b+c}{n} \tag{2.3}$$

$$accuracy = \frac{a+d}{n} = 1 - error \tag{2.4}$$

In many information filtering tasks, the non-relevant documents by far outnumber the relevant documents. Thus, the trivial approach of rejecting all documents usually yields high classification accuracy, but does not reflect the user's need. Moreover, the attempt to select relevant documents often degrades accuracy when compared to the trivial approach. Hence, building a filtering system with the objective of maximizing classification accuracy may not yield the desired effectiveness.

To account for this problem, an alternative to traditional accuracy and error rate is to use a more evolved approach that assigns different rewards or penalties for each pair of system response and user judgement as depicted in the contingency table. This leads to the family of *utility* measures defined as[32]

$$utility = au_a + bu_b + cu_c + du_d \tag{2.5}$$

[31]For example, see Yang (1999).

[32]Utility measures are used in the TREC filtering track, e.g. see Lewis (1997).

Choosing appropriate values for $u_.$ allows weighting of any of the four combinations of human judgement and system decision. Note that with $u_a = u_d = 1$ and $u_b = u_c = 0$, maximizing utility is equivalent to maximizing classification accuracy.

The problem of the skewed relevance distribution among the documents is also very common to information retrieval. This field has produced measures such as *recall*, *precision*, and *fallout*, which are insensitive to the total number of documents and can thus abstract from the relevance distribution and better reflect effectiveness. Put in words, *recall* is an estimate of the probability that the filter lets through relevant documents to the user. Likewise, *precision* is an estimate of the probability that a document being presented to the user is indeed relevant, whereas *fallout* is an estimate of the probability that a document, in spite of being non-relevant, is presented to the user. In terms of the contingency table, these measures are defined for non-zero denominators as[33]

$$recall = \frac{a}{a+c} \tag{2.6}$$

$$precision = \frac{a}{a+b} \tag{2.7}$$

$$fallout = \frac{b}{b+d} \tag{2.8}$$

Given the parameter called *generality*, $g = \frac{a+c}{n}$, which is a measure of the density of relevant documents in the set of all documents, we can observe the following functional relationship:[34]

$$\textit{precision} = \frac{g \cdot \textit{recall}}{g \cdot \textit{recall} + (1-g) \cdot \textit{fallout}} \tag{2.9}$$

Therefore, we shall not consider fallout any further but continue to look at the relationship between recall and precision.

As shown in Figure 2.3, recall and precision co-vary in a loosely specified way.[35] Obviously, accepting most documents yields a high recall at low precision, whereas rejecting most documents typically yields low recall at high precision. Choices in between these extremes require some trade-off between recall and precision. In fact, either one of these measures may be misleading when examined in isolation. In information retrieval, it is assumed that "precision and recall are sufficient for the measurement of effectiveness."[36] So, recall and precision should be used in combination to ensure a non-trivial evaluation of a system's effectiveness.[37]

The dissatisfaction with having a pair of numbers required to measure effectiveness in information retrieval led to the proposal of composite measures. One common measure that is frequently used in cross-system comparisons is the recall-precision *break-even point*.[38] The idea is to tune the parameters of the system in such a way that the recall of

[33] For example, see Lewis (1995) or Yang (1999).

[34] See van Rijsbergen (1979), p. 149. Note that, in statistical terms, generality is equivalent to the prior probability of relevant documents.

[35] See van Rijsbergen (1979), p. 154.

[36] From van Rijsbergen (1979), p. 145.

[37] See Lewis (1991).

[38] See Lewis (1992a).

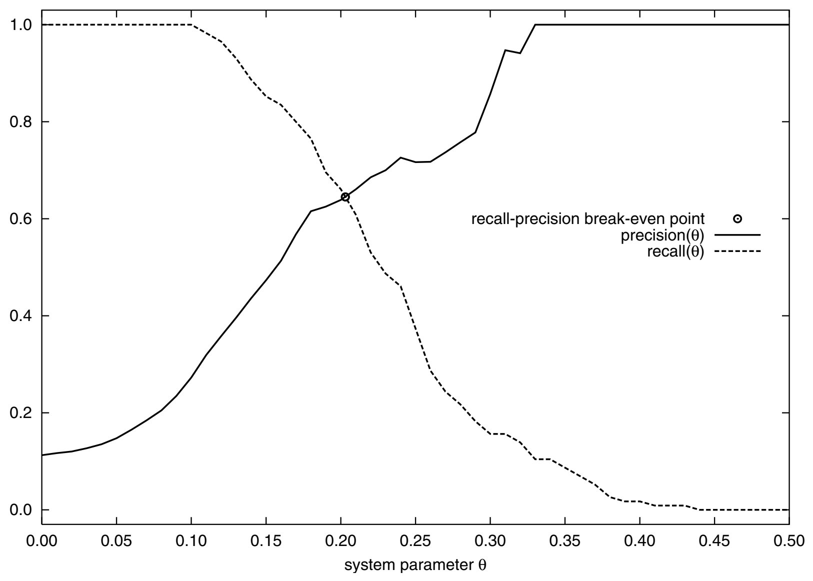


Figure 2.3: Recall and precision operating characteristic curves as interpolated functions of a filtering system's free parameter. Here, this parameter is the value θ, at which classification scores assigned to new documents by a particular classifier are thresholded in order to obtain hard classification decisions. Generally, high recall can be obtained at the expense of low precision, and vice versa.

the system is the same as its precision; see Figure 2.3. And, the objective when building a filtering system is to maximize this value. A problem with this approach is that recall and precision cannot always be made equal. Therefore, recall and precision values often have to be interpolated, which can yield a break-even point that actually cannot be achieved by the system. Furthermore, Schapire *et al.* (1998) state that the break-even point is "neither a desirable nor an informative target from a user's perspective."

Van Rijsbergen introduced a family of measures which are parameterized by a value $\beta \in [0, \infty)$ which reflects the relative importance a user assigns to recall and precision:[39]

$$F_\beta = \frac{(\beta^2 + 1) \cdot \textit{recall} \cdot \textit{precision}}{\beta^2 \cdot \textit{precision} + \textit{recall}} \tag{2.10}$$

Note that, for $\beta = 0$, the resulting measure F_0 is the same as precision, whereas the measure corresponds to recall with $\beta \to \infty$. For instance, a common choice is $\beta = 1$, which gives the same weight to both recall and precision, yielding

$$F_1 = \frac{2 \cdot \textit{recall} \cdot \textit{precision}}{\textit{precision} + \textit{recall}} \tag{2.11}$$

All the effectiveness measures described above require that we have knowledge about the true class labels, i.e. the user's judgements, of all the documents involved in the evaluation. For this reason, the available set of labeled documents is commonly split into independent training and test sets for an experimental evaluation. Doing this allows us to estimate the expected effectiveness of the filtering system. It is assumed that a system

[39]See van Rijsbergen (1979), pp. 168–176.

which performs well under a large number of experimental conditions is likely to perform well in an operational situation where we do not know whether or not a document is relevant for a particular user.[40]

Nevertheless, we know that the document stream is likely to change in a long-term application and we cannot guarantee that a filtering system will maintain its effectiveness as time progresses. This issue calls for *adaptive* information filtering systems.[41] We therefore decide to monitor the performance of the system while it is in operation. To do so, we need to constantly evaluate some of the effectiveness measures suggested, which requires the true class labels for at least some of the classified documents. Even if it is possible to obtain feedback for documents that are presented to the user, the objective of information filtering, namely to reduce information overload, generally rules out getting feedback for documents that the system has rejected. This is a serious problem that has to be tackled in the context of quality control for information filtering. We will have more to say on this issue in Chapter 6.

2.4 Background and State of the Art

We briefly outline the research history of information filtering and review some well-known content-based filtering projects. See Oard (1997) for a more detailed overview on the research history and a discussion of content-based and social filtering projects. Furthermore, Oard and Kim (2000) maintain an excellent web page on information filtering resources.

The concept of information filtering first appeared in the original work of Luhn (1958a). At that time it was called *selective dissemination of information (SDI)*. In Luhn's concept of a "Business Intelligence System", library workers would create profiles for individual users. Lists of new documents for each user would then be produced, using these profiles with an exact-match text selection system. Requests for specific documents would be recorded and used to automatically update the corresponding user's profile. With increasing interest in SDI, a special interest group on this subject (SIG-SDI) was created at the American Society for Information Science in the late 1960's.

The need for information filtering and condensation was further discussed by Ackoff (1967). He noted that the two most essential functions of information systems are filtration and condensation in order to prevent information overload. In his ACM President's Letter, Denning (1982) coined the term *information filtering*. His objective was that not only the generation of information, but also the reception of information be discussed. Denning described the need to filter electronic mail in order to separate messages of varying importance.

Since the late 1980's, several papers on information filtering applications with varying information sources such as electronic mail, newswire articles and Usenet news have appeared in the literature. For example, Malone *et al.* (1987) worked on the Information

[40]See van Rijsbergen (1979), p. 147.

[41]See Hull (1998), pp. 50–51, for a discussion of this issue.

Lens project employing rule-based user profiles to filter email messages that match keywords in mail fields and identified the three filtering paradigms described above, namely content-based, collaborative, and economic filtering. Allen (1990) concluded that the user's reading history can be used to suggest possible future selections and to further use these selections to retrieve additional relevant articles. Foltz (1990) investigated the use of latent semantic indexing in information filtering. And Foltz and Dumais (1992) analyzed different information filtering methods for personalized information delivery.

The Infoscope project developed by Stevens (1992) investigated the use of agents to maintain models of user interest for sorting Usenet messages. The system consists of rule-based agents which observe usage patterns and make suggestions to the user. The agents monitor the contents of those messages which are deemed interesting or uninteresting, make statistical correlations, and suggest changes to the user.

Sheth (1994) proposed the personalized information filtering system News Tailor (Newt) for Usenet newsgroups. The system uses genetic algorithms to control a population of user profiles which represent a user's interest in different topics. The profiles are evaluated on the basis of relevance feedback provided by the users.

Yan and Garcia-Molina (1995) developed the Stanford Information Filtering Tool (SIFT), which provides a service for content-based filtering of Usenet news articles. Users can subscribe to this service by submitting profiles in terms of keywords that describe their interests. They then passively receive new, filtered information geared to their profiles. SIFT offers a further option: updating of user profiles by giving relevance feedback to interesting articles.

Lang (1995) implemented the NewsWeeder system for filtering Usenet news articles. The systems lets users rate their interest levels for each article being read. It then uses text classification algorithms to automatically learn user profiles based on these ratings.

Lieberman (1995) developed Letizia, which acts as an advance scout for Web browsing by watching a user browsing the Web and trying to learn which topics the user is interested in. While the user is reading a Web page, Letizia searches adjacent pages and attempts to anticipate pages the user might be interested in. The agent automates a browsing strategy consisting of a breadth-first search augmented by heuristics inferring user interest from browsing behavior.

Armstrong *et al.* (1995) implemented a system called WebWatcher, which assists users in locating information on the Word Wide Web by taking keywords from the users, suggesting hyperlinks, and receiving evaluation. The idea is to automatically customize the system to individual users by taking each user interaction as a training example. WebWatcher thus learns by observing the links a user follows on the Web and suggests interesting hyperlinks whenever it is confident enough of its success. Personal WebWatcher, described by Mladeniȁ (1996), avoids involving the user in its learning process. It does not request any keywords or user opinions about web pages. The idea proposed for WebWatcher is further automated.

Moukas (1996) proposed the information discovery and filtering system Amalthaea. It is a co-evolution model of information filtering agents that adapt to different user interests and of information agents that monitor and adapt to various online information sources.

Pazzani and Billsus (1997) developed Syskill & Webert, a software agent that learns to rate pages on the Web, deciding which pages might interest a user. The user rates explored pages on a three-point scale, and Syskill & Webert learns a user profile by analyzing the information on each page.

Mostafa *et al.* (1997) developed a system based on smart information filtering technology for electronic resources (SIFTER). This system focuses on efficiently handling uncertainties associated with changing interests of the user and the dynamic document stream. The system automatically learns to filter documents according to a specific user interest with only limited user intervention in the form of optional relevance feedback for documents.

Balabanovic̀ (1997) introduced the adaptive web page recommendation service Fab. This multiagent system tries to take advantage of both content-based and collaborative filtering. Collaborative properties are achieved through interactions between content-based filtering agents. A list of top-ranked recommendations for web pages is presented to users according to their profile upon request. Any feedback provided by the users is then used to refine the corresponding profiles.

Mock and Vemuri (1997) designed the intelligent news filtering organizational system (INFOS) to reduce the user's search burden while browsing a large number of messages by automatically classifying data as relevant or non-relevant based upon user interests. The system is founded on a hybrid technique that learns on the basis of both features taken from input articles and collaborative features derived from other users.

Billsus and Pazzani (1999) developed the personal news agent News Dude, which is designed to become part of a system that uses synthesized speech to read news stories to a user who does not have access to a computer. The system can adapt to the user's preferences and interests based on voice feedback. Long-term and short-term interests of users are modeled by two different approaches. Also, the system gives explanations why it has presented certain news stories. Users can also review these explanations in order to tailor the system to their needs.

2.5 Comparison of Related Tasks

Marchionini describes *information seeking* as a general process "in which humans purposefully engage in order to change their state of knowledge."[42] As such, it is used as an overarching term to describe "any processes by which users seek to obtain information from automated information systems."[43] Following Oard (1997), we compare information filtering to other common information seeking processes for which the decomposition into an information collection, detection, and presentation component as depicted in Figure 2.1 (p. 6) is also appropriate. These processes are summarized in Table 2.2.

The two prime distinguishing characteristics we consider are the users' information needs and the nature of the documents, or their sources. One key feature of both information need and information sources is whether or not they tend to remain unchanged over a

[42]From Marchionini (1997), p. 5.

[43]From Oard (1997), p. 141.

Process	Information Need	Information Sources
Information Filtering	stable & specific	dynamic & unstructured
Alerting	stable & specific	dynamic & structured
Data Mining	stable & specific	stable
Information Retrieval	dynamic & specific	stable & unstructured
Database Access	dynamic & specific	stable & structured
Exploration	broad	varied

Table 2.2: Examples of information seeking processes as taken from Oard (1997), p. 145.

longer period of time, i.e. whether they are stable or dynamic. Another central feature pertaining to the information sources is the way the data is formatted: Whether data is available in a structured or unstructured form has a great impact on the techniques that can be used for seeking relevant information. Whether the user is actually seeking information to satisfy a specific need or is just looking around with no concrete goal is another issue vital to information need.

Closely related to information filtering is the established research area of *information retrieval*. "Information retrieval is concerned with the processes involved in the representation, storage, searching, finding, and presentation of information which is relevant to a requirement for information desired by a human user."[44] At the abstract level of information seeking, both information retrieval and information filtering are concerned with selecting relevant information from a large source of documents with respect to a particular user interest. In fact, when considering information retrieval as a very general information selection technique, information filtering can be viewed as a special case in which the information space is very dynamic. Belkin and Croft (1992) provide a useful description of the difference between information filtering and information retrieval. In information retrieval, the collection of documents is assumed to be relatively static, while the user submits queries that often change, i.e. are dynamic. In contrast, in information filtering, the 'query' is relatively stable, while the collection of documents is assumed to be an incoming stream of documents and, hence, to be dynamic. Consequently, information filtering is concerned more with the *distribution* of documents to groups or individuals, while information retrieval is concerned rather with the *collection and organization* of documents. Thus, information filtering and information retrieval can be considered to be "two sides of the same coin. They work together to help people get the information needed to perform their tasks."[45] Figure 2.4 illustrates this relationship.

The process of *alerting* is also very similar to information filtering. Here, the information need is assumed to be relatively stable with respect to the rate at which the information changes. The difference is that, in an information filtering process, the documents change rather than the information itself, e.g. in the form of structured attribute-value pairs. Thus, alerting can be considered as the database analogue of information filtering. A typical example is a system that monitors a complex machine and gives an alarm whenever a

[44]From Ingwersen (2000), also see Ingwersen (1992), p. 49.
[45]See Belkin and Croft (1992), p. 32.

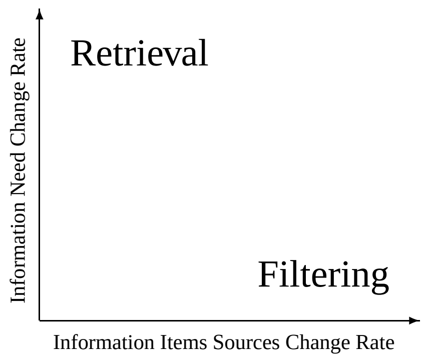

Retrieval
Filtering
Information Need Change Rate
Information Items Sources Change Rate

Figure 2.4: The relationship between information filtering and information retrieval based on the change rate of a user's information need and the collection of documents (following Oard (1997), p. 143).

certain parameter exceeds pre-specified limits. For instance, this idea can be applied in an information filtering system to inform a user whenever an email from a specific user arrives. We will use similar techniques for detecting whether there is a change in the document stream which the user should know about. This is an issue of quality control in information retrieval and will be discussed in Chapter 6.

Data mining is another research area that is closely related to information filtering. Data mining, or more generally *knowledge discovery in databases*, is "the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."[46] Classical data mining tasks are classification, regression, clustering, segmentation, concept description, data description and summarization, deviation detection, and dependency analysis. We view information filtering as a text classification problem. And with text classification being a special kind of classification problem, namely one where the data is text, information filtering can also be considered as a data mining task in a very general sense. And, in fact, information filtering and data mining share many common techniques from feature selection to learning algorithms for classification. Note, however, that data mining traditionally considers structured data residing in relatively stable databases. In contrast, information filtering involves the classification of multi-media and, hence, only partially structured or even unstructured data. For text domains, this fact can be emphasized by using the term *text mining*.

Retrieving information from a database is another well-known information seeking process. Information filtering and information retrieval select documents which contain information that is potentially relevant for a user. The *database access* process, however, actually responds with information. Another difference is that database access involves querying from structured and rather stable data tables. For instance, querying an online library catalog database to find the author of a particular book is a database access process. Yet, occasionally using the same database "to discover whether any new books on a certain topic have been added to the collection by searching for keywords in the title field would be an information filtering process."[47] Applying database systems in this sense can support information filtering.

[46]From Fayyad, Piatetsky-Shapiro and Smyth (1996), p. 6.

[47]From Oard (1997), p. 145.

Finally, *exploration* is a less well-structured process and rather loosely defined term. According to Oard (1997), users may choose to explore either static or dynamic information sources. Hence, exploration has similarities to both information filtering and information retrieval. "'Surfing the World Wide Web' is an example of exploring relatively static information, while reading an online newspaper would be an example of exploring relatively dynamic information."[48] Oard mentions as a main distinguishing feature that the users' interests in exploration are assumed to be broader than for information filtering or retrieval processes. However, a precise definition of 'broader' is often just a matter of judgement and is, hence, difficult to give.

2.6 Summary

The objective of information filtering is to reduce the user's information load with respect to their areas of interest. Hence, an information filtering system should automatically screen out documents which are not relevant to a user's interest, thereby presenting only relevant documents to the user. In this dissertation, we focus on content-based filtering of textual documents. This turns out to be a binary text classification problem where the classification decision is whether or not a document is relevant for a particular user. We will cover techniques for text classification in the following chapter.

[48]From Oard (1997), p. 146.

Chapter 3

Text Classification

This chapter gives an introduction to text classification. We define the task of text classification and show its relationship to the task of information filtering. Our goal is to use machine learning techniques for automatic text classification. In this context, we identify two basic subtasks which we examine closely: representing textual documents in a way that is appropriate as input for machine learning algorithms and the application of these learning algorithms proper.

3.1 Definition and Terminology

The task of *text classification* is to classify documents into a fixed number of two or more predefined classes.[1] A class is considered a semantic category that groups documents that have certain properties in common. Generally, a document can be in multiple, exactly one, or no classes. Yet, with the task of information filtering in mind, i.e. the classification of documents as either relevant or non-relevant, we assume that each document is assigned to *exactly one* class. More precisely, we pose the text classification problem as follows.

Definition 3.1.1 (Text Classification)
Assume a space of textual documents $\mathcal{D}$ and a fixed set of k classes $\mathcal{C} = \{c_1, \ldots, c_k\}$, which implies a disjoint, exhaustive partition of $\mathcal{D}$. Text classification is a mapping, $h : \mathcal{D} \mapsto \mathcal{C}$, from the document space onto the set of classes.

With the enormous growth of online information available through sources such as the World Wide Web, the problem of automatically classifying text documents into predefined classes is of eminent importance in many information organization and management tasks, and information filtering is one of these tasks. Our goal is to use machine learning techniques to automate the process of text classification. Typically, these techniques learn

[1]This task is also referred to as *text categorization*. Please note that 'classification' is an ambiguous term in machine learning, applied statistics, information retrieval, and other fields and commonly refers to processes that group entities. See Lewis (1992b) for a discussion about different text classification tasks. In this work, we consider text classification and text categorization to be synonymous.

classifiers that can predict the class labels of new, previously unseen documents, if given a set of training examples for which the class labels are known. This problem is known as supervised learning. Formally, we define the learning task in text domains, which we refer to as *text learning*, as follows.[2]

Definition 3.1.2 (Text Learning Task)
Assume a set of n labeled training documents $D^\star = \{d_1, \dots, d_n\} \subset \mathcal{D}$ and a fixed set of k classes $\mathcal{C} = \{c_1, \dots, c_k\}$ as above. Let $T : \mathcal{D} \mapsto \mathcal{C}$ be the target function that assigns each training document $d \in D^\star$ its true class label $T(d)$. The objective of the text learning task is to induce a classifier, the hypothesis $h : \mathcal{D} \mapsto \mathcal{C}$, from $D^\star$, which approximates the target function T well with respect to a given effectiveness measure.

According to the *inductive learning hypothesis*, any classifier "found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples."[3] In other words, the classifier can be used to predict the class labels of new, previously unseen examples, i.e. documents in the context of text classification.

In this setting, the problem of automatically classifying documents falls into two phases. First, we apply supervised learning algorithms to construct a classifier in the *learning* or *adaptation phase* as posed in Definition 3.1.2. The classifier constructed can either represent each class simultaneously or, in particular if we are dealing with more than two classes, treat each class as a separate binary classification problem, where each binary problem answers the question of whether or not a document should be assigned to the corresponding class. Finally, we can use this classifier to predict the class label of new documents in the *classification phase*. In the following, we focus mainly on the learning phase and only note that the subsequent classification of new documents is straightforward with the techniques involved in the learning phase.

The learning phase is further divided into two subtasks as follows. Because plain textual documents are generally not suitable as input for machine learning algorithms, we, first, need some way of preprocessing that transforms text into a format appropriate as input for machine learning algorithms. We will look at techniques for representing textual documents in Section 3.2. Second, provided with a suitable representation for documents, in Section 3.3 we turn to learning algorithms that are particularly well suited for text domains. We will refer to these as *text learning algorithms*, or *text learners* in short.

Recall Definition 2.1.1 (p. 5) for information filtering. For the two-class problem ($k = 2$), with $c_1 \equiv non$ and $c_2 \equiv rel$ corresponding to rejecting non-relevant and accepting relevant documents, respectively, we observe that $h = f_\psi$ with respect to a particular user interest ψ. That is, information filtering is an instantiation of text classification with two classes. Furthermore, in Section 2.2 (pp. 10 ff.), we presented a general framework for information filtering from an applicational point of view, where the core of a filtering system essentially consists of a document representation function ρ, a profile acquisition function π, a comparison function κ_s for matching a profile against document representa-

[2]For example, see Joachims (1997a).
[3]From Mitchell (1997), p. 23.

tions, and a decision function τ_s for mapping classification scores onto hard decisions.

In the light of machine learning, we actually try to construct these four functions during the learning phase. In the classification phase, however, these functions are applied to new, unseen documents. In particular, representing text corresponds to finding a suitable function ρ, and learning a classifier subsumes functions π, κ_s, and τ_s.

3.2 Text Representation

The aim of the text representation task is to transform a textual document into a format that is suitable as input for machine learning algorithms.[4] We use the *vector space model*[5] to represent documents because it is widely recognized as an effective representation for documents in the information retrieval community.[6] Also, it can be tailored to mimic other well-known indexing models such as binary or probabilistic indexing as we will see in Subsection 3.2.4. In the vector space model, each document is identified by a feature vector in a space in which each dimension corresponds to a distinct index term. Note, therefore, that we will use the terms *feature* and *index term* interchangeably. Generally, the set of index terms can be provided either manually by a human indexer or automatically by a program based on a specific document collection. In this dissertation, we consider only automatic indexing, and, typically, the document collection on which this indexing is based is the set of training examples that will serve as input for the learning algorithms. A given document vector has, in each component, a numerical value to indicate its importance. This value is commonly determined as a function of how often the corresponding term appears in the particular document and how often it appears in total in the document collection. By varying this function, we can produce different term 'weightings' which can then be interpreted as different indexing models. The resulting representation of text is equivalent to the attribute-value representation, which is commonly used in machine learning.[7]

Definition 3.2.1 (Vector Space Model)
Let $d \in \mathcal{D}$ be a textual document. The representation of d is the document vector $\mathbf{d} = \rho(d) = (w_1, \ldots, w_m)^T \in \mathcal{R} = \mathbb{R}_+^m$, where each dimension corresponds to a distinct term in the document collection and w_i denotes the weight of the i-th term. The set of these m index terms, $\mathcal{V} = \{t_1, \ldots, t_m\}$, is referred to as the vocabulary.

The vector space representation abstracts from the sequence in which index terms appear in a document. Note, however, that we have not yet defined what is to be considered as *index terms.* This issue will be treated in Subsection 3.2.2. Only note at this point that plain words are frequently used as index terms. Figure 3.1 shows the steps involved in transforming the training documents and a new document into feature vectors when plain

[4]See Lewis (1992b) for a thorough survey about research on text representation.
[5]See Salton, Wong and Yang (1975).
[6]See Yan and Garcia-Molina (1994).
[7]See Joachims (1997a).

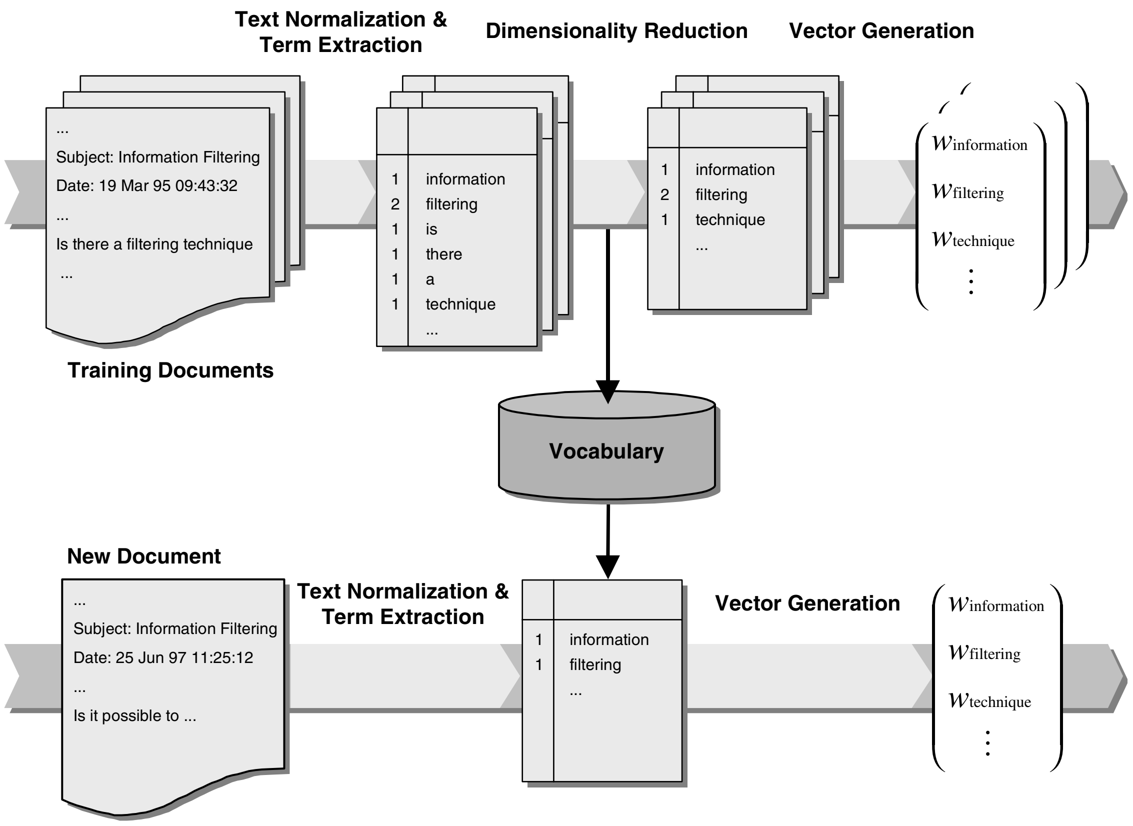


Figure 3.1: The steps that are involved in transforming textual training documents into feature vectors (top). During this step, the set of index terms, which is also referred to as the *vocabulary*, is constructed. Only index terms occurring in the vocabulary are used when transforming a new document into a feature vector (bottom).

words independent of the order of their appearance are used as index terms. For training documents, these steps are as follows.

1. Text normalization.
2. Term extraction.
3. Dimensionality reduction.
4. Vector generation.

The *text normalization* step transforms any type of document into a sequence of word tokens, from which a sequence of possible index terms is created for each document during the *term extraction* step. How the output of the term extraction step is used depends on whether or not new documents or training documents are processed. Note that, in the latter case, we still need to construct the vocabulary. Consequently, all distinct index terms of the training documents are merged to generate a set of candidate index terms which may potentially be used as vocabulary. This is sometimes referred to as *term* or *feature generation*. Since the resulting set of index terms tends to be very large, we aim at reducing the size of the vocabulary in the *dimensionality reduction* step. The index terms in the reduced vocabulary finally correspond to the vocabulary as given in Definition 3.2.1. In case the vocabulary is already established, i.e. when new documents are processed, the term extraction step is assumed to output only index terms which exist in the vocabulary. Finally, the *vector generation* step evaluates weights for all index terms of any given document. These four preprocessing steps are discussed in more detail in the following subsections.

3.2.1 Text Normalization

In general, input to a filtering system and, thus, to the text learning algorithm and the resulting classifier can be regarded as a file representing text. The objective of the *text normalization* step is to convert the input files into a sequence of linguistic items, which are referred to as word *tokens*. In the subsequent term extraction step, these tokens will be used to generate meaningful features, the so-called index terms. There are two steps in the text normalization process.

First, textual components from different file formats have to be recognized. The definition of the file format must be known in order to extract the textual components. For instance, files may be electronic mails, USENET newsgroup articles, $\TeX$, postscript, or HTML documents. When processing electronic mails, for example, usually the subject field and the text body are extracted and, for HTML documents, all tags are removed from the input. On the basis of the text components extracted, the text is parsed into a sequence of tokens which are strings of characters delimited by whitespace.[8] For this dissertation, we assume that we are provided with suitable parsers that can transform a given file into a sequence of tokens.

Second, the resulting sequence of tokens can further be normalized depending on the application.[9] As a rule, all letters are converted to lower-case to avoid capitalization constraints and punctuation marks at the end of tokens are removed. In addition, tokens that contain any non-alphanumeric characters may be deleted. Even tokens containing numeric characters are often omitted. Alternatively, all digits could be mapped onto a predetermined digit, for instance onto '1'. Following another approach to text normalization, named entities such as people, locations, organizations, and product names could be identified and marked as individual tokens. For instance, the text fragment "A␣160" could be identified as a certain type of car. In this case, the digits contained would not be normalized or removed as described above. Note that such a normalization step would generally require domain- and application-specific knowledge and will therefore not be considered any further in this dissertation.

3.2.2 Term Extraction

Assume a sequence of normalized tokens for a document is provided by the text normalization step. The task of the *term extraction* step is to produce a sequence of index terms

[8]Whitespace characters are, for instance, space, tab, and newline.

[9]For example, see Fox (1992), pp. 103–104.

based on these tokens. The way this sequence of index terms is used depends on whether or not we are processing training documents and still need to construct the vocabulary.

If, on the one hand, the vocabulary has already been created, any index terms extracted which are not in the vocabulary are omitted during term extraction and, for each document d_j under consideration, the term frequency vector $\mathbf{d}_{tf} = (tf_d(t_1), \ldots, tf_d(t_m))^T$ is created, where $tf_d(t)$ denotes the number of times term $t \in \mathcal{V}$ appears in document d. Given these vectors, we might apply some technique for the purpose of dimensionality reduction and can then proceed with the vector generation step.

On the other hand, i.e. when the vocabulary has not yet been established, the outputs of the term extraction step for all training documents are merged to generate a set of distinct index terms $\hat{\mathcal{V}} = \{\hat{t}_1, \ldots, \hat{t}_{\hat{m}}\}$, which may potentially constitute the vocabulary $\mathcal{V}$. Depending on the term definition and on the size and domain of the collection of training documents, it is quite common that the number $\hat{m}$ of these preliminary index terms is on an order of 10^3 to 10^6 after term generation. Since the application of machine learning algorithms is often impractical when the number of features is very large, we aim at reducing the final vocabulary size in the subsequent *dimensionality reduction* step.

According to Definition 3.2.1, the dimensions of the vector space correspond to distinct index terms in the document collection under consideration. But exactly what do we consider to be an *index term*? A fundamental challenge in natural language processing and understanding is that information or meaning conveyed by language always depends on context. Words per se, for example, may already have different meanings. Moreover, language cannot always be taken literally. One and the same phrase may have different meanings depending on the context in which it is used. These examples show that trying to represent natural language documents by means of a set of index terms is a challenging task. Different linguistic approaches to this try to capture, or ignore, to a certain extent meaning with respect to context. We divide these approaches into five levels:[10]

1. *Graphemic level:* analysis on a sub-word level, commonly concerning letters.
2. *Lexical level:* analysis concerning individual words.
3. *Syntactic level:* analysis concerning the structure of sentences.
4. *Semantic level:* analysis related to the meaning of words and phrases.
5. *Pragmatic level:* analysis related to meaning regarding language-dependent and language-independent, e.g. application-specific, context.

Note that the first two levels operate solely on plain statistical facts about text, i.e. basically on frequencies of letter combinations or words, which we refer to as term frequencies. Text representations based on these term frequencies cannot completely capture the

[10]This division is certainly not complete. For instance, a morphological level for an analysis concerning the structure of words or a phrasal level for an analysis of groups of words on a sub-sentence level could be considered. Furthermore, there may be interchanges between different levels. For example, ambiguities at one level might be resolved with approaches from a higher level.

meaning of documents. In fact, there is only a weak relationship between term occurrences and document content, the term frequencies of a document are assumed to merely hint at its content.[11] In contrast, higher levels of text analysis try to capture more semantic content by exploiting an increasing amount of contextual information such as the structure of sentences, paragraphs, or documents. In general, natural language processing techniques can provide a much richer representation through a syntactic and semantic analysis of documents.

Intuitively, it seems reasonable to assume that more complex text representations as obtained from higher levels of text analysis would lead to more effective text classifiers.[12] With an infinite amount of text revealing information about the index terms to be extracted, this may actually be true. In real-world problems, however, the number of documents available for text analysis is limited. Moreover, as more complex term definitions lead to more complex text representations, the dimensionality of the feature space increases correspondingly. And, with a limited number of training documents, inducing an accurate classifier is much harder. The problem of having too many features relative to the number of training data is usually referred to as the *'curse of dimensionality'*.[13] Research in information retrieval shows that "[...] all reasonable text representations have been found to result in very similar effectiveness on the retrieval task."[14] Lewis refers to this as the *Equal Effectiveness Paradox*.

Nevertheless, this finding builds on results that are mainly achieved with documents in English. For other languages, deeper linguistic analysis may be more beneficial. In German, for instance, there are many compound words, such as 'Textklassifikationsverfahren', which can cause an enormous increase in the number of features. Hence, techniques that recognize these compounds and split them into smaller parts could remedy this problem.[15] We conclude that the benefit we may get from linguistic analysis is strongly domain- and language-dependent.

From the above discussion we see that choosing a suitable level of text analysis on which to base the term definition is always a trade-off between semantic expressivity and representational complexity. This choice will have a wide impact on a learning algorithm's ability to generalize, as we will further discuss in Section 3.3. Simple term definitions are dominant in information retrieval.[16] Typically, these involve text analysis approaches from the first two levels, and, at times, some syntactical information may also be incorporated. In the following, we describe three widely used term definitions which basically just exploit plain statistical facts about text.[17] The first approach depends on *letter n-grams* and, thus, corresponds to text analysis at the graphemic level. The remaining two approaches, namely single *words* and word *phrases*, are best described as lexical text analysis. Note, though, that the latter may also use some syntactic information.

[11] See Blair (1992), p. 203.

[12] For example, see Blair (1992), p. 203.

[13] See Subsection 3.2.3 for more details.

[14] See Lewis (1992b), pp. 6–7.

[15] See Neumann and Schmeier (1999), for example.

[16] For example, see Apté *et al.* (1994), p. 236.

[17] In Subsection 3.2.3, we will consider some reparameterization techniques for dimensionality reduction which actually try to capture semantic information within document collections.

- *Letter n-grams.* Overlapping, contiguous n letter subsequences of words are known as n-grams, where n is a positive integer.[18] Note that sometimes the term 'n-gram' is used to refer to sequences of words of length n.[19] Below, we will refer to these multi-word terms as phrases. In n-gram analysis, trigrams or quadgrams, i.e. $n = 3$ or $n = 4$, are commonly used, but other values are also possible. For instance, the word 'token' consists of the trigrams 'tok', 'oke', and 'ken'. Note that it is also possible to include a space to the left and the right of a word when generating n-grams. In the case of the word 'token', we would additionally obtain the trigrams '␣to' and 'en␣'. Taking into account only the 26 letters of the English alphabet, there are $26^3 = 17576$ distinct trigrams and $26^4 = 456976$ quadgrams. "The larger n, the more accurately the distances between n-gram vectors correspond with the semantical distances of the original documents."[20] However, higher values of n lead to a huge number of possible terms, whereas n-grams with $n < 3$ do not provide sufficient word-syntactical information.[21] The advantage of n-grams is that the set of possible terms is fixed and known in advance. Furthermore, n-grams are language-independent and are quite robust to both morphological variations and spelling variations and mistakes. N-grams are easy to calculate, but the resulting representation is difficult to analyze by humans.

- *Words.* Information retrieval research shows that single words work well as features.[22] Thus, a very common approach is to use each token, i.e. generally each word, as it is. Since the vector space model does not consider the sequence in which words appear in a document, we actually treat a document as a "bag of words".[23] Therefore, the vector space representation based on words as features is also referred to as the *bag-of-words* model. Obviously, this term definition is language-independent and computationally very efficient. However, a disadvantage is that each inflection of a word is a possible feature and the number of possible features can thus be unnecessarily large. Moreover, morphological variants of a word are not recognized as being similar. A remedy is to reduce each word to its *word stem* in order to conflate morphological variants. However, note that this makes term extraction language-dependent and less efficient. We will discuss stemming as a means to reduce the dimensionality of the feature space in the next subsection.

- *Phrases.* Combinations of tokens are referred to as phrases. Using phrases is justified by the observation that, especially in English, many expressions are in fact multi-word terms such as 'data mining' or 'information filtering'. Typically, only domain-specific phrases are considered, because, otherwise, the number of possible terms would increase drastically. Multi-word terms could be identified as frequently co-occurring sequences of words,[24] or they could be detected through

[18]See de Heer (1982), or Cavnar and Trenkle (1994).
[19]See Fürnkranz (1998).
[20]See Tauritz *et al.* (2000).
[21]See Teufel and Schmidt (1988).
[22]See Salton and Buckley (1988), p. 515. Also see Dumais *et al.* (1998) for text classification.
[23]See Lewis and Ringuette (1994), p. 82.
[24]See Fagan (1987), pp. 36–73, or Cohen and Singer (1996).

the application of natural language processing, which is then also referred to as *syntactic phrase indexing*.[25] Another approach to phrase identification is to manually provide a set of phrases for a particular domain.[26] Obviously, using phrases tends to be domain-dependent. Results with using phrases as index terms are differing: some researchers report an improvement in classification accuracy when using phrases,[27] whereas others do not achieve more effective classifiers in comparison to just using single words.[28] Sahami argues that "some of this discrepancy can be accounted for by the expressivity of the models used for learning. For example, models that do not capture co-occurrence information between words may stand to benefit from multi-word features which explicitly express such dependencies. On the other hand, models capable of learning word co-occurrence information may not gain anything from the inclusion of multi-word phrases and may actually be hindered by having to estimate additional parameters for such terms."[29] Note that this argument supports the aforementioned *Equal Effectiveness Paradox*. Since using phrases does not consistently lead to more effective classifiers, we will not use phrases as index terms in this dissertation.

In the experiments to be conducted in this dissertation, we will primarily use single words as index terms. Note that single words independent of the order in which they appear in a document are frequently used as index terms for two reasons. First, this text representation can be implemented very efficiently. Second, and even more crucial, the representation does not build on domain- or language-specific knowledge and can, thus, be directly applied to any document collection.

3.2.3 Dimensionality Reduction

The bulk of the dimensionality reduction step is applied only in case the vocabulary needs to be constructed, i.e. when the training documents are being processed. Recall that the set of possible index terms $\hat{\mathcal{V}} = \{\hat{t}_1, \ldots, \hat{t}_{\hat{m}}\}$ resulting from the initial term extraction step for the training documents is usually very large. The objective of the dimensionality reduction step is to reduce the number of features that are finally used to represent documents, under the constraint that the resulting set of features should still discriminate the different classes well. As a result, we obtain a smaller set of index terms, the vocabulary $\mathcal{V} = \{t_1, \ldots, t_m\}$, where $m \leq \hat{m}$ denotes the number of index terms that remain.

Controlling the dimensionality of the vector space is essential for two reasons. The complexity of many learning algorithms depends crucially not only on the number of training examples but also on the number of features. Thus, reducing the number of index terms may be necessary to make these algorithms tractable.[30] Also, although more features can

[25]See Fagan (1987), pp. 74–182, Lewis (1992a), Lewis (1992b), pp. 46–55, or Fürnkranz *et al.* (1998).
[26]See Spertus (1997) and Sahami *et al.* (1998).
[27]See Cohen and Singer (1996), for example.
[28]See Dumais *et al.* (1998), for example.
[29]From Sahami (1998), p. 21.
[30]See Dash and Liu (1997), for example.

be assumed to carry more information and should, thus, lead to more accurate classifiers, a larger number of features with possibly many of them being irrelevant may actually hinder a learning algorithm constructing a classifier.[31] The problem of having too many features is often referred to as the *'curse of dimensionality'*.[32] Given a fixed number of training examples, theoretical and empirical results in machine learning show that there is often "a maximum number of features beyond which effectiveness of an induced classifier will begin to decline."[33] Hence, removing less informative features may actually increase classification performance.

We use the term *dimensionality reduction* to subsume any techniques that aim at controlling the dimensionality of the vector space. This includes *feature selection* techniques which attempt to find a suitable subset of the given feature set by explicitly including particular index terms in the vocabulary or excluding them. Note that feature selection is applied only in case the training documents are being processed. It affects the representation of new documents in that only terms existing in the vocabulary are taken into consideration. In addition, we consider techniques based on *reparameterization*, which is the process of constructing new features by taking combinations and transformations of the original features.[34] Hence, these techniques actually replace existing index terms by new terms in order to reduce the dimensionality of the vocabulary. Note that the mapping which reparameterizes the feature space is constructed on the basis of the training documents. Yet, this mapping is then applied not only to training documents but also to new ones. Both feature selection and reparameterization can be further divided into linguistic and thus language-dependent approaches and language-independent statistical approaches. Typically, some of these approaches are used in combination.

Feature Selection

Selection techniques for dimensionality reduction take as input a set of features and output a subset of these features, which are relevant for discriminating among classes.[35] Ideally, we would like to find the subset among all possible subsets that allows a particular learning algorithm to induce the best possible hypothesis with respect to a given effectiveness measure.[36] So, feature selection can be regarded as an optimization problem where the search space corresponds to the power set of the set of all $\hat{m}$ features, i.e. where there are $2^{\hat{m}}$ candidate subsets to look at.[37] Obviously, this renders an exhaustive search intractable since the number of features is usually very large in text domains. Therefore, feature selection must be guided by heuristics.[38] Langley (1994) identifies four dimensions along which search heuristics can vary, namely the starting point, the organization of the search, the evaluation strategy, and the halting criterion. With the subsequent learning task in

[31] See Lewis (1992b), p. 14.
[32] See Duda and Hart (1973), p. 95.
[33] From Lewis (1992b), p. 41.
[34] See Schütze, Hull and Pedersen (1995).
[35] See Dash and Liu (1997) for a survey on feature selection for classification.
[36] See Kohavi and John (1997), p. 276, for the notion of an optimal feature subset.
[37] See Blum and Langley (1997).
[38] See Schürmann (1996), p. 270.

mind, especially the evaluation scheme is of great importance. John *et al.* (1994) distinguish filter and wrapper approaches as two general alternatives with respect to subset evaluation. In this, filter approaches are independent of the learning algorithm, whereas wrapper approaches utilize the learning algorithm proper as part of the evaluation scheme. With respect to classification effectiveness, wrapper approaches should generally be preferred to filter approaches because the usefulness of features crucially depends on the bias of the learning algorithm.[39] In contrast, it is computationally expensive to apply the learning algorithm once, or even v times in case effectiveness is estimated by means of v-fold cross-validation on the training documents,[40] for each subset being considered.[41] Note that, in text classification, this disadvantage is particularly serious because of the large number of features. In the following, we therefore focus on a simple filter approach on which most feature selection techniques are based in text domains.[42]

In this simple filter approach, each feature is treated independently. For each feature, we evaluate a score on the basis of which we can decide whether to include a feature in the vocabulary or to exclude it. The final vocabulary is established either by selecting all features whose score is above or below a predetermined threshold or by selecting the m best features, i.e. either the m largest or smallest features according to score magnitude. Determining an appropriate threshold or the target number m of features, remains an open issue. Note that, in a general sense, empirically determining these parameters based on the effectiveness on some, possibly held-out, training documents actually resembles a wrapper approach.

We start with a widely applied linguistic approach known as *stop word elimination*. Then, we discuss several frequently used numerical measures for evaluating term quality with respect to its ability to discriminate among classes as discussed above.[43] The first four measures ignore the class labels of the documents under consideration, whereas the others crucially depend on the class labels in order to identify promising features. Typically, all measures are based on some frequencies in which terms occur in documents, classes, or the entire document collection. Table 3.1 gives an overview of the document numbers and term frequencies used for the evaluation of the numerical term quality measures. Note that the term frequencies (tf) reflect the actual number of term occurrences in particular documents, whereas the other document numbers depending on terms are based on binary indicators of term presence or absence in the corresponding documents.

The following functional relationships hold among the document numbers for the entire document collection and the partitioning into k classes:

$$n = \sum_{i=1}^{k} n_{c_i} \tag{3.1}$$

$$n(t) = \sum_{i=1}^{k} n_{c_i}(t) \tag{3.2}$$

[39]See John *et al.* (1994).
[40]See Breiman *et al.* (1984), pp. 75–77.
[41]See Blum and Langley (1997).
[42]See Mladenič (1998).
[43]For example, see Yang and Pedersen (1997), and Mladenič and Grobelnik (1998).

n	Number of documents in the training set $D^\star$
n_{c_i}	Number of documents with class label c_i
$n(t)$	Number of documents in which term t appears at least once
$\bar{n}(t)$	Number of documents in which term t does not appear
$n_{c_i}(t)$	Number of documents with class label c_i in which term t appears at least once
$\bar{n}_{c_i}(t)$	Number of documents without class label c_i in which term t does not appear
tf	Number of occurrences of all terms in all documents
$tf(t)$	Number of occurrences of term t in all documents
$tf_{d_j}(t)$	Number of occurrences of term t in document d_j

Table 3.1: Document numbers and term frequencies used for evaluating term quality measures.

The complement document numbers, $\bar{n}(t)$ and $\bar{n}_{c_i}(t)$, are defined as:

$$\bar{n}(t) = n - n(t) \tag{3.3}$$

$$\bar{n}_{c_i}(t) = n_{c_i} - n_{c_i}(t) \tag{3.4}$$

Finally, we observe the following relationships among the term frequencies:

$$tf = \sum_{i=1}^{\hat{m}} tf(t_i) \tag{3.5}$$

$$tf(t) = \sum_{j=1}^{n} tf_{d_j}(t) \tag{3.6}$$

where $\hat{m}$ denotes the number of potential features in the vocabulary $\hat{V}$ before any particular dimensionality reduction step.

Stop Word Elimination. When analyzing a language, we can often observe that there are many words which occur in all documents without regard to classes and, thus, have little or no inherent topical content.[44] These high frequency words are commonly referred to as *stop words*.[45] Typically, these are function words such as articles, prepositions, conjunctions, and pronouns, which provide structure in language rather than content.[46] Since stop words lack discriminative power, it is reasonable to eliminate these words from the set of possible index terms $\hat{\mathcal{V}}$ and, as a result, to reduce the dimensionality of the associated vector space. Note that in case of n-gram based index terms, stop word elimination should, if desired, take place before the term extraction step based on the normalized word tokens.

[44]See Salton and McGill (1983), p. 71, or van Rijsbergen (1979).

[45]For example, see van Rijsbergen (1979), p. 17.

[46]See Sahami (1998), p. 26.

a	be	each	if	last	near	that
about	but	else	in	late	no	the
all	by		is	like		they
an		for	it		of	to
and	did	from	into	many	often	
are	do	further	itself	much	on	with
as	down			more	once	which
at	during	get	just	must	or	whether

Table 3.2: Excerpt from a list of frequently used stop words.

There are two ways to obtain a list of stop words, or a *stop list* for short. One common approach is to manually establish a stop list. Many stop lists for English text can be found on the Internet[47] and in the literature.[48] Typically, these lists contain several hundred stop words. Table 3.2 shows an excerpt from a list of frequently used English stop words. In some applications, it may be useful to provide domain-dependent stop lists in addition to or even instead of a general stop list. Note that removing words on the basis of a stop list does not properly fit in with the notion of the filter approach. Yet, this kind of stop-word removal can easily be cast as a filter approach, simply by defining a function that responds to each term t with a certain value indicating whether or not t is considered a stop word. Stop words would then be eliminated according to this value.

A second approach to stop-word elimination is to construct the stop list automatically based on the document collection under consideration. To do so, a common approach is to consider as stop words—and to eliminate from the feature set—either the i, say, most frequent terms or all terms with a frequency above a given threshold.[49] The threshold can be defined either manually in advance or based on the frequency histogram of the term distribution. This approach eliminates stop words that are specific to the document collection under consideration and is, therefore, domain-specific. So, not only commonly accepted stop words, but also other words such as nouns, verbs, and adjectives may be eliminated. Note that automatically identifying stop words corresponds to term frequency thresholding with an upper threshold, which we will discuss next.

Term Frequency Thresholding and Zipf's Law. A very simple selection heuristic is to eliminate all the terms whose frequencies are either above a pre-specified upper threshold or below a pre-specified lower threshold. The assumption that the frequency of term occurrences is an appropriate measure of term significance originates from the following observation made by Luhn:[50]

> *The justification of measuring word significance by use-frequency is based on the fact that a writer normally repeats certain words as he advances or varies his arguments and as he elaborates on an aspect of a subject. This means of emphasis is taken as an indicator of significance.*

[47]For example, see `ftp://ftp.cs.cornell.edu/pub/smart/english.stop`.
[48]See van Rijsbergen (1979), pp. 18–19, or Fox (1992), pp. 114–115, for example.
[49]See Lang (1995), for example.
[50]From Luhn (1958b), p. 119.

So, terms that rarely appear in a document collection will have little discriminative power and can be eliminated.[51] In contrast, high frequency terms are assumed to be common and thus not to have discriminative power either. Removing these terms in fact corresponds to eliminating stop words where the stop list is automatically constructed from the given document collection based on the term frequency histogram. In the following, we give a justification for thresholding low frequency terms.

Based on empirical studies, Zipf observed over 50 years ago that many words in a document collection appear very infrequently.[52] Zipf's observation about the frequency of word occurrences in a document collection is referred to as *Zipf's Law*, although it is rather an approximate mathematical phenomenon.[53] In spite of the fact that Zipf's analysis and Luhn's observation are originally based on words, they can also be applied to other index terms, such as word stems, phrases, or n-grams.[54] Thus, we formally describe Zipf's Law referring to terms rather than words. Assume all index terms in the preliminary vocabulary, $t \in \hat{\mathcal{V}}$, are sorted in decreasing order according to their total number of occurrences in the document collection, $\mathit{tf}(t)$. Let $\mathrm{rank}(t)$ denote the position of term t in the sorted list. Zipf's Law states that

$$\mathrm{rank}(t) \cdot \mathit{tf}(t) \approx \mathit{const} \tag{3.7}$$

This relationship is demonstrated by the hyperbolic curve denoted *term frequency* in Figure 3.2. On the basis of the analysis of various document collections, Callan (1997) reports that the constant tends to be about $\frac{\mathit{tf}}{10}$ in English text, where tf is the total number of terms in a document collection as introduced above. Following Sahami's analysis of Zipf's Law, we try to estimate the fraction of distinct terms t in a document collection whose total frequency equals a given value, say $\mathit{tf}(t) = \beta$.[55] Let term t_i be the lowest ranked term, with $\mathit{tf}(t_i) = \beta$, and t_j be the lowest ranked term, with $\mathit{tf}(t_j) = \beta + 1$. The rank of the terms can be expressed approximately as $\mathrm{rank}(t_i) \approx \frac{\mathit{const}}{\beta}$ and $\mathrm{rank}(t_j) \approx \frac{\mathit{const}}{\beta+1}$, respectively. Subtracting the latter expression from the first yields

$$\mathrm{rank}(t_i) - \mathrm{rank}(t_j) \approx \frac{\mathit{const}}{\beta} - \frac{\mathit{const}}{\beta+1} = \frac{\mathit{const}}{\beta\,(\beta+1)} \tag{3.8}$$

Further, note that the term with the highest rank, $t_{\max}$, generally appears once in a document collection, and thus

$$\mathrm{rank}(t_{\max}) \approx \frac{\mathit{const}}{1} = \mathit{const} \tag{3.9}$$

Dividing Equation (3.8) by Equation (3.9), we obtain $\frac{1}{\beta\cdot(\beta+1)}$ as an approximation to the fraction of terms that appear β times in a document collection. Hence, Zipf's Law shows that a large fraction of terms is accounted for by those terms that occur most infrequently. For instance, about half of all distinct terms appear only once. Sahami concludes that

[51] See van Rijsbergen (1979), p. 16.

[52] See Zipf (1949), pp. 22–27.

[53] See Sahami (1998), p. 27.

[54] See van Rijsbergen (1979), p. 16.

[55] See Sahami (1998), pp. 27–29.

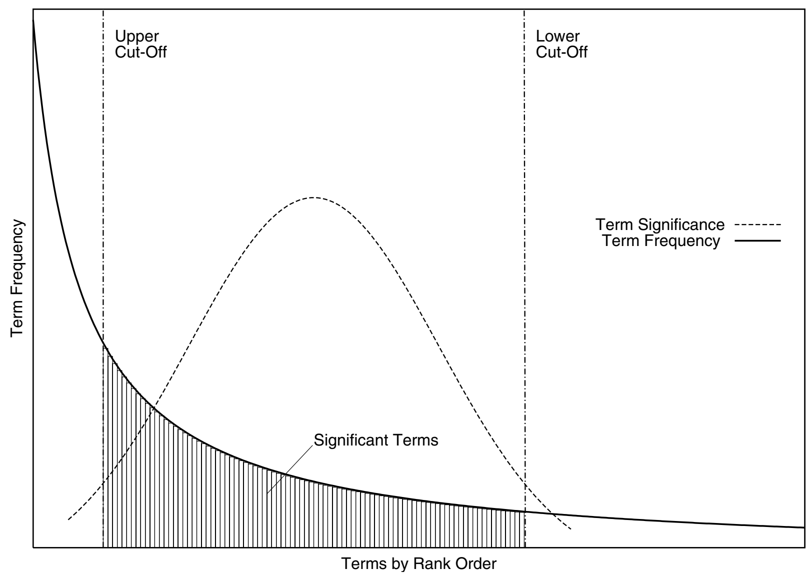


Figure 3.2: Luhn's application of Zipf's Law (following Luhn (1958b), p. 120).

even if Zipf's *Law* is "only a loose approximation, it still provides compelling evidence that we can eliminate a significant number of terms [...] by simply eliminating the terms with the lowest frequency of occurrence" in the document collection.[56]

Figure 3.2 illustrates Luhn's application of Zipf's Law. The frequency of the terms in order of their rank is indicated by the hyperbolic curve in Figure 3.2. The vertical lines demonstrate possible cut-offs for high and low frequency terms. While both high and low frequency terms lack discriminative power, the remaining words of medium frequency are considered significant for discriminating among classes. It is assumed that term significance peaks somewhere between the two cut-off points as sketched by the bell-shaped curve in Figure 3.2.[57] By thresholding non-significant high and low frequency terms, the dimensionality of the feature space can be reduced by up to 50 per cent.[58] Note that finding appropriate thresholds remains on open issue and should be tackled empirically based on the document collection and the application at hand.

Document Frequency Thresholding. The number of documents in a document collection in which a particular term t appears at least once, $n(t)$, is also somewhat misleadingly referred to as the document frequency of term t. A very simple selection heuristic is to exclude all terms from the vocabulary whose document frequency is less than some predetermined threshold.[59] This is based on the assumption that terms that occur in only very few documents are unlikely to carry general class-specific information and sometimes even tend to be noise, e.g. spelling mistakes. Furthermore, using infrequently occur-

[56]From Sahami (1998), p. 28.
[57]See Salton and McGill (1983), p. 62.
[58]See van Rijsbergen (1979), p. 17.
[59]See Yang and Pedersen (1997).

ring terms is not statistically reliable.[60] Hence, removing these terms not only maintains discriminative power, it can also improve classification effectiveness. Yet, note that, in information retrieval, a term that occurs in only a single document may be especially useful for identifying this document. Typically, document frequency thresholding is applied prior to further, more elaborate feature selection, removing terms that occur, for instance, only once or twice in the document collection.[61]

Inverse Document Frequency and TFIDF. As stated above, rarely occurring terms are unlikely to be relevant for characterizing a certain class. In contrast, terms that occur in a large portion of the document collection might not be discriminative either. That is, term importance is assumed to be inversely proportional to the number of documents a particular term appears in. A possible measure for this is the inverse document frequency for term t, which is typically defined as[62]

$$\mathrm{idf}(t) \quad = \quad \log \frac{n}{n(t)} \tag{3.10}$$

such that terms with higher values are preferred. Having removed common stop words, we here assume that the importance of a term increases with its use-frequency. Combing these ideas leads to the *term frequency/inverse document frequency* (tfidf) measure:

$$\mathrm{tfidf}(t) \quad = \quad \mathit{tf}(t) \cdot \mathrm{idf}(t) \tag{3.11}$$

which assigns higher values to terms that are considered more important. Note that the evaluation of this measure does not depend on the true class labels of the training documents. Also, a similar combination of term frequency and inverse document frequency is often used to assign weights to terms in the vector generation step, which will be covered in Subsection 3.2.4. For term weighting, however, the term frequency of a term in a single document rather than in the entire document collection is considered.[63]

Signal-to-Noise Ratio. In communication theory, the signal-to-noise ratio measures the strength of a signal relative to background noise. Abstracting from its technical meaning, the signal-to-noise ratio of a particular term measures the discriminative power conveyed by that term.[64] For the purpose of dimensionality reduction, terms with larger values are preferred. The evaluation of the background noise is based on the information-theoretic measure of entropy, which we will introduce next.

Information can be considered as reduction in uncertainty. The amount of information conveyed by a message depends on its probability of occurrence and can be measured as minus the logarithm of that probability. Typically, a logarithm of base 2 which measures

[60]See Apté *et al.* (1994), p. 237.

[61]For example, see Joachims (1997b), p. 2.

[62]See Salton and Buckley (1988), p. 516, or Salton (1989), p. 280, for example. Another common definition is $\mathrm{idf}(t){=}1 + \log \frac{n}{n(t)}$, which was originally proposed by Sparck Jones (1972), pp. 17–18.

[63]See Salton and Buckley (1988), p. 516.

[64]See Salton and McGill (1983), pp. 63–66.

the amount of information in bits is assumed. Obviously, receiving a rarely occurring message is more informative than receiving a frequently occurring message. Considering each presentation of a discrete random variable as a message, we can generally define entropy as a measure of average uncertainty in a discrete probability distribution.[65]

Definition 3.2.2 (Entropy)
The entropy of a discrete random variable X that can take on c different values with probabilities p_i, $i = 1, \ldots, c$, is defined by

$$\text{Entropy}(X) \;=\; -\sum_{i=1}^{c} p_i \log p_i \tag{3.12}$$

So, entropy can be evaluated as the amount of information that we can expect to receive on average when observing the particular random variable. Notice that $0 \log 0$ is defined to be 0 for the evaluation of entropy.[66] The range of the entropy function is $[0, \log c]$. Entropy is zero when one outcome of the random variable occurs with certainty, i.e. $p_i = 1$ for an arbitrary i and $p_j = 0$ for $j \neq i$. In this case, we do not receive any information when knowing about the outcome of the random variable X. The more uniform a distribution, the larger is its entropy. Hence, entropy takes on its maximum value $\log c$ if all outcomes of X are equally likely.

A term should be concentrated in only a few documents. A measure for this, here referred to as 'noise', can be evaluated as the entropy of the probability distribution of term t among the documents:

$$\text{Noise}(t) \;=\; -\sum_{j=1}^{n} \mathrm{P}(d_j, t) \log \mathrm{P}(d_j, t) \tag{3.13}$$

where the probability that a particular document d_j and term t co-occur is estimated by $\mathrm{P}(d_j, t) = \frac{tf_{d_j}(t)}{tf(t)}$. The range of the noise function is $[0, \log n]$. According to the definition of entropy, noise is zero when term t appears in one document only, whereas it takes on its maximum value $\log n$ if term t occurs with the same frequency in all documents.

Furthermore, it is assumed that the more frequently a term t occurs, the more discriminative it is. This corresponds to the 'signal' and is measured as $\log tf(t)$. The signal-to-noise ratio is now expressed as the difference of these logarithms, yielding[67]

$$\text{SNR}(t) \;=\; \log tf(t) - \text{Noise}(t) \tag{3.14}$$

The range of the signal-to-noise ratio is $[0, \log tf(t)]$. The ratio is zero when term t appears exactly once in each document, whereas it takes on the maximum value $\log tf(t)$ if term t occurs in one document only.

Note that the signal-to-noise ratio is class-independent since the evaluation of neither the signal nor the noise term depend on the class labels of the training documents. Similar

[65] See Cover and Thomas (1991), pp. 12–15.
[66] See Cover and Thomas (1991), p. 13.
[67] See Salton and McGill (1983), p. 65.

to the tfidf measure, the signal-to-noise ratio prefers terms that frequently occur in only some of the training documents. And if these documents do, in fact, belong to the same class, the signal-to-noise ratio can be helpful in identifying promising features.

Information Gain. The information-gain criterion is frequently used as a feature selection technique in machine learning, particularly for constructing decision trees.[68] It measures the statistical dependence between a term and the class labels and is also based on entropy. For dimensionality reduction, terms with small information gain are discarded.[69] Note that information gain is also known as *average mutual information* between a term and the class labels.[70]

Rather than evaluating the entropy of a term distribution among a set of documents as is done for the signal-to-noise ratio, we now consider the entropy of the class distribution. Let C denote a random variable used for observing the k possible class labels for the training documents. Here, entropy measures the homogeneity of the training set $D^\star$ with respect to the class distribution which governs C:[71]

$$\mathrm{Entropy}(C) \quad = \quad -\sum_{i=1}^{k} \mathrm{P}(c_i) \log \mathrm{P}(c_i) \tag{3.15}$$

where $\mathrm{P}(c_i) = \frac{n_{c_i}}{n}$ denotes the probability of observing a training document with class label c_i. Note again that $0 \log 0$ is defined to be 0 for all entropy evaluations.

Based on this interpretation of entropy, we can measure the discriminative power of a particular index term as follows. Let t and $\bar{t}$ denote the presence and absence of term t, respectively, and T be a binary random variable taking on the values t and $\bar{t}$. The conditional entropy of the random class variable C given T is defined as[72]

$$\begin{aligned}\mathrm{Entropy}(C|T) &= \mathrm{P}(t)\ \mathrm{Entropy}(C|t) + \mathrm{P}(\bar{t})\ \mathrm{Entropy}(C|\bar{t}) && (3.16)\\ &= -\,\mathrm{P}(t) \sum_{i=1}^{k} \mathrm{P}(c_i|t) \log \mathrm{P}(c_i|t) - \mathrm{P}(\bar{t}) \sum_{i=1}^{k} \mathrm{P}(c_i|\bar{t}) \log \mathrm{P}(c_i|\bar{t}) && (3.17)\end{aligned}$$

where $\mathrm{P}(t) = \frac{n(t)}{n}$ and $\mathrm{P}(\bar{t}) = \frac{\bar{n}(t)}{n}$ denote the proportions of training documents in which term t is present and absent, respectively. The conditional probabilities are estimated by $\mathrm{P}(c_i|t) = \frac{n_{c_i}(t)}{n(t)}$ and $\mathrm{P}(c_i|\bar{t}) = \frac{\bar{n}_{c_i}(t)}{\bar{n}(t)}$. Finally, we define the information gain of term t as the expected reduction in entropy caused by partitioning the set of training examples D according to the presence or absence of term t, yielding[73]

$$\mathrm{Gain}(t) \quad = \quad \mathrm{Entropy}(C) - \mathrm{Entropy}(C|T) \tag{3.18}$$

[68]For example, see Quinlan (1993), pp. 20–22, or Mitchell (1997), pp. 57–58.
[69]See Yang and Pedersen (1997).
[70]See McCallum (1996).
[71]See Mitchell (1997), pp. 55–57.
[72]See Cover and Thomas (1991), p. 16.
[73]See Mitchell (1997), pp. 55–60.

which can be transformed into

$$\mathrm{Gain}(t) \;=\; \sum_{i=1}^{k} \mathrm{P}(c_i,t)\log\frac{\mathrm{P}(c_i,t)}{\mathrm{P}(c_i)\mathrm{P}(t)} + \sum_{i=1}^{k} \mathrm{P}(c_i,\bar{t})\log\frac{\mathrm{P}(c_i,\bar{t})}{\mathrm{P}(c_i)\mathrm{P}(\bar{t})} \qquad (3.19)$$

by using elementary probability calculus. The probabilities that a class c_i and a term t do or do not co-occur can be derived from the probabilities introduced above as $\mathrm{P}(c_i,t) = \frac{n_{c_i}(t)}{n}$ and $\mathrm{P}(c_i,\bar{t}) = \frac{\bar{n}_{c_i}(t)}{n}$, respectively.

Mutual Information. Note that information gain is an averaged performance score that measures the mean discriminative power of a term over all classes. Hence, it cannot take into account that a particular term may predict only one of the classes well while performing less well on the other classes.[74] Albeit important to describe a certain class, such a term may not be selected using the information-gain criterion. Obviously, this problem can only arise if there are more than two classes. Note that classification problems with $k > 2$ can always be decomposed into k binary problems, as can the task of evaluating class-dependent measures. Thus, we proceed by evaluating the mutual information between a term t and each class $c \in \mathcal{C}$ individually instead of averaging over all classes. Let $\hat{c}_1 \equiv c$ be a particular class and $\hat{c}_2 \equiv \bar{c}$ be its complement, i.e. $\hat{c}_2$ labels all documents that are not in class c. Applying Equation (3.19) to these artificially introduced classes yields the mutual information between c and t:[75]

$$\mathrm{MI}(c,t) \;=\; \sum_{i=1}^{2} \mathrm{P}(\hat{c}_i,t)\log\frac{\mathrm{P}(\hat{c}_i,t)}{\mathrm{P}(\hat{c}_i)\mathrm{P}(t)} + \sum_{i=1}^{2} \mathrm{P}(\hat{c}_i,\bar{t})\log\frac{\mathrm{P}(\hat{c}_i,\bar{t})}{\mathrm{P}(\hat{c}_i)\mathrm{P}(\bar{t})} \qquad (3.20)$$

We thus obtain scores for each combination of terms and class labels. Based on these, we can select k, possibly overlapping feature subsets, which are subsequently merged to establish the final vocabulary.

Chi-Square Statistic. The χ^2 statistic measures the degree of association between a term and the class labels. Its application is based on the assumption that a term whose frequency strongly depends on the class label of the document in which it occurs will be useful for discriminating among the classes. For the purpose of dimensionality reduction, terms with small χ^2 values are discarded.

Assume again the two-class classification problem where $c_1 \equiv non$ and $c_2 \equiv rel$ denote non-relevant and relevant documents, respectively. Note that in case there are $k > 2$ classes, we can split the feature selection problem into k subtasks as described above. The numbers that reflect the co-occurrence of a term t and the class labels, that is $n_{c_i}(t)$ and $\bar{n}_{c_i}(t)$ for both class labels non and rel, can be regarded as the entries in the two-way contingency table. Based on this, the χ^2 statistic for term t is defined as[76]

$$\chi^2(t) \;=\; \frac{n\,[n_{rel}(t)\,\bar{n}_{non}(t) - \bar{n}_{rel}(t)\,n_{non}(t)]^2}{n_{rel}\,n_{non}\,n(t)\,\bar{n}(t)} \qquad (3.21)$$

[74]See Schürmann (1996), p. 258.

[75]See Dumais *et al.* (1998).

[76]See Schütze *et al.* (1995), for example.

for terms that do not occur in all documents, i.e. if $n(t) < n$. The χ^2 statistic is normalized in the range $[0, n]$. The value is zero if term t occurs independent of the class labels, and it becomes larger the more frequently it occurs in only one of the classes. The extremeness of the resulting value can be judged by comparison to the χ^2 distribution with one degree of freedom.[77] If a term occurs in all documents, i.e. if $n(t) = n$, the χ^2 statistic can be defined as 0 since term t occurs independent of the class labels. For low-frequency terms, a comparison between χ^2 values may not be accurate. And, if these terms are essential for discriminating among classes, the χ^2 statistic may not be appropriate.[78]

Reparameterization

Reparameterization techniques for dimensionality reduction take as input a set of features derived from the training documents and construct a new set of features that contains fewer features by taking combinations and transformations of the existing features, while maintaining or even enhancing discriminative power. Note that this mapping is applied not only to training documents but also to any new document, if reparameterization is chosen as a means of dimensionality reduction. Also, note that the resulting features may be artificial and may thus not correspond to the index terms that can be found in the documents. We start by examining two linguistic transformation techniques: *stemming* and *using a thesaurus*. Then, we briefly turn to *latent semantic analysis*.

Stemming. As discussed in Section 3.2.2, using plain words as index terms often leads to an unnecessarily large number of features since morphological variants of a word are not recognized as being similar and each inflection of a word is a potential feature. “Word stemming is a crude pseudo-linguistic process which removes suffices to reduce words to their word stem.”[79] For example, the words ‘classifier’, ‘classified’ and ‘classifying’ would all be reduced to the word stem ‘classify’. Consequently, the dimensionality of the feature space can be reduced by mapping morphologically similar words onto their word stem. A widely applied stemming algorithm is the suffix stripper developed by Porter (1980). Note that stemming algorithms are commonly based on heuristics and may thus conflate words that are actually not similar. For example, the Porter stemmer maps the word ‘is’ to ‘i’. Frakes (1992) describes various stemming algorithms and gives an overview of several studies comparing different stemming methods for the information retrieval task. Based on this comparison, Sahami states that both stemmed and unstemmed representations lead to roughly equal performance.[80] Obviously, stemming is language- but not domain-dependent. It is rather straightforward to specify simple heuristics for stemming English words, whereas providing stemming rules for other languages, such as German or French, can be much more difficult. Finally, note that in case of n-gram-based index terms, stemming should, if desired, take place before the term extraction step based on the word tokens output by text normalization step.

[77]See Yang and Pedersen (1997).
[78]See Dunning (1993), pp. 62–63.
[79]From Smeaton (1997), p. 122.
[80]See Sahami (1998), p. 20.

Using A Thesaurus. While stemming aims at conflating morphological variants of words, the objective of using a thesaurus in the context of dimensionality reduction is to conflate synonyms, i.e. words of the same or similar meaning. Generally, a thesaurus is a collection of words that are grouped by likenesses in their meaning rather than in alphabetical order. For example, a thesaurus may contain semantic relations between words such as 'is similar to', 'is more specific than', or 'is more general than' to describe synonyms, hyponyms, or hypernyms, respectively. These relations can be used to group words in semantic equivalence classes. The dimension of the vector space can thus be reduced by mapping all the index terms onto the equivalence class to which they belong. Note that another common way of using a thesaurus is to expand rather than conflate the set of index terms. This idea is similar to a technique known as *automatic query expansion* in information retrieval and refers to adding semantically related words to a set of keywords which form a query.[81] Obviously, a thesaurus is domain- and language-dependent, and often manually constructed thesauri are used. Yet, there are also attempts to automatically construct thesauri based on document collections.[82] In fact, we consider latent semantic analysis as a statistical technique for automatically uncovering semantic structure in a document collection, which can be used to reduce the set of features instead of a manually constructed thesaurus.[83]

Latent Semantic Analysis. Latent semantic analysis builds on statistical techniques for estimating semantic associations among terms across the document collection.[84] Exploiting these associations can solve the synonym problem (see above) and reduce the dimensionality of the feature space. The primary assumption of latent semantic analysis is that "there is some underlying or latent structure in word usage that is partially obscured by variability in word choice."[85] In information retrieval, latent semantic analysis is originally based on *singular value decomposition*.[86] More common in the pattern recognition community is the application of *principal component analysis* to find directions in feature space which maximally explain variance.[87] An alternative method for uncovering semantic structure in documents automatically is *term clustering*. Term clustering attempts to find groups of terms with related meaning through cluster analysis.[88]

Method Combination

So far, we have discussed several techniques which enable control of the dimensionality of the vocabulary whose elements correspond to the dimensions of the vector space in which we represent documents. Rather than applying just one of these techniques, in many applications, some of them are used in combination. For instance, it is quite

[81]For example, see Xu and Croft (1996).
[82]For example, see Salton and McGill (1983), pp. 77–81, or Salton (1989), pp. 299–303.
[83]See Dumais (1994).
[84]See Deerwester *et al.* (1990), for example.
[85]From Berry *et al.* (1995).
[86]See Deerwester *et al.* (1990).
[87]See Schürmann (1996), pp. 270–275, for example.
[88]See Lewis and Croft (1990). Also see Sparck Jones (1971) for early work on term clustering.

common to, first, eliminate stop words based on a manually provided stop list. Then, the remaining words are often reduced to their words stems. After that, rarely occurring terms are removed, for example, by either thresholding their term frequency or the corresponding document frequency. Finally, a more elaborate method such as the information-gain criterion is applied to further reduce the number of features.[89]

3.2.4 Vector Generation

Given the term frequency statistics of all training documents, the task of the *vector generation* step is to create a weighted vector $\mathbf{d} = (w(d,t_1),\ldots,w(d,t_m))^T$ for any document d based on its term frequency vector $\mathbf{d}_{tf} = (tf_d(t_1),\ldots,tf_d(t_m))^T$, which commonly results from the term extraction step. Each weight $w(d,t)$ expresses the importance of term t in document d with respect to its frequency in all training documents. The objective of using a term weight rather than plain frequencies is to enhance classification effectiveness.[90]

As mentioned above, term importance is commonly determined as a function of how often a particular term appears in a certain document and how often it appears in the entire document collection. By varying this function, we can produce different term weightings, which can then be interpreted as different indexing models, as we will see in the following. Salton and Buckley (1988) identify three main weighting components in determining term importance, which we refer to as the local, global, and normalization factors.[91] Based on these three weighting factors, the term weight is evaluated as

$$w(d,t) = \frac{w_{\text{local}}(d,t)\, w_{\text{global}}(t)}{w_{\text{norm}}(d)} \tag{3.22}$$

Some common choices for the three weighting components are listed in Table 3.3. The three factors are motivated as follows:

- *Local component.* The local weighting factor $w_{\text{local}}(d,t)$ reflects the importance of term t within a particular document d. Recall from Section 90 that the term frequency is considered as a reasonable indicator of term importance. Information retrieval research suggests that broad, high-frequency terms be emphasized in order to enhance effectiveness in terms of recall.[92]

  Typically, the local weighting factor is obtained by applying a transformation function f to the term frequency in case term t should appear in document d:

$$w_{\text{local}}(d,t) = \begin{cases} f(tf_d(t)) & \text{if } tf_d(t) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.23}$$

  Note that $w_{\text{local}}(d,t)$ is defined to be 0 and, thus, $w(d,t) = 0$ independent of the global weighting factor if term t does not appear in document d.

[89] For example, see Joachims (1997b), pp. 1–2, Dumais *et al.* (1998), or Nigam *et al.* (2000).
[90] See Salton and Buckley (1988), p. 516.
[91] For example, see Dumais (1991) for the terminology of local and global term weighting.
[92] See Salton and Buckley (1988), p. 516.

Two straightforward transformation functions are, for instance, the identity function, which does not change the given term frequencies, and the binary indicator function, which returns one for a non-zero term frequency and zero otherwise. More elaborate functions could normalize the term frequencies through division by the largest term frequency $tf_d(t_{\max})$ within document d. Another common choice is to apply a logarithm to the term frequency such that an additional occurrence of term t in document d is considered more important at smaller term frequency levels than at larger levels.

- *Global component.* The global weighting factor $w_{\text{global}}(t)$ takes into account the importance of term t within the entire set of training documents. As discussed in Section 90, a term is considered more important if it is concentrated in only a few documents. A measure for this is often referred to as the *inverse document frequency*.[93] Giving more weight to narrow, highly specific terms allows us to better reject non-relevant documents and can enhance precision.[94]

  For example, a widely applied global weighting factor is the inverse document frequency defined in Section 90 as $\log \frac{n}{n(t)}$. Recall that n denotes the total number of training documents, whereas $n(t)$ denotes the number of documents in which term t appears. In a probabilistic context where no relevance feedback is available, the inverse document frequency could be defined as $\log \frac{\bar{n}(t)}{n(t)}$, where $\bar{n}(t) = n - n(t)$ denotes the number documents in which term t does not appear.[95]

- *Normalization component.* The denominator in Equation (3.22) $w_{\text{norm}}(d)$ is a normalization factor. Normalizing the weight vector of a document d permits abstraction from varying document lengths. Typically, the weighted vectors are normalized to unit length, which is also referred to as cosine normalization. In a probabilistic context, it is also common to normalize by the sum of the weighted vector components of a document.[96]

  Note that longer documents often tend to contain more distinct index terms. And with a larger number of index terms present in a document, the chance of finding terms that match those in other documents is higher. Consequently, longer documents have a higher chance of being similar to other documents if they are not normalized.[97] Also, many similarity calculations such as the cosine similarity normalize the weight vectors, as we will see in Section 3.3. Hence, it may be computationally more efficient to normalize the weight vectors once prior to these calculations rather than each time a document is involved in a similarity calculation.

The codes for the weighting components given in Table 3.3 correspond to the notation used in the information retrieval system Smart.[98] A particular term weighting scheme can

[93]See Harman (1992), pp. 373–376.
[94]See Salton and Buckley (1988), p. 516.
[95]See Croft and Harper (1979).
[96]See Salton and Buckley (1988), p. 517.
[97]See Salton and Buckley (1988), p. 517.
[98]The Smart system is publicly available at `ftp://ftp.cs.cornell.edu/pub/smart`.

Code	Description	Expression
Local component for term t in document d:		$w_{\text{local}}(d,t) =$ (if $tf_d(t) > 0$)
n	no conversion (plain term frequency)	$tf_d(t)$
b	binary term indicator	1
m	normalize by most frequent term $t_{\max}$	$\frac{tf_d(t)}{tf_d(t_{\max})}$
a	augmented normalized term frequency	$\frac{1}{2} + \frac{1}{2}\frac{tf_d(t)}{tf_d(t_{\max})}$
l	logarithm of term frequency	$1 + \log tf_d(t)$
Global component for term t:		$w_{\text{global}}(t) =$
n	no global weighting	1
t	tfidf style inverse document frequency	$\log \frac{n}{n(t)}$
p	probabilistic inverse document frequency	$\log \frac{\bar{n}(t)}{n(t)}$
Normalization component for document d:		$w_{\text{norm}}(d) =$
n	no normalization	1
s	normalize by sum of all term weights	$\sum_{i=1}^{m} w_{\text{local}}(d,t_i)\; w_{\text{global}}(d,t_i)$
c	cosine normalization	$\sqrt{\sum_{i=1}^{m} (w_{\text{local}}(d,t_i)\; w_{\text{global}}(d,t_i))^2}$

Table 3.3: An overview of common term weighting components.

be characterized by specifying a three-letter code in which the first letter corresponds to the local factor, the second letter to the global factor, and the third letter to the normalization component.

For example, using the 'nnn' weighting scheme leaves the term frequency vectors unchanged, $w_{tf}(d,t) = tf_d(t)$, whereas the weighting scheme 'ntn' produces the well-known tfidf weights:[99]

$$w_{\text{tfidf}}(d,t) \quad = \quad tf_d(t)\;\log \frac{n}{n(t)} \tag{3.24}$$

For 'bnn' we obtain a simple binary representation of documents, in which each component of the document vector denotes whether or not the corresponding term appears in the document. This is equivalent to

$$w_{\text{bin}}(d,t) \quad = \quad \begin{cases} 1 & \text{if } tf_d(t) \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.25}$$

Note that a binary indicator of term presence or absence in a document is widely applied in many learning algorithms such as decision tree induction or the naïve Bayes classifier, as we will see in the following section. Combined with an appropriate similarity measure, this representation can also be used to mimic Boolean indexing, which many information retrieval systems operate on.[100] Other well-known retrieval models are based on

[99]See Salton and McGill (1983), p. 63, for example.
[100]See Salton and Buckley (1988), p. 517.

probabilistic indexing.[101] For instance, the weights used under the binary independence assumptions for terms can be obtained by the 'bpn' weighting scheme:[102]

$$w_{\text{prob}} = \begin{cases} \log \frac{\bar{n}(t)}{n(t)} & \text{if } tf_d(t) \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.26}$$

Retrieval as for the vector space model is originally based on the cosine similarity between documents with simple tfidf style weights, which is equivalent to using the 'ntc' weighting scheme.[103] Derivatives like the 'mtc', 'atc', and 'ltc' weighting schemes were proposed to enhance retrieval effectiveness. In the similarity-based learning algorithms to be discussed in the following section for the task of text classification, we will use the 'ltc' weighting scheme, which is also preferred by Yang (1999) in a comparative study of different text learning algorithms. In this case, the weight of term t in document d is defined as

$$w_{\text{ltc}}(d,t) = \frac{f(tf_d(t)) \cdot \log \frac{n}{n(t)}}{\sqrt{\sum_{i=1}^{m} f(tf_d(t_i)) \cdot \log \frac{n}{n(t_i)}}} \tag{3.27}$$

with $f(\xi) = 1 + \log \xi$ if $\xi > 0$ and otherwise $f(\xi) = 0$, as discussed previously.

3.2.5 Summary

With the techniques described above, we are now able to map any document $d \in \mathcal{D}$ onto its vector representation $\mathbf{d} = \rho(d) = (w_1, \ldots, w_m)^T \in \mathcal{R} = \mathbb{R}_+^m$ with respect to the term frequency statistics of a given document collection. We, thus, carry out the mapping $\rho : \mathcal{D} \mapsto \mathcal{R}$ from the space of actual documents onto the document representation space. The resulting document vector $\mathbf{d}$ is suitable as input for the learning algorithms to be covered in the following section. For the sake of simplicity, we will use the terms document and document vector synonymously and refer to the actual documents by means of their vector representation. Specifically, we refer to the set of training documents through their representations as $D = \{\mathbf{d} \in \mathbb{R}_+^m \mid \exists d \in D^\star : \mathbf{d} = \rho(d)\}$.

Consequently, we assume that a document d can be uniquely identified by its document vector $\mathbf{d} = \rho(d)$. Although chances seem low in real-world text classification problems, two distinct documents may, in fact, be mapped onto the same document vector. Assume, for example, that two electronic mails are sent to two different newsgroups and are thus considered to belong to two different classes. Furthermore, let both mails express a user's intention to be subscribed to a particular mailing list in such a way that each mail contains solely the subject information 'subscribe' and no additional text body. Both mails are then likely to be mapped onto the same document vector. Nonetheless, we assume throughout this dissertation that we can uniquely identify a document by its document vector.

[101] See Harman (1992), pp. 367–370, for example.
[102] See Croft and Harper (1979), p. 287.
[103] See Harman (1992), pp. 366–367.

3.3 Text Learning

3.3.1 The Supervised Learning Task Revisited

In Section 3.1, we have formally defined the supervised text learning task as follows. Assume a fixed set of k classes $\mathcal{C} = \{c_1, \ldots, c_k\}$ and a set of n training documents from the document space $D^\star = \{d_1, \ldots, d_n\} \subset \mathcal{D}$, for which the true class labels are given by the target function $T(d)$ for each $d \in D^\star$. The text learning task is to induce from $D^\star$ a classifier, the hypothesis $h : \mathcal{D} \mapsto \mathcal{C}$, that approximates the target function T well with respect to a given effectiveness measure. In accordance with the *inductive learning hypothesis*, this classifier can then be used to predict the class labels of new, previously unseen documents.

With respect to the document representation introduced in the previous section, the text learning task is now formulated as inducing from the set of training document vectors $D = \{\mathbf{d}_1, \ldots, \mathbf{d}_2\} \subset \mathcal{R}$ a classifier, the hypothesis $H : \mathcal{R} \mapsto \mathcal{C}$, that approximates the target function $T : \mathcal{R} \mapsto \mathcal{C}$ well with respect to a given effectiveness measure. Note that we again denote the target function by T. Yet, this time the domain is the document representation space $\mathcal{R}$ instead of the document space $\mathcal{D}$ proper. Nevertheless, there should be no risk of confusing these notations since the intended meaning should become clear from their context.

Note that the hypotheses H and h are connected by the document representation function ρ, since $h(d) = H(\rho(d)) = H(\mathbf{d})$ for $d \in \mathcal{D}$. As mapping a document d onto its surrogate representation $\mathbf{d}$ is fixed prior to learning a classifier, we will stick to the representation space of documents in the remainder of this chapter. That is, we refer to documents in terms of the document vector as if these representations were the actual documents.

For the sake of notational simplicity, in some formulae we will also denote the class label of training document $\mathbf{d}$ by $y \equiv T(\mathbf{d}) \in \mathcal{C}$. Furthermore, for some learning algorithms introduced in this section, it will be convenient to introduce a notation for grouping all training documents into subsets according to their class labels. In Definition 3.1.1 we noted that the class labels given to documents imply an exhaustive and disjoint partition of the document space. We now denote the set of those training documents belonging to class $c_i \in \mathcal{C}$ as $D_{c_i} = \{\mathbf{d} \in D \,|\, T(\mathbf{d}) = c_i\}$, such that $D = \bigcup_{c_i \in \mathcal{C}} D_{c_i}$. Also, recall that $n_{c_i} = |D_{c_i}|$ is the number of training documents belonging to class $c_i \in \mathcal{C}$ as previously defined in Section 3.2. Specifically, for information filtering, D_{rel} and D_{non} denote the subsets of the n_{rel} relevant and the n_{non} non-relevant training documents, respectively.

The Document Source

So far, we have talked about training documents and new documents that are to be classified. In a theoretical framework, it is often assumed that all these documents come from some hypothetical source. The application of this framework is two-fold. On the one hand, it can be used explicitly to derive probabilistic classifiers as we will do below. On the other hand, the concept of a general document source can serve as a means to describe different properties and assumptions of learning algorithms as is commonly done in computational learning theory.[104] As for now, we focus on the latter point.

The *document source* is a stochastic model which describes the general setting of the learning task.[105] In particular, the document source is assumed to be capable of generating unlimited numbers of documents of different classes. For the document generation process, we can assume that, first, a class-specific distribution is selected, and a document is then generated according to this distribution. So, each document generated is associated with a particular but unknown class label. The document source can more precisely be considered as a mixture model which comprises a finite number of class-specific document sources, each associated with a specific distribution for generating documents.

The training set D is a randomly drawn sample from the document source, for which the true class labels $T(\mathbf{d}) \in \mathcal{C}$ for all $\mathbf{d} \in D$ are known. A classifier is learned from the training documents and subsequently applied to predict the class labels of new documents, which are also assumed to be generated by this same document source. And this is exactly the fundamental *stationarity assumption*, which entitles us to employ the inductive learning hypothesis as described above. For all learning algorithms introduced subsequently, it is generally assumed that the distributions which fully determine the document source be *stationary*. In other words, the class-specific document sources do not change over time. We discuss the issue of *dynamic*, i.e. non-stationary, document sources in Chapter 6.

In addition to this basic stationarity assumption, there are usually further assumptions associated with a learning algorithm. For example, some learners require that there be exactly one mixture component for each class, enabling a one-to-one correspondence between mixture components and classes. Assumptions of this kind are part of the general idea of *inductive bias* as we will discuss next.

Inductive Bias

The objective of the learning task is to infer a hypothesis from training data which can subsequently be used to classify previously unobserved examples. Classifying new examples requires generalizing beyond the provided training data. How can this inductive leap be logically justified? The idea of the *inductive bias* captures “the policy by which the learner generalizes beyond the observed training data, to infer the classification of new instances.”[106] Without inductive bias, there is no rational basis for classifying any new examples.

In general, it cannot be proven that the hypothesis inferred by a learner is correct. In other words, the class label predicted for a new example need not follow deductively from the training data and the representation of the new example. Inductive bias can thus be defined more precisely as follows.[107]

[104] See Kearns and Vazirani (1994), for example.

[105] See Schürmann (1996), pp. 20–21, for a more general definition of a pattern source for pattern recognition tasks.

[106] See Mitchell (1997), p. 42.

[107] See Mitchell (1997), p. 43.

Definition 3.3.1 (Inductive Bias)
The inductive bias of a learner can be defined as any minimal set of assertions subject to which the learner's inductive inferences follow deductively.

Originating from work in artificial intelligence, the process of learning from examples is often considered as a *search* through an implicit or explicit hypothesis space for the hypothesis that best fits the training data.[108] To make it efficient, the search through the space of possible hypotheses must be limited or directed in some way.[109] These issues greatly determine the study of learning algorithms.

One key factor that limits the hypothesis space a learner explores is the set of features used to describe examples.[110] As set out in Section 3.2, deciding upon a suitable document representation always involves a trade-off between semantic expressivity and representational complexity. Too many features may unnecessarily increase the set of possible hypotheses, whereas too small a feature set may not provide enough information to separate classes. Hence, the choice of representation has a strong impact on a learner's ability to induce an effective hypothesis.

Besides this restriction due to the representation of examples, limiting and directing the search through the space of possible hypotheses corresponds to two types of inductive bias: *restriction* or *language bias* and *search* or *preference bias*.[111] On the one hand, the restriction bias restricts the space of possible hypotheses by limiting the language by which hypotheses are expressed. For example, the hypothesis space could be defined as the set of conjunctions over Boolean variables. "It is important to note that by selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn."[112] So, if the restriction bias is too strict, the hypothesis space may not contain suitable hypotheses. The preference bias, on the other hand, employs a specific order on the set of possible hypotheses which provides a basis for deciding which hypothesis to choose if there are two or more that fit the training data equally well.

When we introduce different learning algorithms in the following, describing their inductive bias will help to better characterize these algorithms and gain some understanding whether the application of a particular algorithm is appropriate for a given problem setting. Certainly, this will greatly depend on the nature of the document source which is assumed to generate the documents as described above.

Text Properties

Before starting to discuss some commonly applied text learning algorithms, let us look more closely at the properties of text being represented as document vectors. Analyzing

[108]See Mitchell (1982).
[109]See Langley (1996), p. 18.
[110]See Lewis (1992b), p. 13.
[111]See Mitchell (1997), p. 64.
[112]See Mitchell (1997), p. 23.

text properties can help to find out in general what methods are promising for text learning tasks. We characterize data properties in text domains by the following four points:[113]

- *High-dimensional feature space.* As we have seen in Section 3.2, there are commonly some ten thousand index terms which are used to represent text, despite the fact that we chose to use a rather simple representation. It is known from computational learning theory, that usually the number of training examples should be a multiple of the number of features if reasonable results are to be guaranteed.[114] For a large number of features, however, obtaining a reasonable number of training documents easily becomes intractable, even when ignoring the fact that a person usually has to hand-label the training documents for the supervised learning task. Solutions that aim to remedy the latter point will be dealt with in Chapter 4. Nonetheless, the problem of a reasonable training set size remains and will have a great impact on the choice of the learning methodology. In fact, we will see that several of the commonly applied learning algorithms to do not iteratively optimize the parameters of the hypothesis. So, often there is no proper search process but a straightforward computation of parameters based on the training documents.

- *Sparse document vectors.* Even though the number of possible index terms is usually very large, only very few actually occur in a particular document. We are therefore dealing with *sparse* document vectors. The high-dimensional feature space in which documents are represented is therefore mostly empty. This makes the problem of supervised learning in a high-dimensional feature space even more severe.[115]

- *Few irrelevant features.* The dimensionality reduction techniques presented in Subsection 3.2.3 aim at reducing the number of features used to represent documents under the constraint that the remaining set of features should still discriminate the different classes well. Hence, dimensionality reduction tacitly assumes that some of the features are irrelevant for the classification problem. Experiments show, however, that even those features that are discarded as they are assumed less informative according to a given feature selection measure often bear some information based on which classes can be discriminated better than random.[116] We can thus conclude that there are only few irrelevant features in text classification. The class descriptions are therefore said to be *dense*.

- *Linearly separable classes.* Experiments with some commonly used text corpora show that many classes are linearly separable.[117] Two classes are said to be linearly separable if they can be separated by a single hyperplane. In many cases, linear separability may be facilitated by the high-dimensional feature space. Nevertheless, it also greatly depends on the class definitions. In information filtering, the relevance classes are generally rather broad and thus may not be linearly separable. As this

[113]See Joachims (1997b), pp. 3–4.
[114]See Lewis (1992b), p. 14.
[115]See Jimenez (1998) for discussion of supervised classification in high-dimensional space.
[116]See Joachims (1997b), p. 3.
[117]See Joachims (1997b), pp. 3–4, and Tong and Koller (2000), p. 1006.

issue is related to the homogeneity of classes, it will be discussed in more detail in Chapter 5.

The fact that classes are often linearly separable and generally only few training examples relative to the high-dimensional feature space are available suggests that rather simple learning algorithms be used. And as we will see below, some of the most widely used text learners are indeed simple but still provide surprisingly effective classifiers. Two approaches, namely the naïve Bayes classifier and the instance-averaging prototype-based classifiers, can be characterized as *additive*, which makes them especially well suited for learning dense classes from sparse input vectors.[118] Another learning paradigm that appears particularly appropriate for this kind of data is the support vector machine.[119]

3.3.2 Learning Algorithms

A variety of text learning algorithms have been studied and compared in the literature.[120] In the following, we describe some widely applied supervised learning algorithms which are based on different learning paradigms. We consider three different types of classifiers. First of all, a very prominent kind are similarity or distance classifiers such as the nearest-neighbor rule which we refer to as prototype-based classifiers. Second, also very common to text classification are probabilistic classifiers such as the naïve Bayes classifier. The third type deals with support vector machines. Finally, we give a brief overview of further approaches. Note that the subsequent subsection deals with ensembles of classifiers by discussing frameworks for learning and combining multiple classifiers.

Prototype-based Classifiers

Prototype-based classifiers represent each class in terms of prototypes, thus using the same representational language for the classification model as for the examples.[121] Here, this corresponds to the vector space of distinct index terms as introduced in Section 3.2, typically in combination with a tfidf-style term weighting scheme.

Having generated a set of prototypes based on the training documents, classifying a new document by means of prototypes is a two-step process. First the similarity of the new document to each prototype is computed.[122] These similarity scores are then used to determine the class label of the new document.

[118]See Kivinen and Warmuth (1995), p. 290.
[119]See Joachims (1997b), p. 4.
[120]For example, see Dumais *et al.* (1998), Yang (1999), and Yang and Liu (1999).
[121]Langley (1996), pp. 96–101, describes these methods as instance-based classifiers.
[122]In information retrieval and text classification, similarity measures are applied more frequently than distance measures. Note, however, that distance can also be interpreted as dissimilarity between two items and is, thus, in some way inversely related to similarity. Therefore, when we refer to similarity measures in the following, this can likewise refer to distance measures in combination with an appropriate transformation function.

This brief description suggests that there be three prime components to any prototype-based classifier, namely a means of generating prototypes from training examples, an appropriate similarity measure that measures the degree of resemblance between two objects in feature space, and a classification scheme for determining the class label of a new example based on its similarity scores to the prototypes:

- *Prototype generation.* Prototype-based classifiers vary greatly in how the set of prototypes is generated. The prototypes may either be true examples as they are provided with the training set or be artificially constructed. A whole span of techniques which transform training examples into suitable prototypes can be considered with two very simple approaches falling at opposite ends of this spectrum.[123]

  At the one end, each training example is assumed a prototype. At this point we ignore the fact that a training document need not necessarily be prototypical. Without any preprocessing or learning step, all training examples are retained and used for comparison with new examples. Typically, the instance-storing approach leads to the family of nearest-neighbor rules which we will discuss shortly. On the other end, each class is represented by a single prototype computed as the average of all training examples assigned to it, which can thus be referred to as an *instance-averaging* approach.

  In between these simple instance-storing and instance-averaging approaches, there are techniques that use either multiple prototypes to represent each class or—depending on the side you look at it from—techniques that use an edited, i.e. usually condensed or reduced, set of training examples to represent classes. The family of *instance-based learning* algorithms proposed by Aha *et al.* (1991) comprises some well-known examples of this type. Note that we consider approaches which try to handle heterogeneous class definitions by means of multiple prototypes per class in Chapter 5. At this point, we concentrate on the simple instance-averaging methods.

  In terms of the generative document source, the inductive bias of instance-averaging approaches can be described as follows. The assumption is that there is a one-to-one correspondence between classes and mixture components with uni-modal distributions so that each class can be represented by a single prototype. So, at a severe cost in representational and learning power, the simple instance-averaging method is very efficient and elegant.[124] Its decision boundaries are determined by single hyperplanes, so that only linearly separable and singly connected classes can be distinguished. Using the vector space model of distinct index terms as representational language theoretically allows even the instance-averaging method to model arbitrary hyperplanes. Note, however, that decision boundaries cannot be learned. In other words, the hyperplanes constructed by the instance-averaging method are completely determined by the average values of the training examples. There is no optimization process with respect to a given effectiveness measure involved in the prototype generation step. Hence, other simple methods that at least try to adapt

[123]See Langley (1996), p. 139.
[124]See Langley (1996), pp. 100–101.

decision boundaries generally seem to compare favorably to the simple instance-averaging method.[125] Simple instance-averaging methods are therefore seldom applied in general classification tasks. Nevertheless, originating from information retrieval, some straightforward and often successfully applied approaches to text classification problems are based solely on this simple instance-averaging idea.

- *Similarity measure.* The similarity measure is used to determine the degree of resemblance between two document vectors. To achieve reasonable classification results, a similarity measure should generally respond with larger values to documents that belong to the same class and with smaller values otherwise. In the following, we describe several similarity measures with a focus on those that are frequently used in information retrieval and text classification.[126]

  The most well-known distance measure is the distance between two points in Euclidean space. It is a preferred choice in many applications because of its simple geometric interpretation. To evaluate the Euclidean distance between two documents, we consider their document vectors as points in the m dimensional feature space, yielding

$$\mathrm{dist}(\mathbf{d}_1, \mathbf{d}_2) = \sqrt{\sum_{j=1}^{m} (w_{1,j} - w_{2,j})^2} \tag{3.28}$$

  where $w_{i,j}$ is the weight of term t_j in document $\mathbf{d}_i$. The range of the Euclidean distance function is $[0, \infty)$ with $\mathrm{dist}(\mathbf{d}_1, \mathbf{d}_2) = 0 \iff \mathbf{d}_1 = \mathbf{d}_2$. A major disadvantage of the Euclidean distance for text classification tasks is that two document vectors may have a large degree of similarity even though they do not have any terms at all in common.[127] Therefore, it is not widely used in the context of text classification. However, in some cases we will use Euclidean distance to better illustrate the fundamental ideas of some classifiers to be described subsequently.

  The dominant similarity measure in information retrieval and text classification is the *cosine similarity* between two document vectors. Geometrically, the cosine similarity evaluates the cosine of the angle between two document vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ and is, thus, based on angular distance. This allows us to abstract from varying document length. The cosine similarity can be calculated as the normalized dot product, yielding[128]

$$\mathrm{sim}_{\cos}(\mathbf{d}_1, \mathbf{d}_2) = \cos(\angle(\mathbf{d}_1, \mathbf{d}_2)) = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{\|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|} \tag{3.29}$$

$$= \frac{\sum_{j=1}^{m} w_{1,j} \cdot w_{2,j}}{\sqrt{\sum_{j=1}^{m} w_{1,j}^2}\,\sqrt{\sum_{j=1}^{m} w_{2,j}^2}} \tag{3.30}$$

[125] See Langley (1996), p. 101.
[126] See Salton and McGill (1983), pp. 201–204, Wang *et al.* (1992), or Losee (1998), pp.45–62.
[127] See Willett (1988).
[128] For example, see Salton and McGill (1983), p. 124 and p. 203.

where $w_{i,j}$ again denotes the weight of term t_j in document $\mathbf{d}_i$. The range of the cosine similarity is the unit interval $[0,1]$ for non-negative vector components[129] with $\mathrm{sim}_{\cos}(\mathbf{d}_1,\mathbf{d}_2) = 0 \iff \mathbf{d}_1 \perp \mathbf{d}_2$ and $\mathrm{sim}_{\cos}(\mathbf{d}_1,\mathbf{d}_2) = 1 \iff \mathbf{d}_1 = \lambda\,\mathbf{d}_2$. Note that $\mathbf{d}_1$ and $\mathbf{d}_2$ are perpendicular so that their degree of similarity is zero only if they do not have any terms at all in common. The computation of the cosine similarity simplifies to the dot product of the involved documents if the document vectors are normalized to unit length. Furthermore, for normalized document vectors, the following monotonic relationship holds between cosine similarity and Euclidean distance:

$$\mathrm{dist}(\mathbf{d}_1,\mathbf{d}_2) = \sqrt{2\,(1 - \mathrm{sim}_{\cos}(\mathbf{d}_1,\mathbf{d}_2))} \tag{3.31}$$

Two other similarity measures common to information retrieval are the Dice and Jaccard coefficients, which are defined as[130]

$$\mathrm{sim}_{\mathrm{Dice}}(\mathbf{d}_1,\mathbf{d}_2) = \frac{2\sum_{j=1}^{m} w_{1,j}\cdot w_{2,j}}{\sum_{j=1}^{m} w_{1,j} + \sum_{j=1}^{m} w_{2,j}} \tag{3.32}$$

$$\mathrm{sim}_{\mathrm{Jaccard}}(\mathbf{d}_1,\mathbf{d}_2) = \frac{\sum_{j=1}^{m} w_{1,j}\cdot w_{2,j}}{\sum_{j=1}^{m} w_{1,j} + \sum_{j=1}^{m} w_{2,j} - \sum_{j=1}^{m} w_{1,j}\cdot w_{2,j}} \tag{3.33}$$

Note that the three similarity measures introduced in (3.30), (3.32), and (3.33) are in some way normalized by vector length. This type of normalization was found to be significant in information retrieval research.[131] Furthermore, they are compelling since they are easy to calculate and often found as effective as other more complicated measures.[132]

In the following, $\mathrm{sim}(\mathbf{d}_1,\mathbf{d}_2)$ denotes any similarity function for measuring the degree of resemblance between two documents. Notice, however, that throughout this dissertation, we generally assume that the cosine similarity $\mathrm{sim}_{\cos}$ is applied when prototype-based classifiers are used.

- *Classification scheme.* Two interpretive methods are typically used to determined the class label of a new document based on its similarity scores to each of the generated prototypes, namely the threshold and the more common competitive interpretation.[133]

  Thresholding a similarity score to decide upon the class labels may be used for binary classification tasks when only one of the classes is represented by a single prototype. Also, if multiple assignments of documents to class labels are desired, thresholding similarity scores would be a straightforward interpretation.

[129]Using the text representation introduced in the previous section, a naturally occurring document is represented by a vector with components $w_i \geq 0$. For artificially constructed documents, however, this property has to be established by setting components with negative values to zero.

[130]See Salton and McGill (1983), pp. 202–203.

[131]See Willett (1983), p. 142.

[132]See Willett (1988), and Salton and McGill (1983), p. 204.

[133]See Langley (1996), p. 14, who further considers logical interpretation.

Yet, when assigning new documents to exactly one of two or more classes, the competitive interpretation is commonly applied. A simple competitive decision scheme is to predict the class associated with the nearest prototype. In the prototype-based framework, this *winner-take-all* principle is often referred to as *nearest-prototype classification*. More elaborate schemes could take a simple or weighted vote among the K, say, nearest prototypes. This may be useful when the set of generated prototypes is not truly prototypical, for example due to outliers in the training set. And this may be the case for the nearest-neighbor rules as we will see below.

In the following, we describe some realizations of simple prototype-based classifiers. The Rocchio algorithm and its variants such as the single-prototype classifier are instance-averaging methods, while nearest-neighbor rules are typical instance-storing methods.

The Rocchio Algorithm. The Rocchio algorithm was originally developed as a method for relevance feedback in information retrieval.[134] As such, it is commonly used to automatically optimize queries initially formulated by a user on the basis of relevance judgements for retrieved documents.[135] Slightly modified variants of this algorithm are frequently applied and often serve as benchmark classifiers in text classification tasks.[136] In this paragraph, we first describe Rocchio's original method before we look at its variants tailored to address text classification tasks.

Relevance feedback is an iterative information retrieval process. Assume a retrieval system which returns a ranked list of documents to a query representing a particular user interest. Users are supposed to mark a subset of these documents as either relevant or non-relevant according to their interests. Rocchio's method provides a formula for incorporating these relevance judgements into the original query, yielding a refined query. Let $\mathbf{q}$ denote the representation of the initial query. The refined query $\mathbf{q}'$ is obtained by adding the relevant documents to the original query and subtracting the non-relevant documents:

$$\mathbf{q}' = \alpha\mathbf{q} + \beta\frac{1}{n_+}\sum_{\mathbf{d}\in D_+}\mathbf{d} - \gamma\frac{1}{n_-}\sum_{\mathbf{d}\in D_-}\mathbf{d} \tag{3.34}$$

where D_+ is the set of n_+ documents marked relevant by the user, and D_- is the set of n_- documents marked non-relevant by the user. Notice that the document vectors are assumed to be normalized to abstract from different document lengths. By varying the parameters α, β, and γ, the relative importance of both the initial query vector $\mathbf{q}$ and the relevant and non-relevant documents can be adjusted. Note that negative vector components of the resulting query $\mathbf{q}'$ are set to zero.[137] By repeating this query refinement step, a user can automatically be supported in finding relevant documents without the need for manually providing additional query terms.

[134]See Rocchio (1971), pp. 313–323.
[135]See Salton (1971).
[136]For example, see Joachims (1997a), Joachims (1997b), p. 8, and Dumais *et al.* (1998).
[137]See Rocchio (1971), p. 317.

The Rocchio algorithm can easily be adapted for binary text classification tasks.[138] Note, however, that in information retrieval a query is associated with one specific topic. In text classification, on the other hand, a class may comprise more than one topic. Hence, recall the assumption for simple instance-averaging approaches, which states that classes can be represented by a single prototype. Whether this restriction will cause problems in the case of broad class definitions will be investigated in Chapter 5.

Without loss of generality, let rel and non denote the two classes. And particularly in Equation (3.34), we replace the sets D_+ and D_- of documents marked relevant and non-relevant by the user with the sets of relevant and non-relevant training documents, D_{rel} and D_{non}, respectively. Recall that the document vectors are assumed to be normalized. Since there is no initial query $\mathbf{q}$, we obtain the following modification:

$$\mathbf{p}_{rel} = \beta \frac{1}{n_{rel}} \sum_{\mathbf{d} \in D_{rel}} \mathbf{d} - \gamma \frac{1}{n_{non}} \sum_{\mathbf{d} \in D_{non}} \mathbf{d} \tag{3.35}$$

Here, negative vector components of the resulting prototype $\mathbf{p}_{rel}$ of the relevant class are set to zero as well. The remaining parameters β and γ are used to control the relative importance of relevant and non-relevant documents. The classification of new documents requires further consideration. While in information retrieval a collection of documents is ranked according to a specific query, in the context of binary text classification we have to decide upon one of the two class labels for each new document. Thus, we threshold documents according to their similarity to the relevant prototype, yielding the following binary decision rule:

$$H_{\text{Rocchio}}(\mathbf{d}) = \begin{cases} rel & \text{if } \operatorname{sim}(\mathbf{d}, \mathbf{p}_{rel}) > \theta \\ non & \text{otherwise} \end{cases} \tag{3.36}$$

where θ denotes the threshold parameter which can either be manually provided by the user or be automatically determined based on the training documents. In the latter case, we could apply v-fold cross-validation to obtain unbiased similarity scores for each training document. On the basis of these scores, the threshold θ could then be set so as to optimize a given effectiveness measure. Note that the similarity score $\operatorname{sim}(\mathbf{d}, \mathbf{p}_{rel})$ between the new document and the relevant prototype can be used as a confidence value for the relevant class.

For classification tasks with two or more classes, the following modification of the Rocchio algorithm can be used. Instead of evaluating only one prototype for the relevant class, we now evaluate a prototype $\mathbf{p}_{c_i}$ for each class $c_i \in \mathcal{C}$:[139]

$$\mathbf{p}_{c_i} = \alpha \frac{1}{n_{c_i}} \sum_{\mathbf{d} \in D_{c_i}} \mathbf{d} - \beta \frac{1}{n - n_{c_i}} \sum_{\mathbf{d} \in D \setminus D_{c_i}} \mathbf{d} \tag{3.37}$$

Again, negative components of $\mathbf{p}_{c_i}$ are set to zero. The resulting set of prototypes $\{\mathbf{p}_{c_i}\}$, $c_i \in \mathcal{C}$, represents the classifier, which is also known as the tfidf classifier owing to

[138] See Joachims (1997b), p. 8.

[139] See Ittner *et al.* (1995), p. 303. Also see Cohen and Singer (1996), Lewis *et al.* (1996), and Joachims (1997a) for text classification applications of the Rocchio classifier.

the tfidf weighting scheme typically used for document and classifier representation in this context. A new document is assigned to the class whose prototype has the largest similarity to the new document:[140]

$$H_{\mathrm{tfidf}}(\mathbf{d}) = \arg\max_{c_i \in \mathcal{C}} \mathrm{sim}(\mathbf{d}, \mathbf{p}_{c_i}) \tag{3.38}$$

where $\arg\max_{x \in X} f(x)$ returns the value of x that maximizes $f(x)$.

A probabilistic analysis of the tfidf classifier leads to a probabilistic variant of the Rocchio algorithm, which is known as the PrTFIDF classifier.[141] This approach provides theoretically motivated settings for the term weighting scheme, the similarity measure, and the parameters β and γ for combining the class-specific document averages, which are often more effective than the plain tfidf classifier.

Note that the learning phase for variants of the Rocchio algorithm only involves simple computations of document vector averages. In fact, there is no explicit optimization process with respect to a given effectiveness measure. In assuming that a single prototype is sufficient to represent a class, these algorithms have a strong search bias in addition to their restrictive preference bias. For this reason, these algorithms can be implemented extremely efficiently with a time complexity linear in the number of training examples n and the number of index terms m, i.e. learning time is bounded by $O(nm)$. Further, classifying a new document is linear in the number of classes k and the number of index terms m; hence, running time for classification is bounded by $O(km)$.

The Single-Prototype Classifier. The *single-prototype classifier* (SPC) is another variant of the Rocchio algorithm. Moreover, it truly corresponds to the simple instance-averaging approach to prototype generation described above. It is only described separately in this paragraph because we will make extensive use of this algorithm in this work. Recall that it is assumed as above that each class can be represented by a single prototype. This accords this algorithm just like all Rocchio variants a very efficient implementation. In Chapter 5, we will examine whether using more than one prototype per class can help to improve classification effectiveness, especially when class definitions are broad and classes are likely to be heterogeneous.

In particular, the single-prototype classifier is an instantiation of the tfidf classifier introduced above with parameters fixed to $\alpha = 1$ and $\beta = 0$, yielding[142]

$$\mathbf{p}_{c_i} = \frac{1}{n_{c_i}} \sum_{\mathbf{d} \in D_{c_i}} \mathbf{d} \tag{3.39}$$

as a prototype for each class $c_i \in \mathcal{C}$. Hence, each prototype $\mathbf{p}_{c_i}$ is the plain average (centroid) of all training documents belonging to class c_i. The decision rule for a new document is the same as for the tfidf classifier:

$$H_{\mathrm{SPC}}(\mathbf{d}) = \arg\max_{c_i \in \mathcal{C}} \mathrm{sim}(\mathbf{d}, \mathbf{p}_{c_i}) \tag{3.40}$$

[140] See Joachims (1997a).

[141] See Joachims (1997a).

[142] The same parameter setting is used by Lang (1995), p. 333, and Dumais *et al.* (1998).

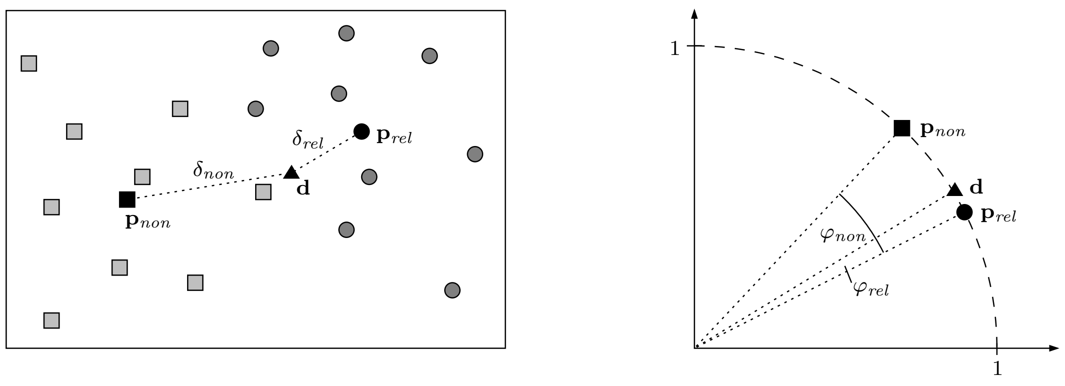


Figure 3.3: The single-prototype classifier exemplified with Euclidean distance (left) and cosine similarity (right). The solid square and circle labeled p_{non} and p_{rel} are the prototypes of the two relevance classes. The solid triangle denoted $\mathbf{d}$ is a new document to be classified. According to Euclidean distance, $\mathbf{d}$ is closer to $\mathbf{p}_{rel}$, since $\delta_{rel} < \delta_{non}$. According to cosine similarity, $\mathbf{d}$ is also more similar to p_{rel}, since $\varphi_{rel} < \varphi_{non}$ and, thus, $\cos \varphi_{rel} > \cos \varphi_{non}$. Consequently, $\mathrm{sim}(\mathbf{p}_{rel}, d) > \mathrm{sim}(\mathbf{p}_{non}, d)$, and d would be labeled *relevant* in both cases.

Figure 3.3 exemplifies a classification decision of the single-prototype classifier for a new document $\mathbf{d}$ using both Euclidean distance and cosine similarity for a two-class problem, where documents are represented by only two index terms. Each index term corresponds to one dimension in the plane as described in Definition 3.2.1.

Nearest-Neighbor Rules. The nearest-neighbor concept is one of the most obvious and well-known statistical approaches to classification.[143] It has been intensively studied in the pattern recognition community for over 40 years. Nearest-neighbor rules are widely applied not only to general classification tasks but also to text classification.[144]

As stated in the introduction to prototype-based classifiers, nearest-neighbor rules simply store all training documents. In other words, there is no effort in terms of learning from these examples. Generalization beyond the training examples takes place at classification time. Therefore, nearest-neighbor rules are sometimes called *lazy learning* methods. The inductive bias of all nearest-neighbor rules corresponds to the fundamental assumption that examples which are close together in feature space according to an appropriate similarity measure will have the same class label. In order to predict the class label of a new document, the closest documents must be determined. Hence, the new document must generally be compared to every stored training document. So, time complexity for classifying a new document is linear in the number of stored training documents n and in the number of index terms m, i.e. it is bounded by $O(nm)$. For a large number of stored training documents, applying the nearest-neighbor rule can be very inefficient computationally, particularly when compared to instance-averaging methods. Note, however, that the process of finding the nearest neighbors can be speeded up dramatically by using efficient indexing techniques such as kd-trees, which store the training examples at the leaves of a tree with similar examples at the same or at nearby nodes.[145]

[143]For example, see Dasarathy (1991) for an overview.
[144]See Masand *et al.* (1992), Li and Jain (1998), or Yang and Liu (1999).
[145]See Mitchell (1997), pp. 230–247.

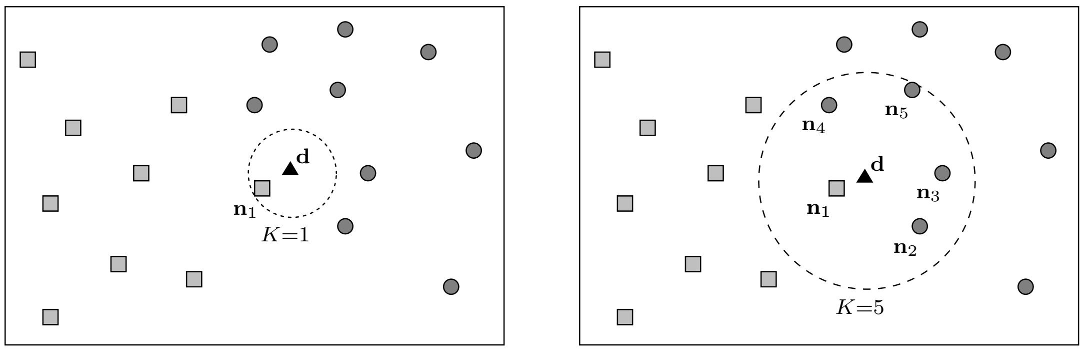


Figure 3.4: Classification decision for a new document $\mathbf{d}$ according to the K-NN rule with $K = 1$ (left) and $K = 5$ (right), using Euclidean distance. The j-th nearest neighbor is denoted by $\mathbf{n}_j$. If only the nearest neighbor $\mathbf{n}_1$ is involved in the decision (left), $\mathbf{d}$ would be labeled *non-relevant*. However, if more neighbors are considered, say $K = 5$, $\mathbf{d}$ would be labeled *relevant*. This demonstrates, that the 1-NN rule is generally more sensitive to outliers.

Classifying a new document $\mathbf{d}$ according to the simple nearest-neighbor rule is accomplished by comparing the new document to each document in the training set and assigning the class label of the closest document. This leads to a decision surface determined by the combination of convex polygons surrounding each training example, which is often illustrated by a *Voronoi diagram* of the feature space.[146] The decision rule can simply be written as

$$H_{\text{1-NN}}(\mathbf{d}) = T(\arg\max_{\mathbf{d}'\in D} \text{sim}(\mathbf{d}, \mathbf{d}')) \tag{3.41}$$

In regions where the class-specific distributions of training examples overlap, the simple one-nearest-neighbor rule commonly leads to a futile over-partition of the feature space. In particular, even though each training document is taken as a prototype, a certain document does not actually have to be prototypical for any of the classes. For example, there may be outliers in the training set, which can cause misclassifications. It is easy to see that the simple nearest-neighbor rule is very sensitive to noise. Figure 3.4 exemplifies this problem, which can often be solved by considering the K nearest neighbors rather than just the one nearest neighbor. For $K \gg 1$, this approach is less sensitive to noise within the training data since noisy training examples are usually outnumbered by more representative examples.[147]

This idea leads to a generalization of the simple nearest-neighbor rule, which is known as the K-nearest-neighbor rule (K-NN).[148] Performance greatly depends on the choice of K. While, as mentioned above, the simple one-nearest-neighbor rule ($K = 1$) may lead to a futile over-partition of the feature space; too large a value for K relative to the training set size may lead to a partition that is too coarse. The value for K could be optimized by using leave-one-out cross-validation on the training set, which would be rather efficient since there is no learning effort. Typically, K is chosen so as to avoid ties. For example, for two-class problems, K is chosen to be odd. The K-NN decision rule is given by

$$H_{K\text{-NN}}(\mathbf{d}) = \arg\max_{c_i\in\mathcal{C}} K_{c_i} \tag{3.42}$$

[146] See Mitchell (1997), p. 233.
[147] See Mitchell (1997), p. 234.
[148] For example, see Cover and Hart (1967).

where K_{c_i} denotes the number of documents among the K nearest neighbors belonging to class c_i. In statistical terms, the fraction $\frac{K_{c_i}}{K}$ amounts to a local estimate of the posterior probability of class c_i (see below). It may, therefore, be used as a confidence value for the corresponding class.

Instead of considering solely the plain number of documents in each class among the K neighbors, a further refinement to the nearest-neighbor concept is to weight each neighbor by its similarity to the document to be classified. This approach is known as the *distance-weighted* K-nearest-neighbor rule.[149] Note that this relaxes the problem of finding a suitable value for K. In fact, at the cost of larger computational effort, we could even consider using all training documents ($K = n$) when determining the class label of a new document.[150] The modified decision rule can be formulated as

$$H_{K\text{-NN}'}(\mathbf{d}) = \arg\max_{c_i \in \mathcal{C}} \sum_{j=1}^{K} \mathrm{sim}(\mathbf{d}, \mathbf{n}_j)\, \delta(c_i, T(\mathbf{n}_j)) \tag{3.43}$$

where $\mathbf{n}_j$ denotes the j-th nearest neighbor with respect to the particular document $\mathbf{d}$ and $\delta(x, y)$ is the Kronecker delta function (identity operator), which returns 1 if its two arguments are the same, i.e. in this case if the training document $\mathbf{n}_j$ belongs to class c_i, and 0 otherwise. Here, the sum of the similarity scores for each class can be used as a confidence value. The scores could be further normalized through division by the sum of the similarity scores of all K nearest neighbors.

Bayesian Probabilistic Classifiers

Bayesian classifiers build on Bayes' decision theory as a fundamental statistical approach to the problem of pattern recognition. It is based on the assumption that the decision problem can be formulated in probabilistic terms according to a specific model which describes the problem setting. When the relevant probabilities are known, classification decisions will be optimal with respect to classification accuracy.[151]

In the following, we first identify the relevant probabilities and show how they are connected by Bayes' theorem. Then, we present suitable decision rules. Since the relevant probabilities are generally unknown, the main part deals with the problem of parameter estimation. Here, we primarily focus on the naïve Bayes classifier, which allows efficient computation of the relevant probabilities by making the simplifying assumptions that all index terms in a document occur conditionally independent given the class label. Although this assumption obviously does not hold in real-world domains, the naïve Bayes classifier is among the most effective text learning algorithms known.[152] Also, this simple Bayesian classifier has been found to perform well across a variety of domains.[153]

[149]See Dudani (1976), for example. Note that in general classification problems distance measures are more commonly applied than similarity measures.

[150]See Mitchell (1997), p. 234.

[151]See Duda and Hart (1973), pp. 10–11.

[152]See Mitchell (1997), p. 155 and pp. 180–184.

[153]See Langley *et al.* (1992), p. 224., or Domingos and Pazzani (1997).

The Bayesian Framework. The following probabilities are primarily involved in the context of Bayesian text classification:

- $\mathrm{P}(c_i)$, the prior probability of a class $c_i \in \mathcal{C}$
- $\mathrm{P}(\mathbf{d}|c_i)$, the likelihood of observing document $\mathbf{d}$ belonging to class c_i
- $\mathrm{P}(c_i|\mathbf{d})$, the posterior probability of a class c_i having observed document $\mathbf{d}$

The framework for Bayesian classification is based on Bayes' theorem, which is used to express the posterior probability of a class given a document in terms of the prior probability of a class and the likelihood of a document given a class, yielding

$$\mathrm{P}(c_i|\mathbf{d}) = \frac{\mathrm{P}(\mathbf{d}|c_i)\,\mathrm{P}(c_i)}{\mathrm{P}(\mathbf{d})} \tag{3.44}$$

where

$$\mathrm{P}(\mathbf{d}) = \sum_{c_i \in \mathcal{C}} \mathrm{P}(\mathbf{d}|c_i)\,\mathrm{P}(c_i) \tag{3.45}$$

is the likelihood of document $\mathbf{d}$. In particular, Bayes' theorem shows how the observation of a document $\mathbf{d}$ changes the prior probability $\mathrm{P}(c_i)$ to the posterior probability $\mathrm{P}(c_i|\mathbf{d})$.[154]

The Bayesian approach to classifying a new document is to predict the most probable class given the document vector $\mathbf{d}$, which yields the maximum a posteriori (MAP) decision rule:[155]

$$\begin{aligned} H_{\mathrm{MAP}}(\mathbf{d}) &= \arg\max_{c_i \in \mathcal{C}} \mathrm{P}(c_i|\mathbf{d}) && (3.46) \\ &= \arg\max_{c_i \in \mathcal{C}} \frac{\mathrm{P}(\mathbf{d}|c_i)\,\mathrm{P}(c_i)}{\mathrm{P}(\mathbf{d})} && (3.47) \\ &= \arg\max_{c_i \in \mathcal{C}} \mathrm{P}(\mathbf{d}|c_i)\,\mathrm{P}(c_i) && (3.48) \end{aligned}$$

Since it is constant for any particular document $\mathbf{d}$, the likelihood $\mathrm{P}(\mathbf{d})$ can be omitted when simply deciding upon the class labels. Its only purpose is to normalize the posterior probabilities so that they add up to one and it is, therefore, only relevant when confidence values are desired for the classification decision.

In some cases, assuming uniform class priors may be more appropriate. Generally, varying the class priors is a straightforward way to respect class-specific misclassification costs. In particular, using uniform class priors suggests that the cost of misclassifying an example be inversely related to the actual probability of observing an example of a particular class. Note that, with uniform priors, the decision rule selects the class which maximizes the likelihood $\mathrm{P}(\mathbf{d}|c_i)$ and is, thus, referred to as the *maximum likelihood* (ML) hypothesis:[156]

$$H_{\mathrm{ML}}(\mathbf{d}) = \arg\max_{c_i \in \mathcal{C}} \mathrm{P}(\mathbf{d}|c_i) \tag{3.49}$$

[154] See Duda and Hart (1973), p. 11.
[155] See Mitchell (1997), p. 157.
[156] See Mitchell (1997), p. 157.

Parameter Estimation. Through Bayes' theorem, the problem of determining a class posterior probability is translated to finding the class prior probability and the likelihood of a document given a class. The advantage of this transformation is that the latter probabilities can be estimated from the training data. Note that these probabilities are generally computed on the basis of document and term frequencies within the training set.

Assuming that documents are generated according to a probabilistic mixture model as introduced above, the learning task can be posed as estimating the parameters of this document source.[157] Let the set of parameters describing the mixture model be denoted by $\boldsymbol{\theta}$. We assume that there is one mixture component for each class so that there is a one-to-one correspondence between mixture components and classes. Each component is parameterized by a disjoint subset of $\boldsymbol{\theta}$. In addition to these parameters, each component is associated with a mixture weight $\theta_{c_i} = \mathrm{P}(c_i)$, which corresponds to the class prior probability of class c_i. When generating a document, the model first randomly selects one of the class-specific mixture components, say c, according to the mixture weights. In a second step, the selected mixture component creates the document according to its parameters with distribution $\mathrm{P}(\mathbf{d}|c)$.

Given the training data, the prior probability of a class c_i is typically determined by the maximum likelihood estimate as the fraction of documents in it, giving

$$\hat{\theta}_{c_i} = \frac{n_{c_i}}{n} \tag{3.50}$$

where n is the total number of training documents and $n_{c_i} = |D_{c_i}|$ is the number of examples belonging to class c_i.

What still remains to be done is to estimate for all classes the likelihood of a document given a class. Assuming the generative model, these probabilities certainly depend on the distributions which govern the generation of the index terms by the class-specific mixture components. However, the generally large number of index terms and the complex dependencies among them, e.g. due to grammar and topicality, render its direct computation intractable unless some simplifying assumptions are made. And this is what the following well-known approach does with respect to the dependencies among the index terms.

The Naïve Bayes Classifier. The naïve Bayes assumption is that the index terms in a document occur conditionally independent given the class label. In other words, the assumption is that, given the class label of a document, the probability of observing the conjunction of its index terms can be expressed as the product of the probabilities for the individual index terms.[158] This drastically reduces the number of parameters that must be estimated from the training data to the number of index terms m times the number of classes k. Let these parameters for the class-conditional term probabilities be written $\theta_{t|c} = \mathrm{P}(t|c)$. Clearly, this assumption is generally not true. Yet, the naïve Bayes classifier often performs well in practice. This behavior can be explained by the fact that,

[157]The description of the document source is based on Nigam *et al.* (2000), p. 107.

[158]See Mitchell (1997), p. 117.

although the naïve Bayes classifier generally produces poor approximations to the distribution of the posterior probabilities,[159] it often predicts class labels effectively since the classification decision is only a function of sign with an appropriate bias parameter.[160]

There are, in fact, two distinct classifiers which build on the naïve Bayes assumption: the multivariate Bernoulli and the multinomial approaches. Both abstract from the order of index terms in a document, which permits use of the vector space representation. The difference lies in the term weighting scheme which specifies the event model underlying term probability estimation.

The Multivariate Bernoulli Model.[161] This model considers binary features which indicate the presence or absence of a term in a particular document. Hence, index term frequencies are not significant. Let $w_{\text{bin}}(\mathbf{d}, t) \in \{0, 1\}$ be one, if term t occurs in document $\mathbf{d}$, i.e. if $tf_{\mathbf{d}}(t) > 0$, and zero otherwise. With the naïve assumption that each term is conditionally independent given the class label, the likelihood of a document given the class is computed as the product of class-dependent term probabilities in closed form by[162]

$$\mathrm{P}(\mathbf{d}|c) = \prod_{t \in \mathcal{V}} \mathrm{P}(t|c)^{w_{\text{bin}}(\mathbf{d},t)} (1 - \mathrm{P}(t|c))^{1 - w_{\text{bin}}(\mathbf{d},t)} \tag{3.51}$$

such that, with respect to the generative model, a document can be seen as a collection of multiple independent Bernoulli experiments. And there is exactly one experiment for each index term with parameter $\theta_{t|c} = \mathrm{P}(t|c)$. Note that this approach explicitly includes the non-occurrence probability of index terms in a document.

For each class $c \in \mathcal{C}$ and each index term $t \in \mathcal{V}$, the parameter $\theta_{t|c}$ is computed from the training data as the Laplace corrected maximum likelihood estimate which avoids probabilities of zero or one:[163]

$$\hat{\theta}_{t|c} = \frac{1 + n_c(t)}{2 + n_c} \tag{3.52}$$

where $n_c(t)$ denotes the number of documents belonging to class c in which term t appears at least once, and n_c is the total number of documents in class c. Note that the Laplace priors supplement each of the two possible outcomes of the Bernoulli experiment, namely the presence or absence of an index term, with a count of one.

This approach has frequently been used for text classification problems.[164] It can be considered as a Bayesian network without dependencies among the index terms, so that the presence or absence of an index term depends solely on the class label of a document. The simple multivariate Bernoulli approach and the approach that tries to capture all dependencies among index terms lie at the two ends of a wide spectrum of dependency that can be modeled by a Bayesian network. In between these extremes, there are approaches that consider a limited number of dependencies among index terms.[165]

[159]The term independence assumption causes naïve Bayes to produce extreme, i.e. almost zero or one, class posterior estimates, see Nigam *et al.* (2000), p. 110.

[160]See Domingos and Pazzani (1997), and Friedman (1997), pp. 63–64.

[161]This description follows McCallum and Nigam (1998a).

[162]Also see Schürmann (1996), p. 76.

[163]See Ristad (1995) for a study on different smoothing priors.

[164]See Maron (1961), Lewis (1992a), Larkey and Croft (1996), or Dumais *et al.* (1998).

[165]See Sahami (1998), pp. 124–142, and Dumais *et al.* (1998).

The Multinomial Model.[166] In contrast to the binary term weights used for the multivariate Bernoulli model, the multinomial model uses plain term frequencies in a document. Let the term frequency of term t in document $\mathbf{d}$ be denoted by $tf_{\mathbf{d}}(t)$. A document is considered as an ordered sequence of index terms, which are drawn from a fixed vocabulary by the generating component. We make the simplifying assumption that the length of a document is independent of its class. In addition, the naïve Bayes assumption states that the term probabilities are independent of a term's context and position. Hence, a document is considered to be drawn from a multinomial distribution of index terms with as many independent trials as the length of the document. In statistical language modeling, this approach is known as a *uni-gram language model*. Now, the likelihood of document $\mathbf{d}$ given its class is given by the multinomial distribution as

$$\mathrm{P}(\mathbf{d}|c) = \mathrm{P}(\|\mathbf{d}\|_1)\,\|\mathbf{d}\|_1! \prod_{i=1}^{m} \frac{\mathrm{P}(t_i|c)^{tf_{\mathbf{d}}(t_i)}}{tf_{\mathbf{d}}(t_i)!} \tag{3.53}$$

where $\|\mathbf{d}\|_1 = \sum_{i=1}^{m} tf_{\mathbf{d}}(t_i)$ denotes the L_1 norm of document $\mathbf{d}$ and, thus, expresses length in terms of the sum of index term occurrences. Furthermore, $\mathrm{P}(\|\mathbf{d}\|_1)$ denotes the probability of observing document $\mathbf{d}$ with the specified length. However, this probability has no effect on the class decision and could therefore be omitted since it is assumed to be independent of the class and is a constant for any given document.

The parameters of the class-specific mixture component for each class are the term probabilities denoted by $\theta_{t|c} = \mathrm{P}(t|c)$. Again, for each class $c \in \mathcal{C}$ and each index term $t \in \mathcal{V}$, the parameters $\theta_{t|c}$ are computed as the Laplace corrected maximum likelihood estimate, yielding

$$\hat{\theta}_{t|c} = \frac{1 + \sum_{d \in D_c} tf_{\mathbf{d}}(t)}{m + \sum_{i=1}^{m} \sum_{d \in D_c} tf_{\mathbf{d}}(t)} \tag{3.54}$$

This time, the Laplace supplement for the denominator is m rather than 2 as above, since there are m possible outcomes for each trial, i.e. each possible index term.

The multinomial approach has also frequently been applied to text classification tasks.[167] An empirical comparison of several commonly used text corpora shows that the multinomial approach generally outperforms the multivariate Bernoulli approach due to the additional term frequency information provided.[168]

Learning both the multivariate Bernoulli and the multinomial model corresponds to estimating the km parameters for the term probabilities in addition to the class priors. For this reason, both approaches can be implemented extremely efficiently; time complexity is bounded by $O(nm)$ as, basically, it just requires aggregation of class-specific term statistics over all n training documents. To classify a new document $\mathbf{d}$, the maximum a posteriori rule is applied as introduced above:

$$H_{\mathrm{NB}}(\mathbf{d}) = \arg\max_{c_i \in \mathcal{C}} \frac{\mathrm{P}(\mathbf{d}|c_i)\,\mathrm{P}(c_i)}{\sum_{c \in \mathcal{C}} \mathrm{P}(\mathbf{d}|c)\,\mathrm{P}(c)} \tag{3.55}$$

[166] This description follows McCallum and Nigam (1998a).

[167] See Lewis and Gale (1994), Joachims (1997a), Mitchell (1997), pp. 180–184, or Nigam *et al.* (2000).

[168] See McCallum and Nigam (1998a).

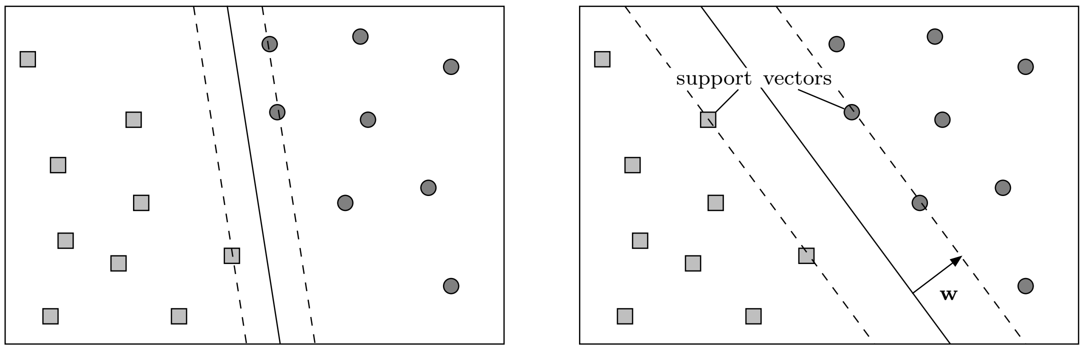


Figure 3.5: The solid lines illustrate two possible decision lines with small margin (left) and maximal margin (right) for training examples of two classes which are linearly separated without error. The dashed lines parallel to the solid lines show how far the decision lines can be moved without causing misclassifications. The distance between a solid decision line and any of the parallel dashed lines is referred to as the *margin*. The support vector learning task is to find the decision line with maximal margin. Training examples on the dashed lines (right) are termed *support vectors*.

Hence, classification time is linear in the number of classes k and the number of index terms m, i.e. it is bounded by $O(km)$.

Support Vector Machines

Support vector machines (SVMs) were proposed by Vapnik in 1979 and have gained much popularity in the learning community within the last decade.[169] They have only recently been successfully applied to text classification problems.[170] Note that support vector machines can only solve binary classification problems. To handle problems with $k > 2$ classes, these could be composed of k binary decisions. In the following, we assume a two-class problem and, without loss of generality, we denote the two classes by $c_1 \equiv -1$ and $c_2 \equiv +1$, respectively.

Support vector machines are defined over a vector space where the problem is to find a decision surface that *best* separates the examples of two classes. To illustrate this basic idea, we assume that the examples of the two classes are linearly separable without error. A decision surface for a linearly separable problem is a hyperplane. We can thus formulate the decision rule for the SVM in its basic form as a threshold function

$$H_{\mathrm{SVM}}(\mathbf{d}) = \mathrm{sign}\{\mathbf{w} \cdot \mathbf{d} + b\} = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{d} + b > 0 \\ -1 & \text{otherwise} \end{cases} \qquad (3.56)$$

where $\mathbf{d}$ is a document to be classified, and the weight vector $\mathbf{w}$ and the bias b are learned from training examples. The distance between the decision surface and the closest training examples is referred to as the *margin*. We define the *best* separation in terms of the *margin* between the training examples of the two classes. Hence, the support vector machine learning task is to find the decision surface that maximizes the margin between the training examples. Figure 3.5 illustrates this idea for linearly separable examples with only two dimensions.

[169] See Vapnik (1995).

[170] For example, see Dumais *et al.* (1998), Joachims (1997b), and Yang and Liu (1999).

The following paragraph motivates why separating examples with maximal margin affords support vector machines the ability to generalize well beyond the training examples. Subsequently, we formulate the support vector learning task as an optimization problem for classification tasks that can be linearly separated without error. In the end, however, we drop these constraints and consider more realistic learning tasks. Note that the subsequent description follows Joachims (1997b).

The Structural Risk Minimization Principle. Why is the separation of examples with maximal margin considered to be best? Originating from computational learning theory, support vector machines are based on the structural risk minimization principle for which error-bound analysis has been theoretically motivated.[171] The idea is to find a hypothesis H for which the lowest true error can be guaranteed. The true error of a hypothesis H is the probability that it will misclassify a previously unseen and randomly selected test example. Our objective is to provide an upper bound for the true error in terms of training error and complexity of the hypothesis space.

In order to measure complexity of a hypothesis space $\mathcal{H}$ in terms of its expressiveness, we introduce the Vapnik-Chervonenkis (VC) dimension of $\mathcal{H}$ as follows.[172]

Definition 3.3.2 (VC Dimension)
The VC dimension of hypothesis space $\mathcal{H}$, $VC(\mathcal{H})$, is defined as the maximum number ξ of examples that can be shattered by $\mathcal{H}$. A set of ξ examples is shattered by $\mathcal{H}$ if and only if it can be separated into two classes in all 2^ξ possible ways by some hypothesis $H \in \mathcal{H}$. The VC dimension is equal to infinity if arbitrarily large finite sets of examples can be shattered.

For instance, the set of linear threshold functions in r-dimensional space has a VC dimension of $r + 1$, since, by using hypotheses from this set, at most $r + 1$ examples can be shattered.

Assume the error of a hypothesis H on the set of n training examples and the VC dimension of the hypothesis space $\mathcal{H}$ containing H. The true error of H is bounded from above with probability of at least $1 - \eta$ by[173]

$$error_{\text{true}}(H) \leq error_{\text{train}}(H) + 2\sqrt{\frac{VC(\mathcal{H})(\log \frac{2n}{VC(\mathcal{H})} + 1) - \log \frac{\eta}{4}}{n}} \tag{3.57}$$

Inequality (3.57) expresses the well-known trade-off between the complexity of the hypothesis space and the training error.[174] On the one hand, a simple hypothesis space (small VC dimension) is unlikely to contain good approximations to the target function and will, therefore, lead to high training and also to true error. On the other hand, too rich a hypothesis space (large VC dimension) is likely to contain good approximations to the target function, but the second term of the upper bound in (3.57) will be large. So, in spite

[171] See Cortes and Vapnik (1995), and Vapnik (1995).
[172] See Vapnik (1995), pp. 76–78.
[173] See Joachims (1997b), p. 4.
[174] See Vapnik (1995), p. 91.

of a low training error, the true error may be large. This problem is generally known as *overfitting*.[175]

We see that deciding on a hypothesis space with an appropriate complexity is a crucial issue. But how do we determine what is appropriate? The structural risk minimization principle provides an answer to this problem. We define a structure of hypothesis spaces $\mathcal{H}_i$ in such a way that their VC dimensions increase:

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_i \subset \ldots \qquad \text{and} \qquad \forall i : VC(\mathcal{H}_i) \leq VC(\mathcal{H}_{i+1}) \tag{3.58}$$

Now, with respect to this structure, the objective is to find the index $i^\star$ which minimizes the upper bound for the true error given in (3.57). For examples that can be separated without error, this means finding the hypothesis that has the lowest VC dimension and correctly classifies all examples.

As mentioned previously, we are considering linear decision surfaces for separation of the two classes. Recall that the VC dimension of linear threshold functions increases linearly with the number features. Therefore, instead of building the structure based on the number of features by using a feature selection strategy, support vector machines use a refined structure which takes into account that most features in text classification are relevant.[176] This structure is based on the following lemma.

Lemma 3.3.1 (Vapnik (1982))
Consider hyperplanes of the form $H(\mathbf{d}) = \text{sign}\{\mathbf{w} \cdot \mathbf{d} + b\}$ as hypothesis space $\mathcal{H}$. If all n examples $\mathbf{d}_i$ are contained in a sphere of radius R and it is required that, for all examples,

$$|\mathbf{w} \cdot \mathbf{d}_i + b| \geq 1, \textit{ with } \|\mathbf{w}\| = A \tag{3.59}$$

then the VC dimension of this hypothesis space is bounded by

$$VC(\mathcal{H}) \leq \min([R^2 A^2], n) + 1 \tag{3.60}$$

The lemma shows that the VC dimension of particular subsets of linear threshold functions does not necessarily depend on the number of features but rather on the Euclidean length $\|\mathbf{w}\|$ of the weight vector $\mathbf{w}$. Consequently, it is possible to achieve good generalization in high dimensional spaces, if the resulting hypothesis has a small weight vector. Note that the hyperplane which has the smallest weight vector is the one separating the two classes with maximal margin, since the distance between the hyperplane and any example $\mathbf{d}$ is $\frac{\mathbf{w} \cdot \mathbf{d} + b}{\|\mathbf{w}\|}$.[177]

The Optimization Problem. As posed above and justified by the structural risk minimization principle, the SVM problem in its basic form is to find the hyperplane that

[175]For example, see Mitchell (1997), pp. 66–69.
[176]See Joachims (1997b), p. 4.
[177]See Boser *et al.* (1992), p. 145.

separates the linearly separable training examples with maximal margin. This problem can be transformed into the following optimization problem.[178]

$$\text{Minimize:} \quad \|\mathbf{w}\| \tag{3.61}$$

$$\text{subject to:} \quad \forall_{i=1}^{n} : y_i \left[\mathbf{w} \cdot \mathbf{d}_i + b\right] \geq 1 \tag{3.62}$$

where $y_i \in \{-1, +1\}$ denotes the true class of the training document $\mathbf{d}_i \in D$. The constraints given in (3.62) require that all training documents be classified correctly. Training examples for which $y_i \left[\mathbf{w} \cdot \mathbf{d}_i + b\right]$ equals one are called *support vectors*. Note that these examples are the ones with minimal distance to the separating hyperplane as shown in Figure 3.5. And only support vectors determine the location of the hyperplane. Noteworthy is the fact that the number of support vectors is generally much smaller than the number of training examples.[179]

Since the above optimization problem is hard to solve numerically, we use Lagrange multipliers to further transform the SVM problem into an equivalent quadratic optimization problem:[180]

$$\text{Maximize:} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \, y_i y_j \left(\mathbf{d}_i \cdot \mathbf{d}_j\right) \tag{3.63}$$

$$\text{subject to:} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \forall_{i=1}^{n} : \alpha_i \geq 0 \tag{3.64}$$

For the solution of this type of optimization problem even in large-scale domains such as text classification, there are efficient optimization algorithms which guarantee that the optimum is found in polynomial time.[181] Let the coefficients $\alpha_i^\star$ denote the solution that maximizes (3.63).[182] On the basis of these coefficients, the two components of the hyperplane specified by (3.61) and (3.62) can be constructed as:

$$\mathbf{w} \cdot \mathbf{d} = \left(\sum_{i=1}^{n} \alpha_i^\star \, y_i \, \mathbf{d}_i\right) \cdot \mathbf{d} = \sum_{i=1}^{n} \alpha_i^\star \, y_i \left(\mathbf{d}_i \cdot \mathbf{d}\right) \tag{3.65}$$

$$\text{and} \qquad b = \frac{1}{2} \left(\mathbf{w} \cdot \mathbf{d}_+ + \mathbf{w} \cdot \mathbf{d}_-\right) \tag{3.66}$$

According to Equation (3.65), the weight vector $\mathbf{w}$ of the resulting hyperplane is a linear combination of the training documents. Obviously, a training document $\mathbf{d}_i$ contributes to the weight vector only if the corresponding coefficient $\alpha_i^\star$ is greater than zero. Note that these are equivalent to the examples identified above as support vectors. Since all support vectors are equidistant to the separating hyperplane, only one support vector of each class is needed to compute the bias b. In Equality (3.66), $\mathbf{d}_-$ and $\mathbf{d}_+$ denote any two support vectors of the two classes -1 and $+1$, respectively.

[178] See Joachims (1997b), p. 5.
[179] See Cortes and Vapnik (1995), p. 275.
[180] See Vapnik (1995), pp. 129–131.
[181] For example, see Platt (1999), or Joachims (1999a).
[182] See Boser *et al.* (1992), pp. 147–148, for a discussion on the properties of this solution.

Non-Linear Hypothesis Spaces. Support vector machines are able to learn not only linear but also non-linear hypotheses. To do this, examples are transformed by a non-linear mapping $\Phi : \mathbb{R}^m \mapsto \mathbb{R}^{\hat{m}}$ from the original m-dimensional document representation space into an $\hat{m}$-dimensional feature space, where commonly $\hat{m} > m$. In this new document representation space, it is then possible to find a separating hyperplane as described above.

Arbitrary types of hypotheses can be constructed by simply replacing the dot products in (3.63) and (3.65) by different dot products $K(\mathbf{d}_1, \mathbf{d}_2)$.[183] These general forms of the dot product, which are referred to as *kernel* or *convolution* functions, have to satisfy Mercer's theorem such that they compute the dot product of the examples $\mathbf{d}_1$ and $\mathbf{d}_2$ after they have been mapped into the new feature space by the chosen non-linear mapping Φ:[184]

$$\Phi(\mathbf{d}_1) \cdot \Phi(\mathbf{d}_2) = K(\mathbf{d}_1, \mathbf{d}_2) \tag{3.67}$$

For example, the following convolution functions could be used to learn polynomial classifiers, radial basis function (RBF) networks, or neural networks with one hidden layer and sigmoid activation functions:[185]

$$\begin{aligned} K_{\text{polynomial}}(\mathbf{d}_1, \mathbf{d}_2) &= (\mathbf{d}_1 \cdot \mathbf{d}_2 + 1)^d && (3.68)\\ K_{\text{RBF}}(\mathbf{d}_1, \mathbf{d}_2) &= \exp(\gamma\,(\mathbf{d}_1 - \mathbf{d}_2)^2) && (3.69)\\ K_{\text{sigmoid}}(\mathbf{d}_1, \mathbf{d}_2) &= \tanh(s\,(\mathbf{d}_1 \cdot \mathbf{d}_2) + c) && (3.70) \end{aligned}$$

Note that using convolution functions typically introduces new parameters such as the polynomial degree d or the variance γ for the RBF networks. A straightforward and efficient approach to finding the best parameter values is based on the upper bound for the true error (3.57).[186] The idea is to learn SVMs for different parameter values and, then, to select that one with the lowest VC dimension.[187]

Non-Separable Problems. So far, we have described the support vector learning task under the constraint that the training examples be separable without error. In reality, however, this constraint is often violated. Therefore, we now assume that some training examples may be misclassified. In this case, the learning task can be described as finding the hyperplane which minimizes the number of errors on the training set and separates the remaining training examples with maximal margin.[188]

In order to handle non-separable classes, Cortes and Vapnik propose that slack variables be introduced into the constraints (3.62).[189] This necessitates adapting the objective function (3.61) and also the transformed optimization problem (3.63) with its constraints (3.64). A simpler approach is to monitor the coefficients α_i while optimizing (3.63).[190]

[183]See Cortes and Vapnik (1995), pp. 282–284.
[184]See Vapnik (1995), pp. 135–136.
[185]See Joachims (1997b), p. 6.
[186]See Vapnik (1995), pp. 137–141.
[187]See Joachims (1997b), pp. 6–7.
[188]See Cortes and Vapnik (1995), p. 281.
[189]See Cortes and Vapnik (1995), pp. 280–282.
[190]See Boser *et al.* (1992), p. 147, or Joachims (1997b), p. 7.

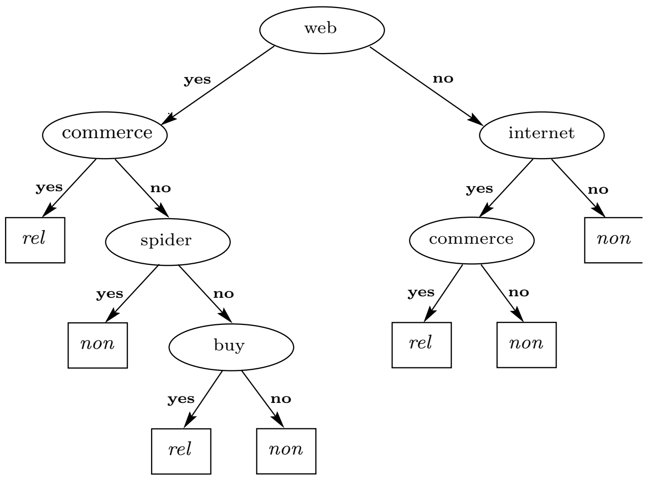


Figure 3.6: A simple decision tree that identifies the topic ‘E-Commerce’ as relevant (following Sahami (1998), p. 37). Leaf nodes are depicted as rectangles labeled with the class that they indicate. Ellipses correspond to decision nodes which test a document for the presence or absence of the index terms with which they are labeled. The outcomes of each test are denoted **yes** and **no** depending on whether the respective index term does or does not occur in a particular document.

Training examples which correspond to coefficients with large values are identified as those close to the decision boundary and, thus, difficult to separate from examples of the other class. If the value α_i of training document $\mathbf{d}_i$ exceeds a predefined threshold, this document would be removed from the training set and the support vector machine would then be learned from the remaining examples.

Further Approaches

Above, we have described some of the most commonly used text learning approaches. Yet, there are numerous other approaches which have been successfully applied to text classification tasks. Below, we briefly summarize some of these approaches.

Decision tree and rule set induction methods have been extensively studied in the machine learning literature, and they have been successfully applied to a broad range of learning tasks.[191] For instance, Lewis and Ringuette (1994), Joachims (1997b), and Dumais *et al.* (1998) describe applications of decision trees to classify text documents. Figure 3.6 shows a very simple decision tree that identifies the topic ‘E-Commerce’ as relevant. Several propositional rule learning methods have been successfully applied to text classification tasks, see Apté *et al.* (1994), Moulinier *et al.* (1996), and Cohen and Singer (1996), for example. Cohen (1995a) and (1995b) has applied relational rule induction algorithms to classify text documents.

We have seen that simple instance-averaging approaches such as variants of the Rocchio algorithm describe linear decision surfaces in feature space. Since the corresponding parameters are merely calculated, rather than optimized with respect to a given effective measure, approaches that actually learn linear threshold functions are often preferred. For example, Lewis *et al.* (1996) applied the Widrow-Hoff algorithm and the exponentiated-

[191]For example, see Mitchell (1997), pp. 52–80 and pp. 274–306, for a general overview.

gradient algorithm to learn linear text classifiers which outperformed a simple Rocchio variant on several tasks. Dagan *et al.* (1997) present further mistake-driven text learners which are based on Littlestone's Winnow family of algorithms.[192] The application of linear, and also non-linear, polynomial classifiers to the information routing and filtering task is described by Bayer *et al.* (1998). They state that the linear polynomial classifier is similar to the linear least-square fit classifier proposed by Yang and Chute (1993). For further successful applications of approaches from statistics such as linear discriminant analysis or logistic regression, see Schütze *et al.* (1995) and Hull *et al.* (1996).

Learning linear threshold functions as above can also be formulated in a neural network framework. Apart from this and the support vector machines with sigmoid convolution functions, we did not consider neural networks approaches for text classification. Nevertheless, several neural networks have been evaluated on text domains, for example, see Schütze *et al.* (1995), Wiener *et al.* (1995), Ng *et al.* (1997), and Yang and Liu (1999).

3.3.3 Ensembles of Classifiers

In the preceding, we have described various approaches for learning text classifiers from training documents. Each of these classifiers has its very own characteristics, which we broadly described by its inductive bias. Because of these characteristics, distinct parameter settings, and different views on the training data, classifiers generally vary in predicting the class labels of documents. So, an interesting issue is the discussion of frameworks for combining multiple classifiers. This usually involves integrating the responses of a set of different classifiers, which is also known as a *committee* or an *ensemble of classifiers*, into a final classification decision so that the resulting entity behaves like a single classifier. As a rule, the objective of combining classifiers is to improve overall classification effectiveness.[193] The idea behind this has its analogy in consulting several experts when dealing with a difficult problem. Nevertheless, in some cases the objective is simply to make classification tasks with more than two classes tractable for binary classifiers.[194]

The key distinguishing factor of ensemble learning approaches is the issue of whether each of the classifiers involved treats the same classification task or whether each classifier is tailored towards solving a special subtask. Note that while there are numerous approaches to learning ensembles of classifiers in the literature, we focus on those developed in the context of text classification in the following.

Ensembles of Classifiers Developed for the Same Task

The crucial factor in designing an ensemble of classifiers where each classifier solves the same classification task is the strategy pursued to generate the set of diverse classifiers. We will consider two different approaches: applying different learning algorithms and manipulating the training data while using the same base learner.[195]

[192]See Littlestone (1988).
[193]See Dietterich (2000a), pp. 1–4, for explanations why combining classifiers can improve effectiveness.
[194]See Schürmann (1996), pp. 330–356, for example, for a general discussion on combining classifiers.
[195]See Dietterich (2000a) for a discussion on methods for constructing ensembles of classifiers.

Given the ensemble of classifiers, the remnant task is to find a method for combining the individual classification decisions. The most straightforward and often applied technique is to use either simple or weighted majority voting. Weighted votes are typically based on a function of the classifiers' effectiveness on the training data or the classification scores. These scores commonly measure some degree of certainty or confidence for a particular classification decision. Note, however, that the scores may require normalization to make them comparable across different classifiers. This normalization step is often referred to as *confidence mapping* and will be discussed in Subsection 3.3.4. An alternative approach is to treat classifier combination as a separate learning task.[196] Yet, simple voting approaches are often preferred since they are less prone to overfitting.

Using Different Learning Algorithms. Except for the confidence mapping approach set out in Section 3.3.4, learning an ensemble of classifiers with different learning algorithms is straightforward. Each individual classifier is learned independent from the other classifiers by means of a particular learning algorithm. As mentioned above, classification decisions are often combined by voting techniques. Successful applications of classifier ensembles constructed by different learning algorithms are described by Hull *et al.* (1996), Larkey and Croft (1996), and Li and Jain (1998).

Modifying the Training Set. Two very popular ensemble learning paradigms that build on inducing multiple classifiers from modified versions of the training set are known as bootstrap aggregating, or *bagging* for short, and *boosting*. While bagging evolved in the statistics community,[197] boosting has its roots in computational learning theory.[198] Note that some approaches referred to as *classifier iteration* in the field of pattern recognition are very similar to the idea of boosting.[199] The crucial factor for the success of these approaches is the instability of the base learning algorithm: decision tree learning algorithms, for instance, produce instable classifiers, whereas nearest-neighbor rules are considered stable. Classification accuracy can be improved if changes in the training data can cause significant changes in the classifier induced.[200]

Bagging modifies the training set by generating replicated bootstrap samples. That is, each classifier in the ensemble is induced from an individual training set D', which is randomly sampled with replacement from the original training set D, so that both the new and the original training set are of the same size. Generally, D and D' differ as some of the training examples from D might not appear in D' while others might occur more than once. On average, about 63% of the original training examples are found in the new sample. The individual classifiers are combined by simple voting.[201]

The basic concept of boosting is that many rather weak classifiers can be combined into a single highly effective classifier. While there are several variants of boosting, we focus

[196] See Schürmann (1996), pp. 338–341.
[197] See Breiman (1996).
[198] See Schapire (1990) and Freund and Schapire (1996).
[199] See Schürmann (1996), pp. 183–186.
[200] See Breiman (1996).
[201] See Breiman (1996).

on the AdaBoost family of algorithms.[202] AdaBoost iteratively creates classifiers from modified versions of the training data. Yet, unlike bagging, these modifications are not independent. Instead, they depend on the classification accuracy of previously induced classifiers. In particular, AdaBoost maintains a weight for each training example, which is adapted after each classifier is induced so as to give misclassified examples more weight in the next iteration. If a learning algorithm can make use of weighted training examples, the influence of an example increases with its weight. To use the weights otherwise, a classifier can be learned from a new training set sampled from the original training set with a probability distribution corresponding to the weights. In both cases, the resulting classifiers are combined by weighted voting. The weights are determined on the basis of the individual classification accuracies on the weighted training sets. Note that AdaBoost is capable of making much larger changes in the training set than bagging, for example by placing large weights on only a few of the training examples. Therefore, it may require less instability in the learning algorithm than bagging.[203] See Schapire *et al.* (1998) and Schapire and Singer (2000) for applications of boosting in text domains.

Ensembles of Specialized Classifiers

When there are more than two classes, we may consider decomposing the classification task into smaller subtasks for two reasons. On the one hand, the base classifier might only be capable of handling binary tasks such as the support vector machine. Then the problem has to be decomposed into binary subtasks and a classifier learned for each of these subtasks. On the other hand, there might be some hierarchical structure defined on the classes from which specialized classifiers may benefit. Generally, the same learning algorithm is used to learn each of the specialized classifiers.

Binary Decomposition. In order to apply a binary classifier to a classification problem with $k > 2$ classes, the problem could be split into k binary subtasks.[204] A separate classifier is then learned for each subtask. When predicting the class label of a new document, the responses of the k binary classifiers must be combined to obtain a final classification decision. When allowing multiple class labels for a document, the classification decision of each classifier could be used unchanged as provided. However, as we defined text classification as mapping documents onto exactly one class, we have to agree upon a single class label, given the responses of the individual classifiers. To prefer any of the suggested class decisions, we require that a classifier provide a measure of certainty or confidence for its decision. From all the responses, the most confident is then selected. Note that this simple decision rule requires that the confidence values be comparable across the different classification subtasks. As above, this also requires applying confidence mapping techniques, which will be discussed in the following subsection.

[202] See Freund and Schapire (1996).
[203] See Dietterich (2000b), pp. 140.
[204] This is a special case of learning classifier networks, see Schürmann (1996), pp. 350–356.

Class Hierarchy. In many text classification problems, in particular when the number of classes is large, classes can be organized in a hierarchy of increasing specificity.[205] Such class hierarchies are typically found, for instance, in internet guides or directories such as Yahoo! (2000). For the sake of simplicity, we assume that this class hierarchy can be represented in the form of a tree. The benefit from exploiting this structure is two-fold.

First, the complex classification problem may be decomposed into a set of much simpler subtasks according to the splits in the classification hierarchy. So, at each node of the tree, there is a classifier which must distinguish only a small number of classes. Noteworthy is that this classification decision can often be made on the basis of only very few index terms. With appropriate feature selection and learning algorithms, it is therefore possible to focus solely on the relevant index terms for each particular classification subtask. Due to this integration of feature selection with the hierarchical structure, the resulting classifier is more robust and less prone to overfitting. A final classification decision is achieved by sorting a document down the tree until a leaf node is reached.[206]

Aside from this, there are other learning approaches that do not construct ensembles of classifiers but can still benefit from utilizing information from the class hierarchy. Specifically, parameter estimation such as that for learning a naïve Bayes classifier can be improved by adding information about ancestors of sparsely populated classes. This statistical smoothing technique is known as *shrinkage*.[207]

3.3.4 Confidence Mapping

So far, we have discussed several learning algorithms for inducing text classifiers. These classifiers respond to a document with classification scores which vary greatly. For example, probabilistic classifiers try to estimate the posteriori probabilities of classes given a document. Others, such as the Rocchio variants, may respond with similarity scores commonly in the set of real numbers between zero and one. By contrast, support vector machines return a real number according to the position of a document relative to the separating hyperplane in Euclidean space. As these examples show, not only the ranges of classification scores may differ, but depending on the classifier and the domain also the interpretations of scores within the same range may also vary. Consequently, classification scores are generally not comparable across different classifiers. Note that this is true not only for classifiers induced by different learning algorithms but also for classifiers induced by the same learning algorithm for different classification tasks. As discussed previously, this is of great importance for learning ensembles of classifiers. Either when combining different classifiers learned for the same classification task or when learning specialized classifiers for the decomposition of a particular problem, classification scores are required to be comparable across the individual classifiers. To make them comparable, scores have to be normalized. Doing this is commonly referred to as *confidence mapping*.[208]

[205] See Schürmann (1996), pp. 343–350, for a detailed discussion on hierarchical classifiers.
[206] See Koller and Sahami (1997).
[207] See McCallum *et al.* (1998).
[208] See Schürmann (1996), pp. 151–165, for a detailed treatment of confidence mapping.

In the preceding, we loosely referred to classification scores as a measure for the degree of certainty or confidence of particular classification decisions. In this context, larger scores were associated with more confident decisions. We now interpret the concept of confidence more narrowly. A confidence value is assumed to be a real number in the range between zero and one. Also, it is required to be reliable and coherent with statistical experience. So, if a classification decision is based on a confidence score of 0.8, say, then this decision should be correct in 80% of the cases within a larger sample of decisions made with similar confidence values.[209] The objective of confidence mapping is, therefore, to transform classification scores into confidence values.

In the following, we assume that a classifier solves a classification task by, first, providing a classification score for each possible class and, then, applying a decision rule to these scores. For the purpose of confidence mapping, we treat each of these scores individually. For a particular class, let s denote the corresponding classification score which can be used to separate that class from the remaining classes in $\mathcal{C}$. This is a two-class problem which can be posed for each of the k classes. To distinguish the two new classes, let $eigen \equiv c$ denote a particular class and $fremd$ indicate the membership to any class but c.

The confidence value of deciding in favor of class $eigen$ given score s is equivalent to the posterior probability $\mathrm{P}(eigen|s)$. According to Bayes' theorem, we obtain

$$\mathrm{cnf}(s) \equiv \mathrm{P}(eigen|s) = \frac{\mathrm{P}(s|eigen)\,\mathrm{P}(eigen)}{\mathrm{P}(s|eigen)\,\mathrm{P}(eigen) + \mathrm{P}(s|fremd)\,\mathrm{P}(fremd)} \tag{3.71}$$

The class prior probabilities can be computed based on class frequencies as described for the Bayesian classifiers (p. 63). Note that $\mathrm{P}(fremd) = 1 - \mathrm{P}(eigen)$. It is possible to consider class-specific misclassification costs by using different class priors. For example, when the cost of misclassifying a document of any class is inversely proportional to class frequency, then uniform class priors are appropriate. Below, we will discuss how the class-specific distributions of score values given the class, $\mathrm{P}(s|eigen)$ and $\mathrm{P}(s|fremd)$, can be estimated from class-specific histograms. Note that using uniform class priors is equivalent to recovering the distributions from histograms that are normalized through division by the number of examples belonging to the corresponding class.

A class-specific histogram of classification scores describes the characteristic behavior of a classifier given a class. Figure 3.7 shows two typical histograms of scores provided by a single-prototype classifier as similarities to the prototype representing the class $eigen$. To construct a histogram, ideally, unbiased classification scores of labeled examples which have not been used for training are distributed among a given number of histogram bins. At this point, we assume that examples presented to a classifier belong to exactly one of the legitimate classes and obey the statistical laws governing the document source. This is known as the closed-world assumption.[210] Situations in which the closed-world assumption is violated will be discussed in the context of quality control in Chapter 6.

Each histogram count for a particular class corresponds to the product of class prior and class-specific probability density function for scores falling in the respective histogram

[209] See Schürmann (1996), p. 153.

[210] See Schürmann (1996), p. 158 and p. 290.

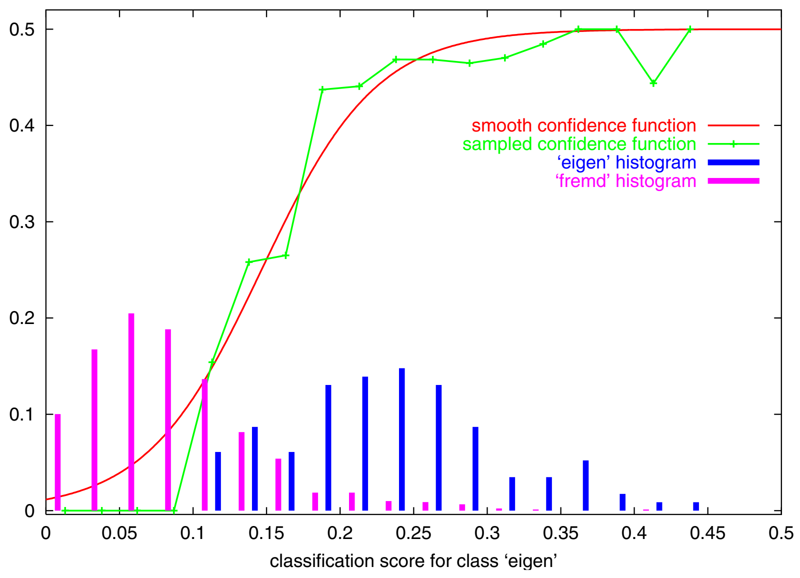


Figure 3.7: Smooth and linearly interpolated sampled confidence functions according to $eigen$ and $fremd$ histograms of classification scores provided by a single-prototype classifier as similarities to the prototype of the class $eigen$. Scores greater than 0.5 did not occur. Note that the histograms display fractions of examples falling in each bin rather than plain integer counts of these examples. The 0.5 level on the histogram scale corresponds to the value one of the confidence functions.

bin. If the histograms are normalized through division by the total number of examples distributed among all histograms so as to ignore the class priors, they provide sampled approximations to the probability density functions $\mathrm{P}(s|eigen)$ and $\mathrm{P}(s|fremd)$. Let $\mathrm{hist}(i|eigen)$ and $\mathrm{hist}(i|fremd)$ denote the histogram counts for classification scores of documents in the respective class falling in the histogram bin i. An estimate for the confidence of all scores falling into bin i is given by[211]

$$\mathrm{cnf}[i] = \frac{\mathrm{hist}[i|eigen]}{\mathrm{hist}[i|eigen] + \mathrm{hist}[i|fremd]} \tag{3.72}$$

Note that bins which are empty in both histograms are ignored to avoid divisions by zero. Since each score can be mapped onto the corresponding bin index i, confidence mapping can be implemented as simple table look-up operations.

Figure 3.7 further illustrates the idea of confidence mapping. For each bin, confidence values are estimated from the $eigen$ and $fremd$ histograms of classification scores. The sampled confidence values are plotted linearly interpolated. The curve shows that the confidence estimates are distorted by random error caused by the finiteness of samples per histogram bin. Thus, especially scarcely populated bins provide uncertain confidence estimates. In the following, we describe a technique for deriving a smooth approximation to the confidence function in order to remedy this deficiency.

We use prior knowledge to decide upon the basic form of the confidence function $\mathrm{cnf}(s)$. In particular, a confidence function typically converges to its extreme values zero and one towards the ends of the domain determined by the range of the classification scores. Furthermore, a confidence function often increases monotonically with its input, and its

[211] See Schürmann (1996), pp. 158–159.

slope is steeper where the class-specific histograms strongly overlap. With respect to these characteristics, using a *sigmoid function* seems appropriate, giving[212]

$$\mathrm{cnf}(s) = \frac{1}{1 + \exp(-\alpha(s - \beta))} \tag{3.73}$$

where α controls the steepness of the sigmoid curve and β its position on the score scale.

Fitting the sigmoid function through the sampled confidence values $\mathrm{cnf}[i]$ at score s_i, where s_i is typically the score at the center of the histogram bin with index i, can be posed as a least mean square approximation problem with respect to the parameters α and β. Thus, the objective is to minimize the target function[213]

$$F(\alpha, \beta) = \sum_i \mathrm{hist}[i](\mathrm{cnf}(s_i) - \mathrm{cnf}[i])^2 \tag{3.74}$$

where $\mathrm{hist}[i] = \mathrm{hist}[i|\mathit{eigen}] + \mathrm{hist}[i|\mathit{fremd}]$ represents a weight that allows focusing on strongly populated bins since these provide more confident points. The resulting smooth approximation to the confidence function is also shown in Figure 3.7.

Note that, as required, the individual confidence values thus obtained are in the range of real numbers between zero and one and are, to a great extent, coherent with statistical experience. When looking at the set of k confidences values now available for the actual k-class problem, however, these confidence values generally will not add up to one. Yet, this property can be established by dividing each confidence value by the total sum of confidence values.

3.3.5 Summary

This section covered the problem of learning a text classifier from a set of labeled training documents. We first analyzed some general characteristics of the text learning task and derived desirable properties for text learning algorithms. We then described several commonly applied learning algorithms in detail: variants of the Rocchio algorithm, nearest-neighbor rules, two variants of the naïve Bayes classifier, and support vector machines. Subsequently, we discussed frameworks for combining classifiers either to improve classification effectiveness or to make problems with more than two classes tractable for binary classifiers. Finally, we introduced confidence mapping as a general technique for transforming raw classification scores returned by any kind of classifier into confidence values reflecting the posterior probabilities of the classes.

In the course of this section, we also pointed out issues related to common assumptions which are violated in practice. As previously discussed in Chapter 1, this basically involved the availability of labeled training examples, problems with heterogeneous class definitions, and changes in the distributions governing the components of the hypothetical document source. The following three chapters discuss these issues in depth.

[212]See Ittner *et al.* (1995), and Dumais *et al.* (1998).

[213]See Schürmann (1996), p. 161. Note, however, that Schürmann describes a more general model for generating a smooth approximating function to the sampled confidence function.

Chapter 4

Semi-Supervised Text Learning

Supervised learning algorithms typically require large amounts of training data to learn reasonably accurate classifiers. Yet, in many text classification tasks, labeled training documents are expensive to obtain, while unlabeled documents are often readily available in large quantities. As a rule, unsupervised learning methods are employed to discover structure in unlabeled data. Yet, learning solely from unlabeled documents cannot be used to classify new documents into predefined classes because knowledge about the classes is missing. Learning from either labeled or unlabeled data through supervised or unsupervised learning, respectively, may be regarded as two extremes of a wide spectrum of learning approaches. In between these extremes, the aim would be to learn from both labeled and unlabeled examples. This allows taking advantage of the strengths of both extremes, i.e. to learn accurate classifiers and to exploit unlabeled data, while getting rid of common drawbacks. Aside from the enormous need for labeled data on the supervised side, common drawbacks of unsupervised approaches are, for instance, the inability to identify known classes and the difficulty in choosing a suitable number of clusters. In this chapter, we propose a general framework for extending any type of learning algorithm to make use of both unlabeled documents and labeled documents, which we refer to as *semi-supervised learning*.

4.1 Introduction

This section revisits the problem associated with the *availability assumption* that arises in many supervised learning tasks: providing a sufficient number of labeled training examples. To stress practical relevance, we illustrate the effect of a small training set size by showing the learning curves of different learning algorithms on several real-world text learning tasks when the amount of training data is varied. Having, by that, motivated the need to reduce the amount of training data required, we give arguments for the use of unlabeled data in addition to labeled data. Then, we describe several general methodologies for combining both sources of information. Further related work will be discussed in the subsequent section.

4.1.1 Problem Description

In the preceding chapter, we have described several supervised learning algorithms that are widely applied to induce text classifiers from a set of labeled training documents. In the last two decades, much progress has been made in solving text classification tasks, in particular by applying machine learning techniques. In the traditional supervised learning setting, we seem to have reached a reasonably high plateau with respect to classification effectiveness. Current state-of-the-art approaches based on different learning paradigms achieve acceptably high overall performance, and many of them perform about equally well.[1]

To achieve these results, however, supervised learning algorithms commonly require large numbers of labeled training documents. In other words, for the successful application of supervised learning to solving classification tasks, it is assumed that there is a sufficient amount of labeled data. We have previously referred to this as the *availability assumption*. Why does this resemble a problem? To assign a class label to an example, i.e. for a document in the context of text learning, a human has to peruse this document first. Obviously, this hand-labeling of documents is not only error prone, but also a tedious and time-consuming process. Consequently, provision of labeled data requires expensive human resources, making it the bottleneck in the supervised learning setting.

But, the severity of the availability assumption can only be understood to its full extent when we realize how large a training set must be in order to achieve reasonable results. Therefore, it is crucial to know just when a training set may be deemed large enough. It is known from computational learning theory that, usually, the number of training examples should be at least a multiple of the number of features if reasonable results are to be guaranteed.[2] Recall that, often, several thousand features are used to represent text. So, for complex classification tasks, obtaining a reasonable number of training documents easily becomes intractable.

4.1.2 The Effect of a Small Training Set Size

Computational learning theory may give some insight as to what is learnable and can provide worst-case bounds on the number of training examples required. For practical problems, however, we generally have to find out empirically how large the training set should be.[3] As we have seen above, providing as many training examples as theoretical considerations suggest is generally not possible in text domains. The high dimensional feature space is the reason for this. Thus, the vital question is how well can we do with less than the recommended amount of training data? In the following, we examine the classification accuracy of some common text learning algorithms when varying the number of training examples for classification tasks on three frequently used real-world text corpora.

[1]See Liere and Tadepalli (1996).

[2]See Lewis (1992b), p. 14.

[3]See Schürmann (1996), p. 310.

Datasets and Experimental Setups

To run the experiments, we use an enhanced version of the rainbow system originally written by McCallum (1996).[4] We choose to employ the Rocchio variant termed single-prototype classifier (SPC), the multinomial naïve Bayes classifier (NB), a linear support vector machine (SVM), and the one-nearest-neighbor (1-NN) rule as four of the most frequently applied learning algorithms.[5] We use the 20 Newsgroups dataset, the WebKB dataset, and a subset of the TREC dataset as text corpora.[6] For the 20 Newsgroups and the WebKB dataset, we basically follow the setups described by Nigam *et al.* (2000).

The 20 Newsgroups dataset consists of about 20,000 articles divided almost evenly among 20 different UseNet discussion groups. The task is to classify an article into the one of the twenty newsgroups to which it was posted. When tokenizing the documents, UseNet headers are skipped, and tokens are formed from contiguous alphabetic characters. We remove common stop words and all words that occur only once. For all experiments, we use as possible vocabulary the entire set of remaining features. We create a test set of 4,000 documents by selecting by posting date the last 20% of the articles from each newsgroup. Next, an unlabeled set of 10,000 documents is randomly selected and set aside for later use. Labeled training sets are formed by partitioning the remaining 6,000 documents into non-overlapping sets. All sets are created with an equal number of documents per class. Where applicable, up to ten trials with disjunct labeled training sets are run for each experiment. Results are reported as averages over these trials.

The WebKB dataset contains 8,230 web pages gathered from different university computer science departments, including the entirety of four departments and, in addition, an assortment of pages from other universities. Only the 4,199 documents of the classes *course, faculty, project,* and *student* are used. The number of documents in each class varies from 504 in class *project* up to 1,641 documents in class *student*. The task is to classify a web page into the most appropriate one of the four classes. We do not apply stop-word removal. Numbers are converted into special tokens. The potential vocabulary consists of all words found in the text corpus more than once. We create four test sets, each containing all pages from one of the four entire departments. Corresponding to each test set, the pages remaining are randomly split into different non-overlapping labeled training sets and an unlabeled set of 2,500 pages for later use. Results are obtained in *leave-one-university-out* fashion and are reported as averages over the four test sets.

From the large TREC dataset, we select a subset of five classes: *001, 003, 006, 128,* and *142*. For the sake of homogeneity, we consider only articles published by the *Wall Street Journal*, yielding a total of 1,494 documents. The number of documents in each class varies from 234 in class *001* up to 386 documents in class *003*. The task is to classify documents into the most appropriate one of the five classes. We apply stop-word

[4]See Appendix A for more details on this software.

[5]See Section 3.3 for a description of common text learning algorithms. Recall from Section 3.2.4 that we use 'ltc' style weighting of documents for all classifiers but naïve Bayes. According to Nigam *et al.* (2000), for the naïve Bayes approach on the 20 Newsgroups dataset, the term frequencies of each document are scaled in such a way that each document has constant length, with potentially fractional frequencies.

[6]See Appendix B for detailed information about these text corpora.

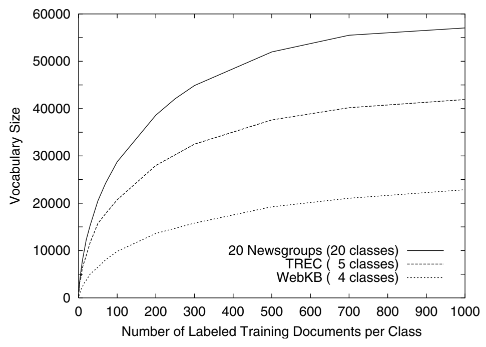


Figure 4.1: The curve shows average vocabulary sizes when varying the training set size for each of three text corpora used. That is to say, the average numbers of distinct features remaining after dimensionality reduction in a set of ten randomly selected training sets, each with the indicated number of documents per class, are shown. Note that, after the rudimentary dimensionality reduction step, there are 57,044 distinct terms in the entire 20 Newsgroups dataset, 23,830 distinct terms in the entire WebKB dataset, and 45,424 distinct terms in the TREC subset.

removal and discard all words that appear only once. The remaining words are used as possible index terms. From each class, 70 documents are randomly selected to create non-overlapping training sets of different sizes. The remaining documents are split into a test set of 350 documents and an unlabeled set of 700 documents for later use. Where applicable, up to ten trials with non-overlapping labeled training sets are run for each experiment. Results are reported as averages over these trials.

Except for removing stop words and words that occur only once in a text corpus, we do not apply any further dimensionality reduction. In any particular trial, only the subset of distinct terms that occur in any of the selected training documents are effectively used as vocabulary. Terms that do not occur in the training set are considered unknown to the learner since we construct the vocabulary on the basis of the training data. So, vocabulary size increases with the number of training documents as shown in Figure 4.1. Note that, in some cases, performance could be improved by further reducing the size of the vocabulary as described in Section 3.2.3. As experiments show, however, this has no crucial effect on the availability problem. That is to say, we need a sufficiently large number of index terms to adequately represent text in such a way that classes can be discriminated. And with the learning algorithms applied here, the performance degradation caused by using too many features—an effect that is often referred to as the curse of dimensionality (see p. 32)—is smaller than that due to using too small a vocabulary.[7] Also, note that applying class-specific feature selection techniques when only very few training examples are available may turn out not to be statistically reliable.

Experimental Results

Figure 4.2 shows the classification accuracies of the four supervised text learning algorithms on the three text corpora selected when the number of labeled training documents is varied.[8] The horizontal axes indicate the number of labeled training documents on a log scale. Note, for instance, that a total of 20 training documents for the 20 Newsgroups

[7]See Lanquillon (2000a) for experiments showing this effect.

[8]See Section 2.3 (pp. 13 ff.) for the definitions of common performance measures such as *accuracy*.

dataset corresponds to one document per class and, for the WebKB dataset, a total of four training documents corresponds to one document per class. The vertical axes indicate the average classification accuracies on the test sets. Note the different magnifications of the vertical scales.

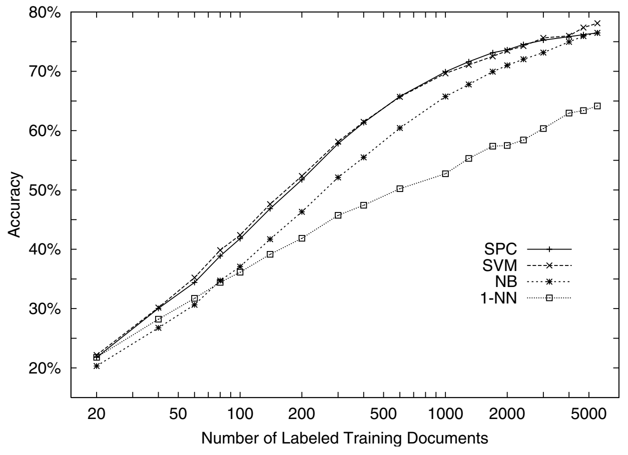
On the 20 Newsgroups dataset, the single-prototype classifier (SPC) and the linear support vector machine (SVM) achieve the best results. In the depicted range, their learning curves are almost identical. Only towards the right side of the curve with many training examples available, is the support vector machine superior to the single-prototype classifier. While the naïve Bayes classifier (NB) yields comparable results only when the number of labeled training examples is large, the simple one-nearest-neighbor rule (1-NN) rule is clearly outperformed by the three other algorithms. On the WebKB dataset, the support vector machine yields the best results. This time, the single-prototype classifier performs, on average, a few points worse than the support vector machine, whereas the naïve Bayes classifier is outperformed on average by almost 10 points by the top-performing support vector machine. The one-nearest-neighbor rule performs substantially worse than any of the three other approaches. On the TREC subset, the nearest-neighbor rule, again, performs much worse than the other three approaches, which achieve very accurate results.

Notably, only the support vector machine is consistently among the best performing approaches. This finding is in accordance with results described in other studies on text classification tasks.[9] Furthermore, the nearest-neighbor rule performs substantially worse than the three other approaches. A possible explanation for this finding is the inability of nearest-neighbor rules to exploit co-occurrence patterns across documents and, thus, to generalize beyond the training data provided. Moreover, the available number of training examples may not be sufficient to cover the feature space in such a way that documents of the correct class can be found near new documents. Note that the general K-nearest-neighbor rule with $K > 1$ does not yield better results than the simple nearest-neighbor rule in these settings.

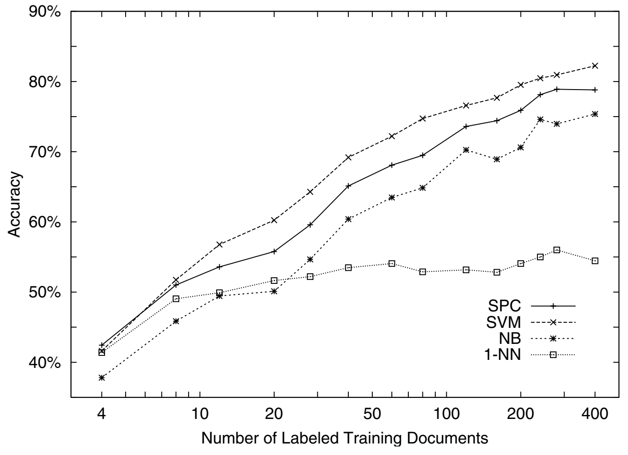
Notice that the accuracies achieved vary greatly across different datasets and different amounts of labeled data. A reason for this lies not only in the separability of classes, but also in the number of classes per se. For a particular dataset, however, the learning curves of all learners but the one-nearest-neighbor rule are quite similar. Even though at different levels, all learning curves illustrate that learning reasonably accurate classifiers requires many labeled training documents. Generally, the more labeled data there is, the better the performance achieved. Labeled examples have an exponential value in reducing the probability of error.[10] In particular, additional examples yield substantial improvements in classification accuracy when training data is scarce. In contrast, only marginal improvements are gained through the provision of additional training examples when many training examples are available already. For the text corpora examined, the learning curves begin to converge to a certain dataset-specific level when the training sets contain some hundred examples per class. Although this critical training set size is much smaller than theoretically suggested, providing the labeled training data is still a tedious and time-consuming task.

[9] See Joachims (1997b), and Dumais *et al.* (1998), for example.

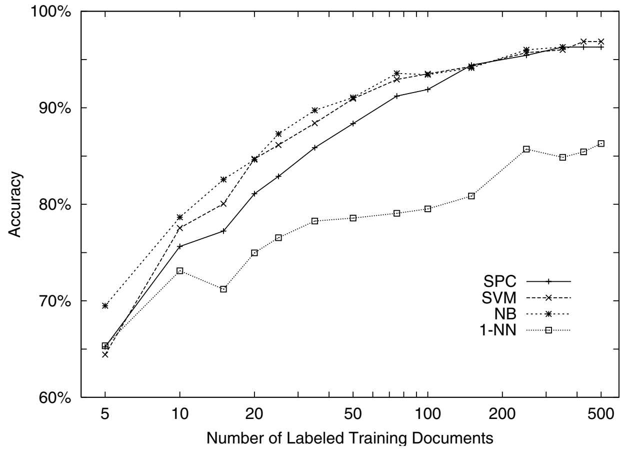
[10] See Castelli and Cover (1995), p. 110.



(a) 20 Newsgroups dataset



(b) WebKB dataset



(c) TREC dataset

Figure 4.2: The effect of the training set size on the classification accuracy of the single-prototype classifier (SPC), a linear support vector machine (SVM), the multinomial naïve Bayes classifier (NB), and the simple one-nearest-neighbor rule (1-NN) on three common text corpora. The horizontal axes indicate the number of labeled training documents on a log scale. The vertical axes indicate the average classification accuracies on the test sets. Note the different magnifications of the vertical scales.

4.1.3 Argument for the Use of Unlabeled Documents

As the discussion above has shown, if not intractable, it is at least expensive to provide a sufficient amount of labeled training data. Hence, a key issue is the question of how to reduce the need for labeled data.

This question can be rephrased as follows. What other sources of information can we exploit in order to partly substitute the need for labeled training data? We consider the following two types of information:

- *Unlabeled data.* In many text learning tasks, documents can be extracted from online sources at very low cost. So, while providing class labels for converting plain examples into labeled training examples is expensive, collecting the data is typically cheap. Hence, unlabeled documents can be considered to be readily available in large quantities. Note that we assume that both labeled and unlabeled data come from the same source, so that the only difference between the two sources is the class labels assigned.

- *Background knowledge.* Aside from information in form of examples, we consider background knowledge to be any type of useful information about the problem that is to be solved.[11] For instance, this might be knowledge about the class structure such as relations among classes in the form of a class-hierarchy,[12] semantic knowledge about words and their co-occurrence patterns, or, in general, some domain theory. Typically, background knowledge is domain or application-specific. As such, it usually requires human effort to be engineered and provided and may, therefore, be considered expensive, just like labeled data.

Although helpful in reducing the need for labeled data, relying on background knowledge often merely shifts the problem from providing class labels to providing other types of manually engineered information about a particular problem. Since reducing human effort is the true goal in minimizing the need for labeled data, exploiting background knowledge is often not a remedy. Also, utilizing background knowledge makes any approach domain- or application-dependent. It would thus be difficult, if not impossible, to apply any approach which depends on background knowledge to other text learning tasks. In the following, we therefore focus on using unlabeled data to partly substitute labeled data so as to reduce the need for labeled data.

A natural way to make use of unlabeled data is through unsupervised learning. A typical unsupervised learning task is to automatically discover groups of examples in data which are similar in some way, when there is no hint about actually existing classes. This process is known as clustering.[13] Note that the previously discussed classification task amounts to describing a finite set of predefined classes so as to be able to classify new examples

[11]Nevertheless, unlabeled data might also be referred to as background knowledge as in Emde (1994). We refrain from doing so because we do not consider raw data as knowledge.

[12]See Section 260 (pp. 75 f.).

[13]For example, see Jain and Dubes (1988) for a description of common clustering algorithms.

Document	Type	t_1	t_2	t_3	t_4	Class
$\mathbf{d}_1$	labeled	$\times$				rel
$\mathbf{d}_2$	labeled				$\times$	non
$\mathbf{d}_3$	unlabeled	$\times$	$\times$			**?**
$\mathbf{d}_4$	unlabeled			$\times$	$\times$	**?**
$\mathbf{d}_5$	new		$\times$			**?**

Table 4.1: An example of a text classification problem for which exploiting co-occurrence patterns is helpful. The rows correspond to documents; crosses mark the occurrence of a particular term t_i in document $\mathbf{d}_j$. Without unlabeled documents, there is no basis in terms of term occurrences in the labeled documents $\mathbf{d}_1$ and $\mathbf{d}_2$ for assigning either one of the two relevance class labels rel and non to the new document $\mathbf{d}_5$. In the presence of unlabeled documents, however, the co-occurrence of terms t_1 and t_2 in document $\mathbf{d}_3$ provides some evidence that $\mathbf{d}_5$ should be considered relevant even though we do not know the class label of document $\mathbf{d}_3$.

into the predefined classes, whereas clustering aims at identifying a finite set of classes in the data by itself without the ability to classify new examples. Although other learning tasks can be defined, in the following we will use classifier design synonymously with supervised learning and clustering synonymously with unsupervised learning.[14]

Note that learning from unlabeled data *alone* is generally insufficient if class assignments that are better than random are to be yielded.[15] In particular, clustering may be able to discover the classes desired and, thus, to construct meaningful decision regions. However, associating these regions with the correct class labels cannot be done without any information about the classes such as labeled data.[16] Castelli and Cover (1996) determine the relative value of labeled versus unlabeled examples in a Bayesian analysis. They conclude that labeled examples are exponentially more valuable than unlabeled examples. This is a very pessimistic view of the value of unlabeled data. Although labeled data is, admittedly, more valuable than unlabeled data for the classification task because of the additional class information provided, the fact that decision regions can be recovered given a sufficient amount of unlabeled data shows that unlabeled data carry valuable information. And exploiting this should be considered as a means to reduce the need for labeled data.[17] Early empirical results in text learning tasks show that approaches exploiting unlabeled documents are particularly promising.[18]

How can using unlabeled data help when learning a text classifier? It is well known in information retrieval research that words in natural language do not occur independently but rather in strong co-occurrence patterns.[19] While some words are likely to co-occur in certain documents, others are not. In fact, the differences in class-specific co-occurrence patterns, i.e. the class-specific word distributions, help to discriminate among classes. Consequently, exploiting co-occurrence information can enhance classification accuracy. Yet, to exploit co-occurrence patterns to a large extent, it is necessary that documents be represented by a sufficiently large number of terms. Unlike in standard classification tasks

[14]Regression learning, for instance, is another supervised task, whereas association rule generation is another unsupervised task.

[15]See Castelli and Cover (1995).

[16]In fact, clustering does not even mean to relate its clusters with predefined class labels.

[17]Also see O'Neill (1978) for results that support this in the context of linear discriminant analysis.

[18]For example, see Nigam *et al.* (1998), and Joachims (1999b).

[19]See van Rijsbergen (1977), for example.

with structured data, however, the feature set used to describe text is not very well defined and is not limited but by combinatoric constraints. So, because we commonly construct the vocabulary based on the training data, the number of potential features increases with the size of the training set as previously shown in Figure 4.1. When labeled training data is scarce, the resulting vocabulary might be too small to adequately represent text. Thus, additional unlabeled training documents can provide information about other possible features, in general, as well as information about their co-occurrence patterns with features that occur in the labeled documents, in particular. This is illustrated in the small example given in Table 4.1. Correctly classifying the new document is solely supported by means of term co-occurrence information from the unlabeled documents.

4.1.4 Methodologies for Learning from Labeled and Unlabeled Data

Above, we motivated the use of unlabeled data in addition to labeled data. At this point, the principal question is how to combine both sources of information. Note that any methodology which integrates learning from labeled and unlabeled data could generally be described as partially or semi-supervised since the supervision in terms of the given class labels is available only for a certain subset of the data. Nevertheless, we will refer to only those approaches which combine supervised and unsupervised techniques as semi-supervised. As we will see below, the mere fact of combining labeled and unlabeled data does not require that use be made of elements from both types of learning directions.

The following description is split into two parts. First, and primarily, we focus on general methodologies for integrating labeled and unlabeled data. Typically, these methodologies can be used in combination with any supervised learning algorithm. Subsequently, we briefly review some approaches that call for specific learning algorithms.

General Methodologies

Figure 4.3 gives an overview of some common methodologies that are generally applicable to any supervised learning algorithm. Admittedly, this taxonomy is not complete, nor is it exclusive. In many situations, the dividing line between methods or technologies is fuzzy at best. In particular, it is possible to define hybrid approaches along the tree by combining typical elements of different branches.[20]

In order to benefit from unlabeled data in a supervised setting, a learner must augment unlabeled examples by class labels in some way. The key distinguishing feature for combining labeled and unlabeled is whether or not an approach is allowed to query the class labels of selected unlabeled examples. This results in a distinction between *pool-based active learning* and *bootstrapping classifiers* as follows.

- *Pool-based active learning.* Most supervised learning algorithms are *passive* in that they receive as input a set of labeled training examples from which they have to induce a classifier. In contrast, *active learning* is the study of how to use the ability to

[20] See Nigam and Ghani (2000) for some examples.

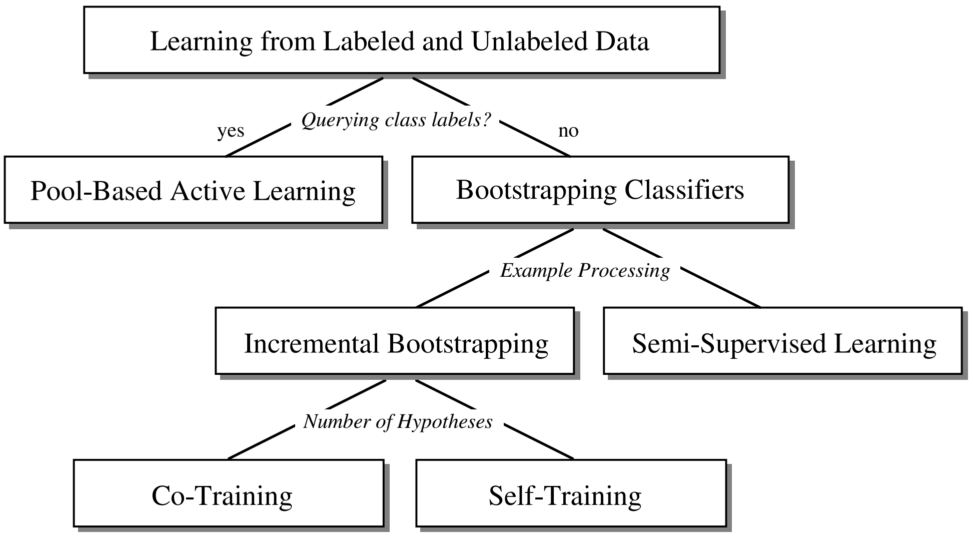

Learning from Labeled and Unlabeled Data
Querying class labels?
yes
no
Pool-Based Active Learning
Bootstrapping Classifiers
Example Processing
Incremental Bootstrapping
Semi-Supervised Learning
Number of Hypotheses
Co-Training
Self-Training

Figure 4.3: A taxonomy of approaches that aim to learn from both labeled and unlabeled data. The key distinguishing feature is whether an approach is allowed to query true class labels of selected examples from the given pool of unlabeled examples. Approaches that do so and subsequently learn a new classifier based on the augmented training set are commonly known as *active learning*. In contrast, approaches that do not receive feedback from the user are often referred to as *bootstrapping* approaches. While *co-training* and *self-training* methods incrementally enlarge the training set by adding unlabeled examples augmented by class labels that were predicted by any of the underlying supervised learning algorithms, *semi-supervised learning* is based on iteratively clustering the unlabeled data and then using information based on the cluster partition to enhance the training data for a supervised learner.

interact with the environment defining the learning task.[21] Particularly in the supervised learning setting is an active learner given the opportunity to query the true class labels of selected data. Thus, the promise of active learning is that the need for labeled training examples can be significantly reduced when the learner itself is responsible for selecting its training data. Typically, an active learner aims to select exactly those examples which would maximally improve classification effectiveness if the true class labels were known. A well-known approach to heuristically determining these examples is the *query-by-committee* concept, which examines the disagreement among the class labels assigned by an ensemble of classifiers.[22]

Generally, the active learning setting allows the learner either to synthetically create query examples or to select query examples from an incoming data stream. These approaches are inappropriate in text domains because the first approach might generate awkward documents, whereas the latter one would only inefficiently model the data distribution.[23] In *pool-based* active learning, however, the learner has access to a pool of unlabeled data in addition to the labeled training set and can iteratively request the true class labels for some selected examples in this pool.[24] This approach naturally fits in the text learning setting and, thus, provides a straightforward way to combine sets of labeled and unlabeled data.

[21]See Cohn *et al.* (1996), p. 129.

[22]See Seung *et al.* (1992), and Freund *et al.* (1997).

[23]See McCallum and Nigam (1998b).

[24]For example, see Lewis and Gale (1994), Lewis and Catlett (1994), McCallum and Nigam (1998b), and Tong and Koller (2000).

Although put on the same level as the general bootstrapping framework in Figure 4.3, active learning components may actually be added to each of the bootstrapping variants to be described below.[25] In fact, we must assume that an approach that can actively query class labels is superior to an approach without this ability. The reason for this is simple: the reduction of uncertainty in the information about the class labels. Numerous empirical studies show that active learning substantially reduces the need for labeled examples.[26] However, this advantage always comes at the expense of additional user effort. Therefore, we focus on plain bootstrapping approaches since they do not require any user interaction. Note, though, that we should consider adding an active learning component to otherwise passive approaches whenever the user is willing to interactively label data.

- *Bootstrapping classifiers.* In the supervised learning setting, bootstrapping is a general framework for combining labeled and unlabeled data.[27] Bootstrapping a classifier basically amounts to iteratively generating additional training data through the classifier's own effort and learning a refined classifier based on the augmented training set. More precisely, the set of labeled examples is generally used as seed information to learn an initial classifier.[28] Then, a bootstrapping iteration consists of two steps. First, the current classifier is used to estimate the class labels for the unlabeled data. On the basis of these labels, additional training examples are generated. Then, a refined classifier is learned from the augmented training set. Typically, this iteration stops either when a given convergence criterion is fulfilled or after a certain number of steps have been carried out to prevent endless oscillations.

The general bootstrapping framework leaves many essential design choices open. In current research, two key choices are the number of hypotheses, or classifiers, used in the framework and the way additional training examples are incorporated, yielding the following distinction among approaches as depicted in Figure 4.3:

- *Incremental bootstrapping.* The incremental feature of these approaches refers to the way the unlabeled examples are incorporated as new training examples into the training set. Generally, only one or very few unlabeled examples that were labeled with high confidence are added to the training set at each bootstrapping iteration. The decision to convert an unlabeled example into a labeled training example is always considered final. Typically, the bootstrapping process terminates when all unlabeled examples have been converted into labeled training examples. Below we briefly describe two incremental bootstrapping approaches that differ in the number of underlying hypotheses learned from the current training data, namely *co-training* and *self-training*.

[25]For example, see McCallum and Nigam (1998b) for such a hybrid approach.

[26]See McCallum and Nigam (1998b), Schohn and Cohn (2000), and Tong and Koller (2000) for examples of active learning approaches to text classification.

[27]For example, see Yarowsky (1995), Blum and Mitchell (1998), and Jones *et al.* (1999). In statistical terms, however, bootstrapping typically refers to a resampling method, see Section 256 (p. 73).

[28]It is also possible to use other types of seed information depending on which initial classifiers are generated, McCallum and Nigam (1999), for example, use class-specific keywords.

Blum and Mitchell (1998) originally proposed a co-training algorithm. Their co-training setting assumes that there are two redundantly sufficient sets of features for representing examples. Such redundant representation naturally occurs, for instance, when classifying web pages. For this task, page contents, on the one hand, and anchor texts from links pointing to the particular page, on the other hand, might define two representations. The inherent redundancy in the features allows learning of two distinct classifiers and, then, them to be used to train each other over unlabeled data. Recent co-training approaches explore settings with only one representation. In order to benefit from the co-training setting, they either artificially split the feature set[29] or they apply different learners while using the same features.[30] Thus, the key ingredient of co-training strategies is essentially to exploit variability in predicting class labels due to differences in example representation or inductive bias.[31]

A self-training approach is probably the most straightforward bootstrapping approach to combining labeled and unlabeled data. Having learned an initial classifier from the labeled data, this approach iteratively adds self-labeled examples to the current training set and refines the current classifier.[32]

- *Semi-supervised learning.* The objective of semi-supervised learning is to benefit from both supervised and unsupervised learning when combining labeled and unlabeled data. The promise in the approach lies in the fact that we can utilize the strengths of either approach while getting rid of common drawbacks. Specifically, supervised learning can produce accurate classifiers but has an enormous need for labeled training examples, whereas the ability to exploit cheap unlabeled data through unsupervised learning methods is accompanied by a difficulty in choosing a suitable number of clusters and generating an appropriate starting solution for the clustering task plus the inability to associate clusters with classes in the absence of labeled data.[33]

  With unsupervised learning, we focus on partitional clustering approaches. In this setting, the fundamental difference to incremental bootstrapping is that semi-supervised learning algorithms generally use all unlabeled examples when learning a refined classifier, while allowing the class label of each unlabeled example to change after each iteration. Consequently, decisions about the class labels of unlabeled examples are never considered final. Partitional clustering methods generate a single partition of the data in an attempt to recover natural groups present in the data. With the guidance of labeled data, the hope is that these groups will better match the underlying class structure. See Section 4.3 for a thorough discussion.

When discussing learning from labeled and unlabeled data, the focus of this work is on semi-supervised learning. Yet, depending on the underlying learning algorithms used, the

[29] See Nigam and Ghani (2000).
[30] See Goldman and Zhou (2000).
[31] These aspects also motivated learning ensembles of classifiers as discussed in Section 3.3.3 (pp. 72 ff.).
[32] See Nigam and Ghani (2000), for example.
[33] See Bensaid *et al.* (1996), for example.

boundary between semi-supervised learning and self-training is not always clear. In fact, both approaches can be considered as opposite ends of a whole spectrum of choices of how to relabel and incorporate unlabeled data. As already noted above, hybrid approaches can often be found.

Approaches for Specific Learning Algorithms

So far, we have not considered approaches that address particular learning algorithms only. For example, an interesting and very promising direction of research is to learn similarity functions, which can be employed instead of the commonly used measures described in Section 3.3. Hofmann (2000) develops a general method for learning the similarity between documents in terms of a general dot product based on information-geometric principles. The derivation of this dot product, which is known as the Fisher kernel, is based on a latent class decomposition of the document by term matrix and is therefore similar to latent semantic indexing as described in Section 134. Since this approach does not require the true class label of the documents, it can be used prior to both unsupervised and supervised learning. In particular, provision of this advanced similarity measure can be regarded as a means of incorporating unlabeled data into any supervised learning technique that can make use of generalized dot products such as similarity-based classifiers, like the Rocchio variants, and support vector machines.

Bennett and Demiriz (1998) describe work on semi-supervised support vector machines. This approach, as well as the transductive support vector machine applied to text classification by Joachims (1999b), uses the unlabeled test data in addition to the labeled training data to better adjust the parameters of the support vector machine. Although designed for classifying the examples of just this test set, the resulting support vector machine might also be applied to classify new, unseen examples as assumed in this work. However, there is as yet no empirical evidence of how well this works. In a probability analysis on the value of unlabeled data for classification problems, Zhang and Oles (2000), in fact, show that support vector machines in their current forms are not suitable if benefit is to be drawn from additional unlabeled data in a semi-supervised bootstrapping framework.

Zelikovitz and Hirsh (2000) describe an approach for combining labeled and unlabeled data that is different from what we have discussed so far. They consider the problem of classifying short text strings. An unlabeled document source of larger documents that does not come from the same source but reflects common background knowledge on the underlying domain is used to improve classification performance. The idea is to use co-occurrence information in the unlabeled documents in an intermediate step to assess document similarity. Rather than evaluating the similarity between any two short text strings directly, it is assessed on the basis of the individual similarities of the two strings to any one unlabeled document. Consequently, this approach does not attempt to assign labels to the unlabeled data in order to utilize the unlabeled data. Finally, note that the classification of short text strings is distinct from the classification of documents. The difficulty in classifying short text strings is that there are only a small number of matching terms across different strings. In this context, the use of each unlabeled example as background knowledge can be seen as a kind of implicit query expansion.

4.2 Related Work

This chapter deals with learning from both labeled and unlabeled data in a semi-supervised fashion that aims at combining elements from supervised and unsupervised learning. In the previous section, semi-supervised learning has been put in context with other state-of-the-art methodologies for learning from labeled and unlabeled data such as pool-based active learning and co-training. These related methodologies will not be covered again here.

The concept of semi-supervised learning is neither unambiguous nor new. Below, we first shed some light on approaches that are legitimately referred to as semi-supervised but which are distinct from the notion of partial supervision employed in this work. Second, we focus on approaches which follow the same basic idea, namely combing supervised and unsupervised learning for classification tasks.

Contrast to Approaches Named Alike

Supervised learning (classification) and unsupervised learning (clustering) can be referred to precisely as approaches that learn in the presence or absence, respectively, of a target variable (class label). Hence, supervised and unsupervised learning can be considered as the two ends of a whole continuum of learning approaches, in which information about the class labels is only partly available. These approaches can all be described as partially or semi-supervised.

Recall that our understanding of semi-supervised learning is that the class information is only given for a subset of the training data, so that a labeled and an unlabeled set of examples are available for training. Furthermore, we require from a semi-supervised learner that it combine elements from both supervised and unsupervised learning. In this sense, pool-based active learning or co-training, though combining labeled and unlabeled data, are not referred to as semi-supervised because they do not employ any unsupervised learning strategy.

Jeon and Landgrebe (1999), for instance, propose a partially supervised classification method for coping with learning tasks where labels for only one class of interest are given. Assuming a sufficient number of positive training examples, negative examples must be determined from an unlabeled dataset. Note that, in our work, we assume that there is an insufficient number of labeled examples for any class and an unlabeled dataset can be used to obtain additional training examples.

In their work on semi-supervised clustering, Pedrycz (1984), Bensaid *et al.* (1996), and Bensaid and Bezdek (1998) employ the same definition of partially labeled data as we do: that the training data fall into a labeled and an unlabeled subset. However, their goal is to improve clustering rather than classifier design. We will have more to say on this in the following section and in the following chapter.

Building on the framework of *probably approximately correct* learnability developed in computational learning theory,[34] Board and Pitt (1989) propose a theoretical framework of

[34]See Valiant (1984).

semi-supervised learnability. They consider the situation in which a collection of disjoint concepts, i.e. classes, is to be simultaneously learned with only partial information about class memberships available to the learning algorithm. Rather than assuming that class labels are given, Board and Pitt assume in their notion of partial supervision that for any two examples the learner only knows whether or not these examples belong to the same class. Obviously, this is distinct from the notion of partial supervision employed in this work, namely that the class labels are given for a subset of the training data only.

Overview of Similar Approaches

Admittedly, the idea of learning from both labeled and unlabeled data is not new. Yet, only recently is it gaining in popularity in the field of text classification, where it appears to be particularly beneficial. A simple reason for this interest is that the performance achieved by supervised methods has reached a certain plateau and that, rather than by marginally enhancing effectiveness, progress is made with respect to efficiency by, for instance, reducing the enormous need for labeled data.

We identify two key perspectives on combining supervised and unsupervised learning. First, learning from labeled and unlabeled data is viewed as a type of problem with missing or incomplete data, where the class labels of the unlabeled data are considered the missing items. Rather than discarding examples with missing values, the goal is to impute class labels and proceed in a supervised fashion as if all data were labeled. Many sophisticated imputation approaches solve the missing-value problem by using the Expectation-Maximization (EM) algorithm.[35] The family of EM algorithms and their application to classification is broadly studied in the statistics literature. Among the published responses to the original EM paper by Dempster *et al.* (1977), Little (1977) mentions the idea of using an EM-based approach to improve a classifier by treating the class labels of unlabeled data as missing values. This idea was meant to extend work on using unlabeled data in discriminant analysis by McLachlan (1975). And McLachlan and Ganesalingam (1982) propose an approach for the same problem using the EM algorithm.

Typically, EM-based approaches to learning from labeled and unlabeled approaches are based on probabilistic mixture modeling.[36] More recent approaches on combining labeled and unlabeled data via the EM approach and their application to real-world problems can be found in Shahshahani and Landgrebe (1992) and (1994) and Miller and Uyar (1997).

Similar to these approaches—but with respect to text classification based on a multinomial naïve Bayes classifier—is the work by Nigam *et al.* (1998) and (2000). In further work, McCallum and Nigam (1999) use keywords instead of labeled examples as seed information to initialize their bootstrapping approach.[37] The work described in this dissertation is, in fact, a generalization of Nigam's seminal work on combining labeled and unlabeled data via the EM algorithm. Rather than depending on the naïve Bayes classifier, however, our approach permits use of any type of supervised learning algorithm.

[35]See Ghahramani and Jordan (1994), for example.

[36]See McLachlan and Basford (1988), p. 28, for further references.

[37]Also see Jones *et al.* (1999).

In the other perspective, combining supervised and unsupervised learning so as to solve classification tasks is regarded straightforwardly as a two-step process.[38] In a first step, any unsupervised learning algorithm can be applied to uncover groups in the data. As we have seen above, this does not yet allow classification of new examples. In a second step, the clusters discovered are associated with class labels by means of a supervised learning technique. On the basis of these relations between clusters and classes, it is possible to classify new data. Although the practical significance of this type of learning problem to reduce the need for expensive labeled example was already recognized by Lippmann (1989), only little work has been done on this particular problem since then. For instance, some approaches that are similar to this scheme are described by Greenspan *et al.* (1991), Pao and Sobajic (1992), and Emde (1994).

Generally, these approaches differ greatly in the degree that information inherent to the labeled data is incorporated into the unsupervised learning step. For instance, determining a suitable number of clusters or generating an appropriate starting solution might be supported by using labeled data.[39] Depending on the degree of supervision in terms of using the labeled data during the unsupervised learning step, both perspectives on combining labeled and unlabeled data described above can, in fact, be very similar. In particular, note that the EM algorithm can be regarded as probabilistic clustering algorithm, akin to fuzzy c-means clustering.

4.3 The Semi-Supervised Learning Framework

This section describes a family of semi-supervised learning algorithms for integrating labeled and unlabeled documents, extending the approach proposed by Nigam *et al.* (1998) and (2000). The promise of semi-supervised learning is that it will take advantage of the strengths of both supervised and unsupervised techniques, while avoiding common drawbacks. Specifically, supervised learning can learn accurate classifiers, given a large amount of labeled data.[40] Yet, where provision of labeled data is expensive, unlabeled data is abundant and cheap. However, learning from unlabeled data alone cannot produce a classifier since class information is missing. Furthermore, determining a suitable number of clusters and generating an appropriate starting solution for the clustering process together create a challenge in unsupervised learning. To sum up, combining a small amount of labeled data and a large set of unlabeled data in a semi-supervised fashion can yield an accurate classifier at greatly reduced cost.

4.3.1 Integrating Supervised and Unsupervised Learning

Extending Definition 3.1.2 (p. 24) for the supervised text learning task, the problem of learning a classifier from labeled and unlabeled data can be described as follows. Given

[38]See Lippmann (1989), p. 48, for example.

[39]See Bensaid *et al.* (1996), for example.

[40]If an accurate classifier cannot be learned, i.e. classes cannot be separated effectively, adding unlabeled data would not help. So, we assume that an accurate classifier can be learned, given enough labeled data.

a set of training documents D, for some subset of the documents $\mathbf{d}_j \in D^l$, we know the class label $y_j = T(\mathbf{d}_j)$ and, for the rest of the documents $\mathbf{d}_j \in D^u$, the class labels are unknown. Thus, we have a disjoint partitioning of the training documents into a labeled set and an unlabeled set of documents, i.e. $D = D^l \cup D^u$. Let $n_l = |D^l|$ and $n_u = |D^u|$ denote the number of labeled and unlabeled documents, respectively. The task is to build a classifier based on the training documents D for predicting the class label of unseen unlabeled documents.

The concepts to be presented in this section are general enough so as to be applicable for any type of data. So, instead of employing the terminology for text classification introduced in the previous chapter, we use a terminology that is less specific. In particular, we refer to datasets consisting of data points or patterns rather than documents. Hence, let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ denote a set of n patterns $\mathbf{x}_j \in \mathbb{R}^m$. This dataset is split into a labeled and an unlabeled subset X^l and X^u, respectively, such that $X = X^l \cup X^u$. Again, let $n_l = |X^l|$ and $n_u = |X^u|$ denote the number of labeled and unlabeled patterns. For each labeled pattern $\mathbf{x}_j \in X^l$ the true class label y_j is given. In Section 4.3.2 we shall return to the terminology peculiar to text classification tasks.

We proceed in four steps to introduce our general semi-supervised learning framework. First, we briefly describe the concept of unsupervised partitional clustering in its most basic form and identify some of its major drawbacks. Then, we turn to semi-supervised clustering with the objective of guiding clustering by labeled data. In the third step, we use the labeled data to associate the identified clusters with the predefined class labels of the supervised learning task so as to allow the classification of new patterns. Finally, we generalize this approach with respect to finding appropriate cluster representations to obtain our semi-supervised learning framework.

Unsupervised Partitional Clustering

The most straightforward way to make use of unlabeled data is through unsupervised learning. This is typically known as clustering. In the following, we outline the essentials of common partitional clustering algorithms. The description is, of course, not complete; many variants and extensions can be found in the literature.[41] Algorithm 4.1 shows a common scheme of a partitional clustering algorithm in pseudo code.

Input to the clustering algorithm is a dataset X of patterns.[42] There is no labeled data, i.e. $X^l = \emptyset$, so that $X = X^u$ and $n = n_u$. Determining the number c of clusters to be identified in X is a crucial issue. Here, choosing c is listed as the first step of the algorithm. Alternatively, it might also be given as input to the algorithm. The goal of partitional clustering is to find a partition of X into c non-empty clusters in such way that patterns within the same cluster are as similar as possible and patterns in different clusters are as dissimilar as possible. This is often referred to as the within-cluster homogeneity and between-cluster heterogeneity. Typically, the clustering algorithm outputs the set of representations for the c clusters, $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_c\}$, and the cluster membership matrix

[41] For example, see Jain and Dubes (1988) for a description of common clustering algorithms.

[42] Albeit essential, we do not consider any preprocessing issues such as appropriate scaling at this point.

Algorithm 4.1 Partitional Clustering

```
Input: dataset $X$
1: choose number of clusters $c$, $c \in \{2, \ldots, n-1\}$
2: initialize cluster memberships $\mathbf{U}$ randomly
3: compute initial cluster representations $V$
4: repeat
5:     re-compute cluster memberships $\mathbf{U}$
6:     re-compute cluster representations $V$
7: until cluster memberships $\mathbf{U}$ do not change significantly
Output: cluster representations $V$ and class memberships $\mathbf{U}$
```

$\mathbf{U}$ which describes how the unlabeled data is partitioned among the c clusters. Note, however, that often either the partition or the representations are desired as output only.

The notation of cluster memberships is vital for any clustering algorithm. Typically, membership degrees of patterns in clusters are represented by a $(c \times n)$ matrix $\mathbf{U} = (u_{ij})$, where $u_{ij} \in [0, 1]$ reflects the membership degree of pattern $\mathbf{x}_j \in X$ in cluster i. As a rule, the constraint $\sum_{j=1}^{n} u_{ij} > 0$, for $i = 1, \ldots, c$, ensures that the clusters are non-empty. According to the terminology for classification tasks, membership degrees are sometimes also referred to as labels. In general, two types of labels can be distinguished:[43]

- *Hard* or *crisp* labels describe the most traditional type of membership degree. A pattern $\mathbf{x}_j$ either does ($u_{ij} = 1$) or does not ($u_{ij} = 0$) belong to a certain cluster i, so that $u_{ij} \in \{0, 1\}$. Typically, the constraint $\sum_{i=1}^{c} u_{ij} = 1$, $j = 1, \ldots, n$, ensures that any pattern $\mathbf{x}_j$ belongs to exactly one cluster.

- *Soft* or *fuzzy* labels, in contrast, permit graded memberships. This is often more appropriate when a pattern cannot be clearly assigned to one of the clusters. In soft labeling, the membership degrees are commonly within the unit interval, i.e. $u_{ij} \in [0, 1]$. Depending on the constraints put on the membership degrees for a particular pattern, we can further distinguish *possibilistic* or *unconstrained fuzzy* and *probabilistic* or *constrained fuzzy* labels.[44] Possibilistic labeling allows any combination of membership degrees except for the degenerate case where all values are zero for a particular pattern, making it the least restrictive form of labeling. Probabilistic labeling requires that the membership degrees for any particular pattern $\mathbf{x}_j$ be normalized in such a way that they sum to unity.

Another fundamental notation is that of cluster representation. Commonly, the representation $\mathbf{v}_i$ of cluster i is evaluated as the centroid, or mean, of the patterns assigned to that particular cluster. These cluster representations are often referred to as *cluster prototypes*. Obviously, their computation depends on the membership degrees, yielding

$$\mathbf{v}_i = \frac{\sum_{j=1}^{n} (u_{ij})^p \, \mathbf{x}_j}{\sum_{j=1}^{n} (u_{ij})^p} \qquad \forall i \in \{1, \ldots, c\} \tag{4.1}$$

[43]See Bensaid and Bezdek (1998), for example.

[44]Often, constrained fuzzy labels are simply referred to as fuzzy labels.

where the parameter $p \in [1, \ldots, \infty)$ controls the fuzziness of the clusters if applicable.[45] Note that, for hard labeling, i.e. $u_{ij} \in \{0, 1\}$, the parameter p can be ignored. Many extensions to this simple cluster representation can be considered.[46] Later in this section, we discuss an alternative approach to representing clusters which will lead to the general semi-supervised learning framework.

Given the fundamental notation of the cluster representations V and memberships $\mathbf{U}$, the partitional clustering task can be formulated as minimizing the objective function[47]

$$J_p(\mathbf{U}, V; X) = \sum_{i=1}^{c} \sum_{j=1}^{n} (u_{ij})^p \, \|\mathbf{x}_j - \mathbf{v}_i\|^2 \tag{4.2}$$

subject to constraints on $\mathbf{U}$ according to the labeling type employed. Again, the parameter $p \in [1, \infty)$ controls the fuzziness of the clusters, where applicable. Any norm can be used to evaluate the distance between pattern $\mathbf{x}_j$ and cluster representation $\mathbf{v}_i$. Frequently, Euclidean distance is employed, i.e. $\|\mathbf{x}_j - \mathbf{v}_i\|^2 = (\mathbf{x}_j - \mathbf{v}_i)^T(\mathbf{x}_j - \mathbf{v}_i)$.

Note, for example, that setting $p = 1$, enforcing hard labels, and using Euclidean distance amount to the well-known hard c-means clustering algorithm.[48] An arbitrary norm and setting of p in combination with constrained fuzzy labeling leads to the family of fuzzy c-means clustering algorithms.[49]

A solution that minimizes function J_p can be approximated by alternating optimization. As shown in Algorithm 4.1 (lines 4 to 7), this amounts to iteratively updating the membership matrix $\mathbf{U}$ based on the current cluster representations V and updating V based on the current state of U. In particular, updating the cluster representations is carried out according to Equation (4.1). Updating the memberships depends strongly on the underlying labeling type. Using hard labels, for instance, typically requires that the following update scheme be used for each pattern $\mathbf{x}_j \in X^l$:[50]

$$u_{ij} = \begin{cases} 1 & \text{if } i = \arg\min_{i' \in \{1,\ldots,c\}} \|\mathbf{x}_j - \mathbf{v}_{i'}\| \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{1, \ldots, c\} \tag{4.3}$$

Ties are resolved arbitrarily. For constrained fuzzy labels, in contrast, the relative distances between patterns and cluster representations are taken into account.

Generally, the optimization process stops when the difference between two successive membership matrices is below a predefined threshold. Under certain conditions pertaining to the label types and the updating schemes, convergence—and thus termination—after a finite number of steps can be proven. Otherwise, a maximum number of steps could be set so as to prevent endless oscillations. As a rule, the solution obtained is only a local optimum but not a global optimum.[51] As such, it crucially depends on the starting

[45]Commonly this parameter is termed m. This, however, would clash with the number of features.

[46]For example, see Bensaid and Bezdek (1998), pp. 627–628, and the references therein.

[47]See Bezdek (1981), pp. 43–93, for a treatment of objective function clustering.

[48]See MacQueen (1967).

[49]See Bezdek (1981) for details on fuzzy c-means clustering.

[50]See Bezdek (1981), p. 55.

[51]Note that in some cases the solution might only be a saddle point; see Bezdek (1987).

solution. Without prior knowledge, a common choice is to initialize $\mathbf{U}$ randomly subject to the constraints imposed by the labeling type. The initial cluster representations V are then evaluated based on the random initialization of $\mathbf{U}$; see Algorithm 4.1 (lines 2 and 3).

As this brief description of a general partitional clustering algorithm shows, the solution is largely determined by the number of clusters chosen and by the starting solution, especially in high-dimensional feature space as is in the context of text classification. Of course, different choices of c and different random initializations of the membership matrix may be tried out. Given an appropriate function for assessing cluster validity, the best setting can then be selected. This, however, is a time-consuming process. Also, provision of appropriate measures for cluster validity is not trivial.[52] An alternative approach to ensuring more meaningful solutions is to exploit background knowledge such as labeled data to guide the clustering algorithm.

Guiding Unsupervised Learning by Labeled Data

To ensure meaningful solutions, labeled data can be used as background knowledge to choose a suitable number of clusters and to appropriately initialize the membership matrix. In particular, since we are ultimately interested in classification tasks, we desire the clustering algorithm to automatically identify the class structure of the underlying classification task. Hence, labeled data might be used to set the clustering algorithm on the right track towards a local minimum that is more likely to correspond to what we hope to find with respect to the classification task than a solution that is achieved by starting off from random initialization with a possibly inappropriate number of clusters. This kind of clustering can be described as semi-supervised;[53] see Algorithm 4.2 for a general outline.

To exploit the knowledge inherent to the labeled data, we assume that the classes to which the labels correspond are homogeneous, so that they may be described by the simple cluster representations introduced above. Under this assumption, it is reasonable to set the number of clusters to the number of classes, i.e. $c = k$. This eliminates the first major difficulty in partitional clustering tasks, namely to choose the number of clusters.

Let us turn to the problem of achieving solutions that are local minima. To increase the chances that the solution obtained will match our interest, it is reasonable to set the initial cluster representations to the centroids of the class-specific patterns. This is based on the assumption that the labeled data, though not fully representative for the classes because of their scarcity, roughly describe the classes and, therefore, resemble a *well-seeded* initial solution.[54] To initially ignore the unlabeled data, the initial cluster representations are computed solely from the labeled data.

For the sake of notational simplicity, we denote the labeled patterns by $\mathbf{x}_j^l \in X^l$ with corresponding labels $y_j^l \in \mathcal{C}$, for $j = 1, \ldots, n_l$, and the unlabeled patterns by $\mathbf{x}_j^u \in X^u$, for $j = 1, \ldots, n_u$. So, the entire dataset is denoted by $X = \{\mathbf{x}_1^l, \ldots, \mathbf{x}_{n_l}^l, \mathbf{x}_1^u, \ldots, \mathbf{x}_{n_u}^u\}$. Furthermore, we introduce separate membership matrices $\mathbf{U}^l$ and $\mathbf{U}^u$ for the labeled and

[52] See Bezdek (1981), pp. 95–154, for a treatment of cluster validity.

[53] See Pedrycz (1984), and Bensaid *et al.* (1996) for some similar semi-supervised clustering approaches.

[54] See Bensaid *et al.* (1996), p. 863.

Algorithm 4.2 Semi-Supervised Partitional Clustering

```
Input: dataset $X = X^l \cup X^u$ consisting of non-empty labeled and unlabeled subsets
1: set number of clusters $c$ to number of classes $k$
2: initialize cluster memberships $\mathbf{U}^l$ and $\mathbf{U}^u$
3: compute initial cluster representations $V$ based on labeled data only
4: repeat
5:     re-compute cluster memberships $\mathbf{U}^u$
6:     re-compute cluster representations $V$
7: until cluster memberships $\mathbf{U}^u$ do not change significantly
Output: cluster representations $V$ and cluster memberships $\mathbf{U}^u$
```

unlabeled documents, respectively. Specifically, $\mathbf{U}^l$ is a $(c \times n_l)$ matrix for the cluster membership degrees of the n_l labeled patterns $\mathbf{x}_j^l \in X^l$, and $\mathbf{U}^u$ is a $(c \times n_u)$ matrix for the cluster membership degrees of the n_u unlabeled patterns $\mathbf{x}_j^u \in X^u$. Note that, since there is no labeled data in the traditional clustering approach, the membership matrix $\mathbf{U}^u$ of the unlabeled data corresponds to the aforementioned matrix $\mathbf{U}$. Now, the complete membership matrix may be written as the $(c \times n)$ block matrix $\mathbf{U} = (\mathbf{U}^l|\mathbf{U}^u)$.

The memberships for the labeled data are fixed according to the class labels provided. With $c = k$ and $y_j^l \in \mathcal{C} = \{c_1, \ldots, c_k\}$ for $\mathbf{x}_j^l \in X^l$, we define

$$u_{ij}^l = \begin{cases} 1 & \text{if } y_j = c_i \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{1, \ldots, c\}, j \in \{1, \ldots, n_l\} \tag{4.4}$$

Note that this notation assumes that the class labels of the labeled data are hard. Although this is a reasonable assumption with respect to the supervised classification task, it is not necessary. Instead, any type of labeling might be employed, assuming that U^l be defined accordingly.

Since the memberships $\mathbf{U}^u$ do not affect the initial cluster representations, they should be initialized in such a way that the subsequently updated matrix differs significantly from the initialization. Typically, all entries are set to zero, i.e. $u_{ij}^u = 0$, while ignoring that, for the initial iteration, this violates the constraints imposed by any of the labeling types introduced above.

Now, according to Equation (4.1), the formula for the initial cluster representations based on the labeled data only is given by

$$\mathbf{v}_i = \frac{\sum_{j=1}^{n_l} (u_{ij}^l)^p \, \mathbf{x}_j^l}{\sum_{j=1}^{n_l} (u_{ij}^l)^p} \qquad \forall i \in \{1, \ldots, c\} \tag{4.5}$$

Algorithm 4.2 shows the pseudo code of a partitional clustering scheme which is guided by labeled data. The alternating optimization process (lines 4 to 7) carried out after the modifications pertaining to the initialization phase just described (lines 1 to 3) is similar to that of the traditional partitional clustering scheme. The difference is that the memberships of the labeled data $\mathbf{U}^l$ remain fixed during the optimization process. Therefore, the updating procedure for the cluster representations can be formulated as

$$\mathbf{v}_i = \frac{\sum_{j=1}^{n^l} (u_{ij}^l)^p \, \mathbf{x}_j^l + \lambda \cdot \sum_{j=1}^{n^u} (u_{ij}^u)^p \, \mathbf{x}_j^u}{\sum_{j=1}^{n^l} (u_{ij}^l)^p + \lambda \cdot \sum_{j=1}^{n^u} (u_{ij}^u)^p} \qquad \forall i \in \{1, \ldots, c\} \tag{4.6}$$

so that the first sum regarding the labeled data introduces a constant bias towards the starting solution. The parameter $\lambda \in [0, 1]$ allows the relative weight of the unlabeled data to be adjusted with respect to the labeled data.[55] At this point, however, we will not further consider this weighting factor and, thus, assume $\lambda = 1$. Only note that it is also possible to introduce pattern-specific weights so as to tailor the computation to agree more or less with any particular labeled pattern according to its importance or quality.[56]

Finally, note that treating labeled data as a constant bias assumes that their labels are correct. This is a reasonable assumption, since labels are only required for very few patterns. Nevertheless, if the labeled data were noisy or partly incorrect, it would be better to permit memberships of the labeled data to change as well during the optimization process.

Associating Clusters with Class Labels

The guidance of the clustering algorithm by labeled data increases the chances that the resulting solution resembles what we are looking for: a class structure which matches the classes already given by means of the labeled data. As discussed previously, however, clustering may find meaningful clusters, while it will not provide any association between these clusters and predefined classes. Hence, they cannot be used to assign class labels to new documents. With the help of labeled data, this drawback can easily be fixed.

A common approach to associating clusters with class labels is through supervised learning on the basis of some labeled data. The idea is to learn a simple classifier from X^l and, then, use this classifier to assign class labels to the cluster representations. The most straightforward approach to doing exactly this is to assign class labels in K-nearest-neighbor fashion. Let $T(\mathbf{v}_i)$ denote the label of cluster representation $\mathbf{v}_i$. The K-NN rule yields[57]

$$T(\mathbf{v}_i) = \arg\max_{c \in \mathcal{C}} K_c \qquad \forall i \in \{1, \ldots, c\} \tag{4.7}$$

where K_c denotes the number of labeled patterns among the K nearest neighbors that belong to class c. Recall that $\mathcal{C}$ denotes the set of classes of the supervised classification task. Note that this approach is, in fact, independent of prior guidance by the labeled data. Hence, it is applicable to the result of any type of clustering algorithm. Nonetheless, note that this approach cannot guarantee that each of the predefined classes actually exists among the class labels assigned to the cluster representations.

On the basis of the aforementioned guidance by labeled data, clusters and class labels are, in fact, already associated. In particular, according to Equation (4.5), each cluster

[55]See Nigam *et al.* (2000).

[56]See Bensaid *et al.* (1996), p. 363.

[57]See Section 203 (p. 60) for a description of the K-nearest-neighbor rule.

representation $\mathbf{v}_i$ is initialized as the centroid of all patterns belonging to class c_i, yielding

$$T(\mathbf{v}_i) = c_i \qquad \forall i \in \{1, \ldots, c\} \tag{4.8}$$

where the number of clusters equals the number of classes, i.e. $c = k$. Unless the subsequent incorporation of unlabeled data causes the cluster representations to change in such a way that their resemblance to the labeled patterns would yield different class assignments, the initial associations between clusters and class labels are reasonable. Note that this approach guarantees that each of the predefined classes is represented by a cluster. In the following, we assume that the associations between clusters and class labels as given by the labeled-data-guided initialization are used.

Once associations between clusters and class labels are established, the clustering output can be used to design a classifier. We distinguish two ways of doing so: utilizing either the membership matrix or the cluster representations. A common approach to classification based on the cluster representations is through application of the one-nearest-neighbor rule. Yet, since the cluster representations are prototypical patterns rather than true patterns, this rule is often referred to as the *one-nearest-prototype* (1-NP) rule.[58] The classification of a new pattern $\mathbf{x}$ is given by

$$H_{\text{1-NP}}(\mathbf{x}) = T(\arg\min_{\mathbf{v} \in V} \|\mathbf{x} - \mathbf{v}\|) \tag{4.9}$$

where $\|\cdot\|$ is typically the same norm employed during the clustering process.

Note that this one-nearest-prototype rule is similar to the single-prototype classifier set out in Section 197 (pp. 58 ff.). Albeit based on different learning techniques, both classifiers model each class by exactly one prototype. Yet, while the nearest-prototype rule is formulated based on a general norm and returns the class label of the prototype with minimal distance to a new pattern, the single-prototype classifier, in the context of text classification, is formulated on the basis of the cosine similarity measure and, thus, returns the class labels of the prototype with maximum similarity to a new document. Nevertheless, recall that distance and similarity are in some way inversely related.[59]

Rather than using the cluster representations as the basis for a nearest-prototype classifier, the second approach to classifying new data based on clustering results exploits the final partition of the unlabeled data through supervised learning. For now, assume that supervised learning requires hard class labels. So, we assign hard class labels y_j^u to the unlabeled patterns $\mathbf{x}_j^u$ on the basis of the final membership matrix $\mathbf{U}^u$. Note that in case soft labels were used, the labels need to be hardened. This can be accomplished simply by assigning the class label of the cluster to which the particular pattern has maximum membership, which is known as *maximum membership conversion*.[60] So, the following rule can generally be used to set the class labels of the unlabeled data:

$$y_j^u = \arg\max_{i \in \{1,\ldots,c\}} u_{ij}^u \qquad \forall j \in \{1, \ldots, n_u\} \tag{4.10}$$

[58]See Bezdek (1981), p. 228.

[59]See Section 176 (p. 52).

[60]See Bensaid *et al.* (1996), p. 861.

Ties are resolved arbitrarily. Now, the unlabeled patterns with the imputed class labels in addition to the labeled patterns may serve as a training set for any supervised learning algorithm. The classifier learned from this augmented training set may subsequently be used to classify new data. This idea leads directly to the final step in the approach to our general semi-supervised learning framework. Also, note that the approach presented so far fits in with the perspective of combining supervised and unsupervised learning in a two-step process as set out in Section 4.2.

Using Supervised Learning to Describe Clusters

Most common clustering algorithms represent their clusters as centroids, or means, of the associated patterns as formulated in Equation (4.1). Instead of this point-prototype representation, more sophisticated representations might be used to better describe clusters. For instance, this might allow taking cluster size and shape into account. Common extensions are representations that are linear varieties, hyperspherical shells, and regression models.[61] We consider an alternative approach that explicitly uses supervised learning to describe clusters. Nevertheless, note that the computation of the cluster representations carried out at each iteration based on the current state of the membership matrix follows, in fact, a simple and implicit supervised learning scheme. In the description of the general semi-supervised learning framework, we explicitly formulate this concept of using supervised learning to describe clusters.

Consider that the objective of supervised learning is to construct from a set of labeled patterns a classifier that can identify a finite set of classes so as to be able to assign class labels to new patterns. In this sense, supervised learning will produce descriptions of the predefined classes in the form of classifiers. The representation language used for these descriptions varies greatly among different learning algorithms.[62] Without regard to the form of the class descriptions, the classifiers obtained through supervised learning from the current state of the class labels may be used to represent clusters. Note, however, that using any type of supervised learning comes with all the common pitfalls. In particular, there is the problem of overfitting. Clearly, it is possible to find a particular learner that perfectly fits the training data if it does not contain any errors. But as we know, this may lead to poor generalization ability. Albeit crucial when choosing an appropriate learning algorithm, this issue is highly data-dependent and will not be discussed here.

The idea of the general semi-supervised learning framework is to replace computations of cluster representations through any supervised classifier design. The underlying learning algorithm will be referred to as the *base learner*. Rather then applying unsupervised learning and supervised learning in a two-step process as described above, this framework amounts to interweaving supervised and unsupervised learning to yield an integrated approach. Algorithm 4.3 shows the pseudo code of this integrated framework.

Note that, so far, we have used different labeling notations for supervised and unsupervised learning. Specifically, in the supervised task, we have denoted the class labels of a

[61] See Bensaid and Bezdek (1998), pp. 627–628, and the references therein.

[62] See Section 3.3 for some common text learning algorithms and their representations.

Algorithm 4.3 Semi-Supervised Learning Framework

```
Input: dataset $X = X^l \cup X^u$ consisting of non-empty labeled and unlabeled subsets
1: set number of clusters $c$ to number of classes $k$
2: initialize cluster memberships $\mathbf{U}^l$ and $\mathbf{U}^u$
3: learn initial classifier $H$ from $X^l$ with true labels $\mathbf{U}^l$
4: repeat
5:     use classifier $H$ to evaluate classification scores for unlabeled data $X^u$
6:     transform classification scores into cluster memberships $\mathbf{U}^u$
7:     re-learn classifier $H$ from $X^l$ with true labels $\mathbf{U}^l$ and $X^u$ with imputed labels $\mathbf{U}^u$
8: until stopping criterion is fulfilled
Output: classifier $H$
```

pattern by a scalar value indicating its class. To denote labels in the unsupervised task, we have introduced the concept of the membership matrix which represents the degree of association between each pattern and each cluster by a specific value. Above, we have transformed the membership notation to scalar class labels to make use of the clustering output in supervised fashion. Yet, to integrate supervised and unsupervised learning, we require that both learning approaches use the same labeling notation. Hence, we aim at adjusting the notation of the class labels employed for the supervised learning task to that employed for the unsupervised task.

Assume a labeled pattern $\mathbf{x}$ belonging to class $c_{i'} \in \mathcal{C}$. Rather than using the scalar class label $y = c_{i'}$, we now use the label vector $\mathbf{y} = (y_1, \ldots, y_i, \ldots, y_k)^T \in [0,1]^k$, where k is the number of classes and

$$y_i = \begin{cases} 1 & \text{if } i = i' \\ 0 & \text{otherwise} \end{cases} \qquad \forall i = 1, \ldots, k \tag{4.11}$$

Note that, for hard labels, both the scalar and the vector notation are equivalent. Yet, the vector notation corresponds to the notation of membership degrees as employed to express labels in unsupervised learning. In particular, if the number of clusters equals the number of classes, the label vector $\mathbf{y}_j^l$ of pattern $\mathbf{x}_j^l \in X^l$, $j = 1, \ldots, n_l$, corresponds to the column vector $\mathbf{u}_j^l$ in the aforesaid membership matrix $\mathbf{U}^l = (\mathbf{u}_j^l)$.

Due to the correspondence between class label vectors and the columns of a membership matrix, provided that the number of clusters equals the number of classes, we can use the membership matrices $\mathbf{U}^l$ and $\mathbf{U}^u$ as class labels for the labeled and unlabeled datasets X^l and X^u, respectively. Typically, the true labels $\mathbf{U}^l$ are hard as they are given by the user. For the unlabeled data, however, the possible types of the imputed labels depend on the supervised learning algorithm to be used: hard labels can always be used, whereas soft labels may only be used if the underlying supervised learner can incorporate soft labels; see Section 4.3.3.

Now, let us turn to the semi-supervised framework as set out in Algorithm 4.3. As introduced previously, the number of clusters is set to the number of classes as given in terms of the supervised learning task. Furthermore, the membership matrix $\mathbf{U}^l$ of the labeled patterns is initialized according to the class labels given in Equation (4.4). The membership matrix $\mathbf{U}^u$ of the unlabeled patterns is initialized so as to ensure a significant difference to the subsequent update. The remainder of the algorithm reads as follows.

First, rather than computing initial cluster representations, an initial classifier H is built based solely on the labeled data X^l with the true labels inherent to $\mathbf{U}^l$ (line 3). Note that the homogeneity assumption pertaining to the classes may be relaxed if the base learner can handle heterogeneous class definitions. Then, the semi-supervised algorithm iterates three steps, which make up a bootstrapping iteration (see p. 89) as described below, until the membership degrees $\mathbf{U}^u$ given to the unlabeled data X^u do not change significantly from one iteration to the next or until a predefined maximum number of iterations has been exceeded. Note that the additional condition pertaining to the maximum number of iterations is introduced here to prevent endless oscillations. This may be necessary as we can no longer ensure convergence when using an arbitrary learning algorithm to describe clusters.[63] Finally, the classifier H resulting after the last iteration is returned.

The three steps making up a bootstrapping iteration, which is the core of an approach to bootstrapping a classifier as introduced in Section 295 (p. 89), are as follows:

1. The current classifier H is used to obtain classification scores for each unlabeled pattern (line 5). Note that there is no constraint on the response of the classifier. As a rule, classification scores are hard, probabilistic, or possibilistic by nature.

2. To abstract from the classifier's response, the classification scores obtained are transformed into the membership matrix $\mathbf{U}^u$, so that they are appropriate for the underlying learning algorithm (line 6).[64]

3. On the basis of both the labeled patterns X^l with their known class labels $\mathbf{U}^l$ and the unlabeled patterns X^u with their imputed class labels $\mathbf{U}^u$, a refined classifier H is built (line 7).

Note that through the integration of supervised and unsupervised learning, the semi-supervised framework outputs a classifier that can be used to assign the class label to any new pattern. Hence, this framework is to be characterized as an approach to classifier design rather than clustering. So, this contrasts with the semi-supervised clustering algorithm obtained at an intermediate step (Algorithm 4.2) and described in a similar way by Pedrycz (1984) and Bensaid *et al.* (1996).

Also, this integration has turned the two-step process of combining supervised and unsupervised learning into an approach which is—although not in a probabilistic context—very similar to the basic idea of the EM algorithm when learning in the presence of missing class labels. To see this, consider that, in the EM algorithm, two steps are iterated: the *expectation step* (E-step) and the *maximization step* (M-step). In the E-step, probabilities for each unlabeled pattern belonging to any of the possible classes are estimated. Clearly, this corresponds to the first step (line 5) in our semi-supervised framework, which computes classification scores for each of the unlabeled patterns. In the M-step, a new model

[63]See Section 4.3.3 for further discussions on this issue.

[64]See Section 4.3.3 for more details.

for estimating the probabilities of unlabeled patterns belonging to any of the classes is determined through maximum likelihood or maximum a posteriori estimation on the basis of the currently imputed probabilistic class labels. This corresponds to the third step (line 7) of the semi-supervised framework, which designs a classifier based on the training data with respect to the current state of imputed class labels. As a result, the semi-supervised framework described here can be considered as an EM-like approach to classifier design.

4.3.2 The Semi-Supervised Text Learning Framework

In the preceding, we have derived a framework for semi-supervised learning from labeled and unlabeled data using a general terminology. Now, we return to the terminology used in the context of text classification, yielding a semi-supervised text learning framework as proposed in Lanquillon (2000b). In addition, we summarize the assumptions which have been made during the construction of this learning framework. Particular instantiations of the semi-supervised learning framework will be discussed in the next section.

Input to the algorithm is a set of training documents D. For some subset of the documents $\mathbf{d}_j^l \in D^l$, we know the true class labels and, for the rest of the documents $\mathbf{d}_j^u \in D^u$, the class labels are unknown. Thus we have a disjoint partitioning of the training documents into a labeled set and an unlabeled set of documents, so that $D = D^l \cup D^u$. The task is to build a classifier H based on the training documents D which would predict the class labels of new, previously unseen documents.

The assumption that should hold to allow application of the semi-supervised text learning framework are as follows:

- *Labeled and unlabeled documents come from the same source.* Each document, labeled or unlabeled, is assumed to be drawn from the same document source. So, the only difference between labeled and unlabeled documents is the class labels given the labeled documents by the user. Furthermore, each document belongs to any one of the k predefined classes. This is also known as the *closed-world assumption*. As a consequence, this justifies classifying the unlabeled document into the same classes as the labeled documents and, then, using them as additional training data. Note, of course, that we still assume that the document source is stationary.

- *Each class is represented by at least one labeled document.* The set of labeled documents may be small. Yet, we require that there be at least one labeled training document per class so as to ensure that some seed information is available for each class. Otherwise, it may not be possible to assign the correct class labels to the unlabeled documents.

- *Labeled documents allow better-than-random classification.* Because of their scarcity, the labeled documents may not be fully representative of the classes. Yet, we assume that they provide a rough description that allows learning a classifier which performs better than random. This allows bootstrapping the classifier based on its own effort, namely the assignment of class labels to unlabeled documents.

- *The base learner can distinguish among classes.* Just as for the plain supervised task, the underlying supervised learning algorithm must be able to learn the classes. So, obviously, the same assumption made for the base learner must also hold in the semi-supervised learning framework. For instance, some common classifiers such as the naïve Bayes classifier or the Rocchio variants require that the classes be homogeneous, so that a single prototype is sufficient to represent a class.

- *Labeled documents are correctly labeled.* The class labels of the labeled documents will remain fixed during the iterations of the semi-supervised learning framework. This assumes that the class labels given by the user are correct. This is reasonable because we require only a relatively small number of documents to be labeled. This is, in fact, no additional assumption imposed by semi-supervised learning; rather, class labels are also assumed to be correct in supervised learning.

The semi-supervised text learning framework as shown in Algorithm 4.4 reads as follows:

1. The number of clusters is the same as the number of classes, i.e. $c = k$, since learning the k classes $\mathcal{C} = \{c_1, \ldots, c_k\}$ is to be supported through unlabeled data. To emphasize this, we will use the parameter k rather than c in the following.

2. The $(k \times n_l)$ membership matrix $\mathbf{U}^l = (\mathbf{u}_j^l)$, for $j = 1, \ldots, n_l$, holds the true class labels of the labeled documents in the form of label vectors, i.e. $\mathbf{u}_j^l = \mathbf{y}_j^l$. A label vector $\mathbf{y}^l$ is obtained from the scalar class label $y^l = T(\mathbf{d}^l)$ according to Equation (4.11). Note that the class labels of the labeled documents are assumed to be correct and, therefore, the membership matrix $\mathbf{U}^l$ will remain fixed. The $(k \times n_u)$ membership matrix $\mathbf{U}^u = (\mathbf{u}_j^u)$, for $j = 1, \ldots, n_u$, holds the imputed class labels of the unlabeled documents in the form of label vectors as above. The class labels $\mathbf{u}_j^u = \mathbf{y}_j^u$ may change at each iteration, indicating the current partition of the unlabeled documents among the k classes. Each entry of $\mathbf{U}^u$ is initially set to zero to emphasize that the initial classifier H will be learned solely from labeled data. Also, this initialization ensures that the subsequent update of $\mathbf{U}^u$ may differ significantly from $\mathbf{U}^u$ so as not to cause the process to terminate early.

3. The set of labeled documents D^l with class labels $\mathbf{U}^l$ serves as seed information. So, an initial classifier H is learned from the labeled documents only. Note that any supervised learning algorithm might be used. This base learner, however, is then assumed to be used throughout the semi-supervised framework.

4. The remaining three steps are repeated until the class labels U^u of the unlabeled documents do not change significantly from one iteration to the next or until a predefined maximum number of iterations has been exceeded. The condition pertaining to the membership matrix of the unlabeled documents typically checks convergence of a clustering algorithm. The additional condition pertaining to the maximum number of iterations is introduced to prevent endless oscillations as we can no longer ensure convergence when using an arbitrary base learning algorithm. We will have more to say on this in the following subsection.

Algorithm 4.4 Semi-Supervised Text Learning Framework

```
Input: dataset $D = D^l \cup D^u$ consisting of non-empty labeled and unlabeled subsets
1: set number of clusters $c$ to number of classes $k$
2: initialize cluster memberships $\mathbf{U}^l$ and $\mathbf{U}^u$
3: learn initial classifier $H$ from $D^l$ with true labels $\mathbf{U}^l$
4: repeat
5:     use classifier $H$ to evaluate classification scores for unlabeled data $D^u$
6:     transform classification scores into cluster memberships $\mathbf{U}^u$
7:     re-learn classifier $H$ from $D^l$ with true labels $\mathbf{U}^l$ and $D^u$ with imputed labels $\mathbf{U}^u$
8: until stopping criterion is fulfilled
Output: classifier $H$
```

5. The current classifier H is used to classify the unlabeled documents. We assume that, for each unlabeled document, the classifier responds with a classification score for each class. Depending on the classifier, these scores may be hard, probabilistic, or possibilistic by nature. Typically, a classification decision is made by selecting the class with maximum score. In the context of semi-supervised learning, however, the way in which the classification scores are processed depends on the base learner.

6. To abstract from the classifier's response, the scores obtained are transformed into the class labels $\mathbf{U}^u$, so that they are appropriate for the base learner. Note that using hard labels will work with any learning algorithm as these resemble conventional class labels. Some classifiers may be able to learn from examples with soft class labels. Yet, whether learning from examples with soft labels is superior to learning from examples with hard labels is to be validated carefully, as is done below.

7. The classifier H is re-learned from both the labeled documents D^l with true class labels $\mathbf{U}^l$ and the unlabeled documents D^u with the current state of the imputed class labels $\mathbf{U}^u$. As a rule, all unlabeled documents are considered when refining the classifier.[65] Yet, we exclude an unlabeled document from the training set if its classification scores are all zero. This might occur when the number of labeled documents is very small and the unlabeled document does not have any terms in common with any of the labeled documents. Typically, the number of unlabeled documents which are excluded for this reason decreases as the algorithm precedes, because, then, more co-occurrence information will have been exploited and more index terms can be associated with particular classes.

4.3.3 Instantiations

To apply the semi-supervised text learning framework to actually solve classification tasks, two things must be specified: the base learning algorithm and the function which

[65]Recall that this contrasts with self-training approaches (see p. 90), which add only a small number of confidently classified unlabeled documents to the current training set.

transforms classification scores into class labels. In the following we discuss the use of three different base learners in the semi-supervised setting: the single-prototype classifier, the multinomial naïve Bayes classifier, and the support vector machine. Results for these semi-supervised classifiers will be presented in Section 4.4. Note that initial experiments showed that nearest-neighbor rules do not qualify for semi-supervised learning as they do not generalize beyond the training documents and, thus, cannot exploit co-occurrence patterns, which, however, has been identified as the key to success in exploiting unlabeled data in addition to labeled data.[66]

Semi-Supervised Single-Prototype Classifier

The most straightforward instantiation of the semi-supervised text learning framework is by use of the single-prototype classifier, yielding the *semi-supervised single-prototype classifier* (ssSPC). Using the single-prototype classifier as base learner is the most natural choice, since it is the supervised variant of representing clusters, i.e. classes, by a single prototype. So, in its basic form, the single-prototype classifier may, in fact, be seen as the underlying learning algorithm of conventional hard c-means clustering.

A central issue is the choice of the transformation function to convert classification scores into class labels. Undoubtedly, the single-prototype classifier may be used in combination with any labeling type; Equation (4.6) illustrates this. But which labeling type is most appropriate? As text classification experiments with soft labels show, all class prototypes produced by the semi-supervised single-prototype classifier tend to converge to the same prototype. Clearly, this will not permit meaningful classification of new documents. As a consequence, the choice of an appropriate transformation function is simple: classification scores should be converted into hard labels.

Algorithm Details. The classification scores are obtained as similarities $\mathrm{sim}(\mathbf{d}^u, \mathbf{p}_i)$ between a particular unlabeled document $\mathbf{d}^u \in D^u$ and the prototypes $\mathbf{p}_i$, $i = 1, \dots, k$, of the current single-prototype classifier. Note that the class prototypes of the single-prototype classifier are used as cluster representations, i.e. $\mathbf{p}_i = \mathbf{v}_i$.

A hard class label vector $\mathbf{u}^u = (u_i^u)$, aka $\mathbf{y}^u$, is obtained by hardening with respect to the maximum classification score, yielding

$$u_i^u = \begin{cases} 1 & \text{if } i = \arg\max_{i' \in \{1,\dots,k\}} \mathrm{sim}(\mathbf{d}^u, \mathbf{p}_{i'}) \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{1, \dots, k\} \qquad (4.12)$$

Based on the current class label vectors $\mathbf{U}^u$ for the unlabeled documents D^u and the fixed class label vectors U^l for the labeled documents D^l, the prototypes $\mathbf{p}_i$ are computed according to Equation (4.6), yielding[67]

$$\mathbf{p}_i = \frac{\sum_{j=1}^{n^l} u_{ij}^l \, \mathbf{d}_j^l + \lambda \cdot \sum_{j=1}^{n^u} u_{ij}^u \, \mathbf{d}_j^u}{\sum_{j=1}^{n^l} u_{ij}^l + \lambda \cdot \sum_{j=1}^{n^u} u_{ij}^u} \qquad \forall i \in \{1, \dots, k\} \qquad (4.13)$$

[66]See Lanquillon (2000a).

[67]Note that we have dropped the fuzziness paramter p here.

Recall that the first sum pertaining to the labeled documents introduces a constant bias towards the initial class prototypes. The parameter $\lambda \in [0, 1]$ allows the relative weight of the unlabeled documents to be adjusted with respect to the labeled documents.[68] At this point, however, we will not further consider this weighting factor and assume $\lambda = 1$.

Proof of Convergence. As the description of the semi-supervised single-prototype classifier has shown, it is a generalization of hard c-means clustering. The key difference to the well-known c-means clustering algorithm is that the memberships of some documents, namely the labeled documents, remain fixed during the clustering iterations. The traditional c-means algorithm is certain to converge to a local minimum after a finite number of iterations. But what about the semi-supervised single-prototype classifier?

The proof of convergence for the traditional c-means algorithm is based on the fact that there is only a finite number of hard partitionings of training patterns into classes and also that the sum of squared distances between cluster representations and training patterns—previously introduced as the objective function J—does not increase while iteratively updating the membership matrix and the cluster representations.[69] Hence, the algorithm must converge after a finite number of steps.[70]

The calculation of cluster representations based on training documents and their hard class labels is the same for the semi-supervised single-prototype classifier. Hence, this step does not increase J. As mentioned above, the update rule for the membership matrix $\mathbf{U}$ differs from the traditional c-means algorithm. The label vectors of the labeled documents remain fixed, while the unlabeled documents are always assigned to the most similar cluster representation. The latter is equivalent to the traditional c-means algorithm and, thus, also does not lead to an increase in J. Furthermore, note that fixed class labels cannot cause J to change. Hence, the semi-supervised single-prototype classifier will also converge to a local minimum after a finite number of steps.

Semi-Supervised Naïve Bayes Classifier

Like the single-prototype classifier, the naïve Bayes classifier is an instance-averaging approach to learning classes. As such, it is particularly well suited to represent clusters in the semi-supervised text learning framework. Here, we consider using the multinomial naïve Bayes classifier as the base learning algorithm, yielding the *semi-supervised naïve Bayes classifier* (ssNB).

Formulated in a probabilistic framework, the naïve Bayes classifier is doomed to be used in combination with probabilistic class labels. Hence, the classification scores returned by the naïve Bayes classifier, i.e. the class-specific posterior probabilities given a document, need not be transformed; they can be used directly as probabilistic labels. As experiments will show, the naïve Bayes classifier—unlike the single-prototype classifier—performs

[68]Nigam *et al.* (2000) use this weighting parameter to augment their EM-based naïve Bayes classifier.

[69]Note that, here, we have dropped the subscript p from the objective function J_p, as the corresponding parameter p has no effect on the outcome of hard c-means clustering.

[70]See Selim and Ismail (1984).

very well when employing soft labels. The reason for this might be that the naïve Bayes classifier typically responds with probability estimates which are extremely close to zero or one. As a consequence, the resulting class labels can almost be considered hard rather than soft.

Note that this instantiation of the semi-supervised text learning framework is, in fact, equivalent to the EM-based naïve Bayes classifier for learning from labeled and unlabeled documents proposed by Nigam *et al.* (1998) and (2000). As a matter of fact, our semi-supervised text learning framework is meant to generalize Nigam's EM-based approach so that any learning algorithm may be used in place of the naïve Bayes classifier.

Algorithm Details. Assume probabilistic class labels. Hence, the entries of the membership matrix $\mathbf{U}^l$ represent the true probabilities of class membership for each labeled document in the predefined classes:

$$u_{ij}^l = \mathrm{P}(c_i|\mathbf{d}_j^l) \qquad \forall i \in \{1,\ldots,k\}, \forall j \in \{1,\ldots,n_l\} \tag{4.14}$$

As a rule, these probabilities take on integer values zero or one as they are provided by the user. The entries of the membership matrix $\mathbf{U}^u$ are interpreted as estimates of the probabilities of class membership for each unlabeled document, yielding

$$u_{ij}^u = \hat{\mathrm{P}}(c_i|\mathbf{d}_j^u) \qquad \forall i \in \{1,\ldots,k\}, \forall j \in \{1,\ldots,n_u\} \tag{4.15}$$

Naturally, the class-specific probabilities for each document, both labeled and unlabeled, are required to sum up to unity. As a consequence, we observe the following relationships:

$$\sum_{j=1}^{n_l}\sum_{i=1}^{k} u_{ij}^l = n_l \qquad \text{and} \qquad \sum_{j=1}^{n_u}\sum_{i=1}^{k} u_{ij}^u = n_u \tag{4.16}$$

Recall that the naïve Bayes classifier combines class prior probabilities and the probabilities of a document given each of the possible classes by Bayes' theorem to obtain posterior probabilities of the classes given the document.[71] Below, we present modified formulae to estimate the probabilities involved according to Nigam *et al.* (2000).

On the basis of both labeled and unlabeled documents, the class priors θ_{c_i} are computed as Laplace corrected maximum likelihood estimates, yielding

$$\hat{\theta}_{c_i} = \frac{1 + \sum_{j=1}^{n_l} u_{ij}^l + \lambda \cdot \sum_{j=1}^{n_u} u_{ij}^u}{k + n_l + \lambda\, n_u} \qquad \forall i \in \{1,\ldots,k\} \tag{4.17}$$

Note that the Laplace priors supplement each of the k possible classes with a count of one. This type of smoothing is introduced here so as to avoid difficulties with inaccurate estimates when only very few labeled documents are available. As introduced for the semi-supervised single-prototype classifier, the parameter $\lambda \in [0,1]$ allows adjusting of the relative weight of the unlabeled documents with respect to the labeled documents. For now, we assume $\lambda = 1$. Yet, we will have more to say on this later in this chapter.

[71]See Section 206 (pp. 61 ff.)

Under the naïve Bayes assumption, estimating the probability of a document given a class requires estimates of the probabilities of all terms given the class. Let these term probabilities be denoted by $\theta_{t|c_i} = \mathrm{P}(t|c_i)$ for each class c_i, $i = 1, \ldots, k$, and each index term $t \in \mathcal{V}$. As a modification of Equation (3.54) (p. 65), the parameters $\theta_{t|c_i}$ are computed as the Laplace corrected maximum likelihood estimate based on both labeled and unlabeled documents, yielding

$$\hat{\theta}_{t|c_i} = \frac{1 + \sum_{j=1}^{n_l} u_{ij}^l \; \mathit{tf}_{\mathbf{d}_j^l}(t) + \lambda \cdot \sum_{j=1}^{n_u} u_{ij}^u \; \mathit{tf}_{\mathbf{d}_j^u}(t)}{m + \sum_{s=1}^{m} \sum_{j=1}^{n_l} u_{ij}^l \; \mathit{tf}_{\mathbf{d}_j^l}(t_s) + \lambda \cdot \sum_{s=1}^{m} \sum_{j=1}^{n_u} u_{ij}^u \; \mathit{tf}_{\mathbf{d}_j^u}(t_s)} \tag{4.18}$$

where $\mathit{tf}_{\mathbf{d}}(t)$ denotes the number of occurrences of term t in document $\mathbf{d}$, m is the number of index terms, and the parameter $\lambda \in [0, 1]$ is used as described above.

Proof of Convergence. Note that this instantiation of our semi-supervised learning framework has a strong probabilistic framework. In fact, it is an application of the EM algorithm using the naïve Bayes classifier to learn the underlying model parameters. "In its maximum likelihood formulation, EM performs hill-climbing in data likelihood space, finding the classifier parameters that locally maximize the likelihood of all the data—both the labeled and the unlabeled."[72] At each iteration, the EM algorithm is guaranteed to find parameters which have equal or higher likelihood than that at the previous iteration. As the likelihood is naturally bound above, the EM algorithm—and, as a result, the semi-supervised naïve Bayes classifier—is guaranteed to converge. As a rule, convergence is determined when observing a below-threshold change in the parameters involved in the optimization process. Finally, note that, in most applications, the solution obtained by the EM algorithm is a local maximum. Yet, in some rare cases, we might also observe convergence to a saddle point or even a local minimum.[73]

Semi-Supervised Support Vector Machine

Both the single-prototype classifier and the naïve Bayes classifier are similar in that they produce class descriptions by aggregating class-specific information as given by the training examples. To contrast with this, we now consider the support vector machine, which is currently one of the most successful text learning algorithms, to be applied as the base learner, yielding the *semi-supervised support vector machine* (ssSVM). As previously, we will use linear support vector machines only.

In its current form, the optimization problem to be solved for the support vector machine requires that the class labels of the unlabeled data be hard. Hence, class label vectors are obtained by hardening classification scores with respect to the maximum classification score as described for the semi-supervised single-prototype classifier.

Learning the support vector machine from the combined set of labeled documents with their true class labels and the unlabeled documents with their imputed class labels works

[72]From Nigam *et al.* (2000), pp. 104–105.

[73]See McLachlan and Krishnan (1997), p. 33.

in the known supervised manner. Note that, in the context of support vector learning, problems with $k > 2$ classes require splitting into binary subtasks. The same applies to the semi-supervised support vector machine.

Note that, for the single-prototype and naïve Bayes instantiations, we have introduced a parameter λ to adjust the relative weight of the unlabeled documents with respect to the labeled documents. For the support vector machine, we do not have this facility. Nevertheless, a way to adjust the importance of unlabeled data during the optimization process might be to adapt the behavior by which non-separable examples are punished or excluded from the training set.[74]

Despite the hard class labels and the thus finite number of possible partitions, we cannot guarantee that the semi-supervised support vector machine will converge after a finite number of steps. Instead, we may observe oscillations between particular states of the hard membership matrix $\mathbf{U}^u$ for the unlabeled documents. To ensure termination anyhow, a maximum number of iterations should be specified.

4.3.4 Augmented Framework

In the preceding, we have developed our semi-supervised text learning framework in its basic form. The next section will provide empirical evidence of how effective the three instantiations proposed perform in real-world text classification tasks. Although superior in many applications, in some situations the semi-supervised learning algorithms perform worse than the underlying supervised base learner.[75] The following discussion concerns the instantiations based on the single-prototype classifier and the naïve Bayes classifier and provides an augmentation to the basic semi-supervised learning framework.

As pointed out by Nigam *et al.* (2000), some of the performance degradation observed may be accounted for by violations of the assumptions made for the semi-supervised learning framework. In particular, it is assumed that classes are learnable by the base learner. So, depending on the base learner, this may require homogeneous classes. The issue of having heterogeneous classes, which the base learner may not be able to learn, will be discussed in Chapter 5. Another assumption states that both labeled and unlabeled documents come from the same source. This assumption may be violated in real-world problems. In addition, even the assumption that each document is generated by a particular document source with class-specific parameters may not be adequate to describe the problem setting. If these assumptions are not true, the partition in the unlabeled data identified through clustering might not be in correspondence with the classes of the supervised learning task.[76] As a consequence, exploiting unlabeled data might degrade classification performance. In the following, we address the problem of having a partition in the unlabeled data which does not fully support classification.

Violations of some assumptions may cause the correspondence between clusters and classes to become impaired. Yet, a degradation in classification performance has also

[74]See Section 243 (p. 70).

[75]Also, see Nigam *et al.* (2000) and Lanquillon (2000a) and (2000b).

[76]See Nigam *et al.* (2000), p. 114.

been observed in situations in which the assumptions are presumably true, as both the labeled and the unlabeled sets were artificially created based on the same document collection. Hence, there ought to be further reasons accounting for performance degradation. Assume that the labeled and unlabeled document sources are similar enough to permit application of the semi-supervised learning framework. Still, empirical results show that the benefit we may get from the unlabeled documents largely depends on the number of labeled documents and the number of unlabeled documents available for training.[77] As the analysis in the subsequent section suggests, this may be accounted for by the uncertainty inherent to unlabeled documents in combination with their imputed class labels. Predicted by a classifier rather than provided manually by a user, the class labels of unlabeled documents are error prone and, therefore, uncertain. Just like violated assumptions pertaining to the document source, usage of incorrectly classified unlabeled data in the learning process may cause an impaired correspondence between clusters and classes. And, this becomes particularly apparent by causing performance degradation when a reasonably accurate classifier can already be learned from labeled documents only.[78]

We might argue that there is no need to use unlabeled data when the set of labeled training documents is large. Yet, typically we do not know just when this training set is deemed large enough and, hence, when to better learn without unlabeled data. So, to remedy the problem caused by an impaired correspondence between clusters and classes—due to either incorrect class labels or violated assumptions pertaining to the document source—the influence of the unlabeled data on the semi-supervised learner should be modulated.[79] The parameter $\lambda \in [0, 1]$ introduced previously for the semi-supervised single-prototype classifier and the semi-supervised naïve Bayes classifier allows the relative weight of the unlabeled documents to be adjusted with respect to the labeled documents. When setting $\lambda = 1$, we obtain the semi-supervised learning framework in its basic form as labeled and unlabeled data are given the same weight. Choosing $\lambda < 1$, in contrast, allows discounting unlabeled data, yielding an augmented version of the semi-supervised learning framework.

Now, the key issue is to find a suitable value for λ. Nigam *et al.* (2000) determine λ on the basis of leave-one-out cross-validation and show that performance no longer degrades when discounting unlabeled documents in this way. Also, they observe an inverse relationship between the number of labeled documents n_l and the best weighting factor λ. To abstract from the number of classes k, we heuristically determine the weighting factor as

$$\lambda = \frac{1}{1 + \log(\frac{n_l}{k})} \qquad (n_l \geq k) \tag{4.19}$$

The logarithmic relationship between λ and the average number of labeled documents per class accounts for the fact that, on a log scale, accuracy often increases linearly with the number of labeled training examples before reaching a plateau.[80] Note that λ takes on its maximum, i.e. $\lambda = 1$, when learning from only one labeled document per class and it approaches its minimum, i.e. $\lambda \to 0$, as the number of labeled documents increases.

[77] See Lanquillon (2000b).
[78] Also, see Nigam *et al.* (2000) and Lanquillon (2000a) and (2000b).
[79] See Nigam *et al.* (2000), p. 114.
[80] See Section 276 (p. 84).

4.4 Experimental Evaluation

This section provides empirical evidence that learning from both labeled and unlabeled documents through the semi-supervised variants of the single-prototype classifier, the naïve Bayes classifier, and the linear support vector machine as set out above often outperform the traditional supervised learners, which learn from labeled documents only. We present experimental results on the three text corpora based on which the effect of a small training set size has been demonstrated in Section 276 (p. 82 ff.): the 20 Newsgroups dataset, the WebKB dataset, and a subset of the TREC dataset.

Particularly when labeled training data is scarce, results show substantial improvements in accuracy. For example, learning from only one labeled training document per class for the 20 Newsgroups dataset, the traditional single-prototype classifier achieves only 22% accuracy, whereas the semi-supervised single-prototype classifier reaches about 52% accuracy when adding 10,000 unlabeled training documents. Put in other words, semi-supervised learners may achieve a specific level of classification accuracy with much less labeled training data than their supervised variants. Yet, sometimes—especially when the number of labeled training examples is large—the instantiations of the basic semi-supervised framework perform worse than their plain supervised base learners. In these cases, the augmented framework can often yield a remedy. In the following, we closely examine the behavior of the semi-supervised learners in various settings.

4.4.1 Datasets and Experimental Setups

As previously, we use the enhanced version of the rainbow system to run the experiments on the three text corpora.[81] We employ the experimental setups as set out in Section 276 (pp. 82 ff.). This time, however, we make use of the unlabeled sets held aside. In particular, for the 20 Newsgroups dataset, there are 10,000 unlabeled documents which may be used during training in addition to the labeled training documents. For the WebKB dataset, there is a total of 2,500 additional unlabeled documents, while there are 700 unlabeled documents available for the TREC dataset.

When using both labeled and unlabeled documents, we have to clarify how the additional use of unlabeled data affects the techniques applied during the text representation phase. As we consider both labeled and unlabeled data as training examples, any of these techniques must take both types of training data into account. In particular, term generation and dimensionality reduction necessitate special processing of training data. Term generation amounts to identifying index terms in all the training documents whether labeled or unlabeled. As it is carried out independent of the actual class of a document, term generation is not a problem. Yet, with respect to dimensionality reduction, incorporating unlabeled training data may not be feasible as some feature selection techniques such as the frequently used information-gain criterion require that the class labels be known. To be able to use them anyway, computation of class-dependent measures might be restricted to only the labeled data, thereby accepting loss of information inherent to the unlabeled

[81] See Appendixes A and B for more details on the software and the text corpora, respectively.

Learner	20 Newsgroups		WebKB		TREC	
	min	max	min	max	min	max
SPC	21.78%	77.63%	42.43%	81.36%	65.23%	95.14%
NB	20.32%	80.13%	37.34%	83.73%	69.49%	96.20%
SVM	22.18%	78.18%	41.61%	89.20%	64.43%	97.14%

Table 4.2: Minimum accuracy obtained when learning with one labeled document per class and greatest expected accuracy when learning from both labeled and unlabeled documents as if the true class labels for the unlabeled documents were known for all three supervised base learners on each of the three text corpora.

data. Note, however, that class-dependent feature selection tends to be statistically unreliable as we typically consider situations in which labeled data is scarce. Hence, if not stated otherwise, we apply only the rudimentary, class-independent feature selection techniques as set out above: stop-word removal and elimination of words that occur only once in the training data. Any index terms remaining are used as vocabulary.

Before presenting results obtained with the semi-supervised text learners, let us consider what we can expect to achieve at most when learning from unlabeled data in addition to some labeled documents. We know that labeled data is more valuable than unlabeled data in classifier design. So, to determine the greatest achievable performance, we assume that the true class labels of all the unlabeled documents are known. Table 4.2 shows the maximally achievable accuracy of the three base learners on each of the three text corpora. In addition, these results are contrasted with the performance achieved when a minimal training set consisting of only a single labeled document per class is available. Note that these result are obtained without substantial dimensionality reduction. In some cases, we observed, if at all, only marginally enhanced performance when reducing vocabulary size. Only for the WebKB dataset, could accuracy of the naïve Bayes classifier and the single-prototype classifier be enhanced substantially by as much as 10% for certain amounts of labeled training data when using only some top hundred features selected according to the information-gain criterion.

4.4.2 Results

There is an abundance of settings for which semi-supervised learning might be evaluated. We will consider some variations along the dimensions shown in Table 4.3.

Dimension	**Range**
Datasets	20 Newsgroups, WebKB, TREC
Base learning algorithms	SPC, NB, SVM
Labeled training set size	*various numbers*
Unlabeled training set size	*various numbers*
Vocabulary size	*various numbers*
Number of bootstrapping iterations	*various numbers*

Table 4.3: Dimensions and their variations along which experiments are set up.

The behavior of the semi-supervised text learning approaches is examined as follows: First, we present results for the three instantiations of the semi-supervised text learning framework in its basic form on the three text corpora selected when the number of labeled training data is varied. Then, we provide additional results to analyze operating characteristics and to identify drawbacks, enabling us to understand when learning from both labeled and unlabeled documents helps and when it degrades performance. In particular, we analyze the effect of the number of unlabeled documents and the effect of the vocabulary size. Also, performance as a function of the number of iterations will be studied. A comparison between the performance on the unlabeled training data and test data sheds some light on the issue of overfitting. Having studied the basic semi-supervised framework, we turn to its augmented variant as a remedy for some cases of performance degradation. Finally, we compare our semi-supervised learners to self-training approaches.

Results for Basic Semi-Supervised Text Learning Framework

Semi-Supervised Single-Prototype and Naïve Bayes Classifiers. Figure 4.4 shows the classification accuracies of the semi-supervised single-prototype classifier (ssSPC) and the semi-supervised naïve Bayes classifier (ssNB) on the three text corpora selected when the number of labeled training documents is varied.[82] Note that the semi-supervised learners have access to the set of unlabeled documents held out. The results are contrasted with the learning curves of the corresponding supervised learners, which learn from only the labeled training documents, as described in Section 276. The horizontal axes indicate the number of labeled training documents on a log scale. Note, for instance, that a total of 20 training documents for the 20 Newsgroups dataset corresponds to one document per class and a total of five training documents corresponds to one document per class for the TREC dataset. For the WebKB dataset, a total of four training documents corresponds to one document per class. The vertical axes indicate the average classification accuracies on the test sets. Note the different magnifications of the vertical scales.

In all experiments, the semi-supervised learners perform substantially better than their supervised counterparts when the number of labeled training documents is small. Or, put in other words, semi-supervised learning can achieve a specific level of classification accuracy with much less labeled training data. For instance, with only 100 labeled training examples for the 20 Newsgroups dataset (five documents per class), ssSPC reaches 67% classification accuracy, while the traditional single-prototype classifier (SPC) achieves only 42%. This represents a 43% reduction in classification error. For the same labeled training set size, accuracy of the naïve Bayes classifier increases from 37% to 56%, representing a 30% reduction in classification error. In other words, to reach 65% classification accuracy, for example, SPC requires about 500 and ssSPC only 70 labeled training documents. Similarly, NB requires about 900 and ssNB only 250 labeled training documents to yield 65% classification accuracy. For the WebKB dataset, the performance increase is smaller but still substantial. For instance, for 20 labeled training examples (five documents per class), SPC obtains 56% accuracy and ssSPC 70%, reducing classification error

[82] The results presented for the semi-supervised naïve Bayes classifier on the 20 Newsgroups dataset have been replicated from Nigam *et al.* (2000).

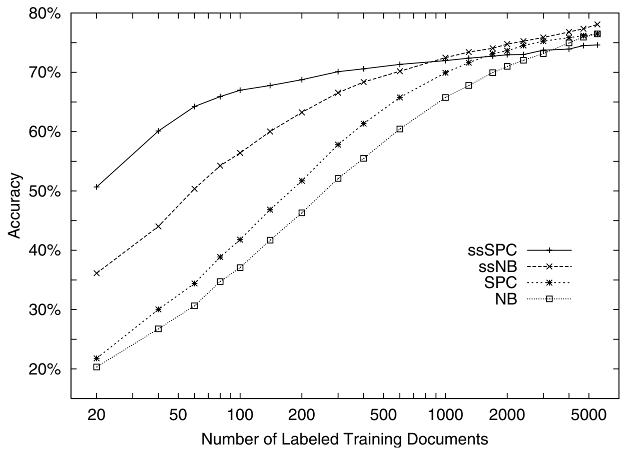
by 32%. For the same number of labeled documents, NB achieves 50% accuracy and ssNB 61%, which represents a 22% reduction in classification error. Note that ssNB can reach about the same performance as ssSPC if the vocabulary is reduced drastically. The effect of the vocabulary size on classification performance will be evaluated below. On the TREC dataset, even though augmented with much less unlabeled data, both ssSPC and ssNB substantially enhance accuracy when labeled data is scarce. The learning curves of both semi-supervised learners are very similar. Yet, since SPC is outperformed by NB, the performance increase of ssSPC is more remarkable than that of ssNB. For instance, with 10 labeled training documents (two documents per class), ssSPC increases accuracy from 76% to 90%, representing a 61% reduction in classification error. In contrast, ssNB increases accuracy from 79% to 89%, representing a 50% reduction in classification error. Altogether, ssSPC is superior to ssNB when the amount of labeled training data is small.

The performance gain achieved by the semi-supervised learners decreases as the number of labeled training documents increases. The reason for this is that more accurate classifiers can be learned from the labeled data alone. As the accuracy obtained through plain supervised learning approaches a dataset-specific plateau, we barely benefit from incorporating unlabeled documents through semi-supervised learning. In fact, note that the accuracy of ssSPC also degrades when the number of labeled training documents is very large. For instance, with 400 labeled training examples (100 documents per class) on the WebKB dataset, classification accuracy decreases from 79% to 76%, representing a 10% increase in classification error. Finally, note that other experiments also show performance degradation for ssNB when using different vocabulary sizes.[83]

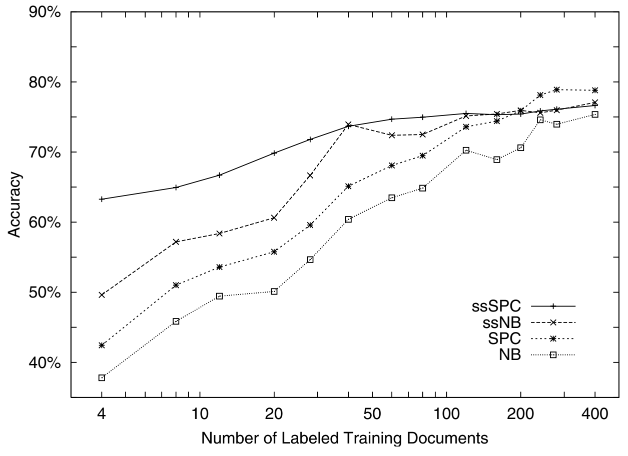
To summarize, the results for the semi-supervised single-prototype classifier and the semi-supervised naïve Bayes classifier show that the benefit we may achieve from the use of unlabeled documents strongly depends on the number of labeled training documents. Of course, performance of semi-supervised learning also crucially depends on the number of unlabeled documents as we will see below. Obviously, we can enhance learning accuracy as the number of labeled training documents increases. So, when the number of labeled training documents is small, the learning algorithm is badly in need for help. Hence, the learner benefits from the additional unlabeled documents even though their imputed class labels are uncertain. Note that the imputed labels are uncertain because many of them tend to be incorrect as they are predicted by a typically weak classifier and, therefore, may not match a user's judgement. With a large labeled training set, however, it is the uncertainty which is inherent to the imputed class labels of the unlabeled documents that may cause performance degradation in some cases. To better illustrate this, assume a labeled training set in which the class labels of a certain fraction of the examples have been altered at random, so that it contains some errors. Then, a classifier learned from the altered training set is typically less accurate than a classifier learned from the original training set.

Semi-Supervised Support Vector Machine. Figure 4.5 shows the classification accuracies of the semi-supervised linear support vector machine (ssSVM) on the 20 News-

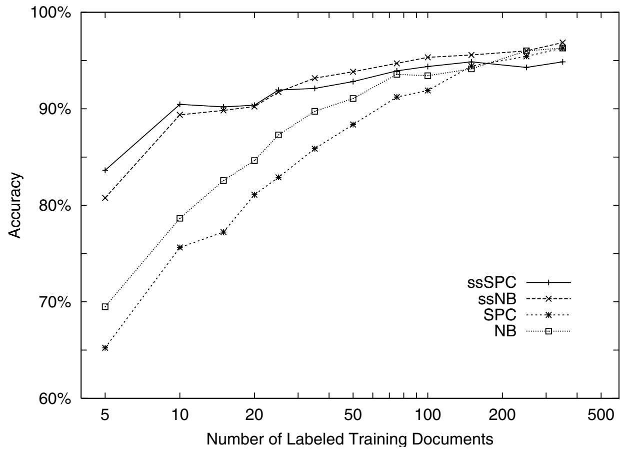
[83]See Nigam *et al.* (2000).



(a) 20 Newsgroups dataset

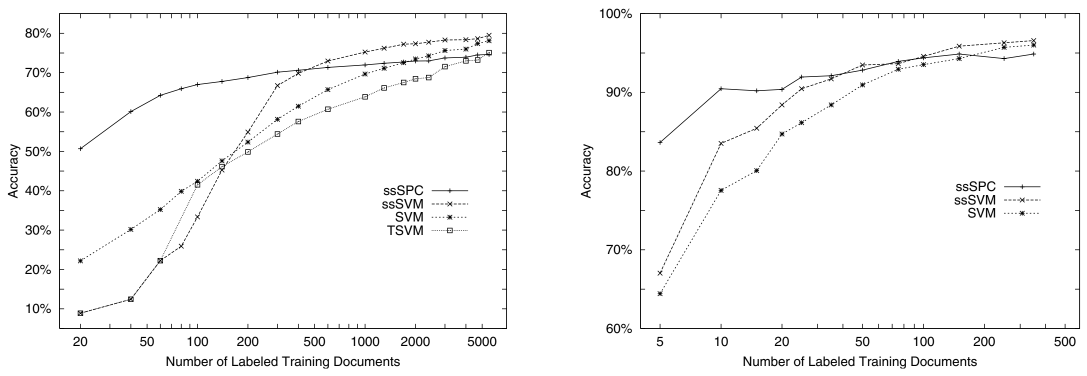


(b) WebKB dataset



(c) TREC dataset

Figure 4.4: Results for the basic semi-supervised learning framework instantiated with the single-prototype classifier (ssSPC) and the naïve Bayes classifier (ssNB) compared to their supervised variants on three common text corpora. The horizontal axes indicate the number of labeled training documents on a log scale. The vertical axes indicate the average classification accuracies on the test sets. Note the different magnifications of the vertical scales.



(a) 20 Newsgroups dataset (b) TREC dataset

Figure 4.5: Classification accuracy of the semi-supervised learning framework based on the linear support vector machine (ssSVM) in comparison to its plain supervised counterpart (SVM), a transductive linear support vector machine (TSVM), and the semi-supervised single-prototype classifier (ssSPC) on the 20 Newsgroups dataset and the TREC dataset. Note the different magnifications of the vertical scales.

groups dataset and the TREC dataset when the number of labeled training documents is varied. The results are compared to the plain linear support vector machine (SVM) and the top-performing semi-supervised single-prototype classifier. Results of the transductive linear support vector machine are also depicted for the 20 Newsgroups dataset.[84]

The performance of ssSVM differs from that of ssSPC and ssNB. In particular, when the number of labeled training documents for the 20 Newsgroups dataset is small, adding unlabeled documents degrades performance substantially compared to plain SVM. With the addition of more labeled data, however, ssSVM starts to outperform SVM and even ssSPC. Though at a different level, the results obtained on the WebKB dataset (not depicted) are similar to those on the 20 Newsgroups dataset. The transductive support vector machine cannot compete at all: it starts as badly as ssSVM and never does perform better than plain SVM. This finding is, in fact, in line with theoretical considerations.[85] Nevertheless, note that the transductive SVM is actually designed to enhance performance on the unlabeled data proper rather than on new, previously unseen documents. And, in fact, further experiments have shown that TSVM outperforms SVM on the unlabeled dataset in the presence of many labeled training examples. On the TREC dataset, results are different: at all times, ssSVM outperforms SVM by a few points. Yet, the benefit obtained through incorporating unlabeled data is rather modest when compared to ssSPC and ssNB. For instance, with 10 labeled training examples (two documents per class), ssSVM obtains 84% accuracy and SVM obtains 78%, representing a 27% reduction in classification error.

As a result, we see that ssSVM seems to benefit from unlabeled data only if the accuracy of the initial classifier learned from only the labeled data is already high. On the TREC dataset, SVM achieves reasonably high classification accuracy with only a few labeled training documents. On the 20 Newsgroups dataset (and also on the WebKB dataset), in contrast, classification accuracy achieved with only a few labeled training documents is

[84]See Section 289 (p. 91).

[85]Zhang and Oles (2000) show that support vector machines in their current form are unlikely to benefit from the use of additional unlabeled data; also see Section 289 (p. 91).

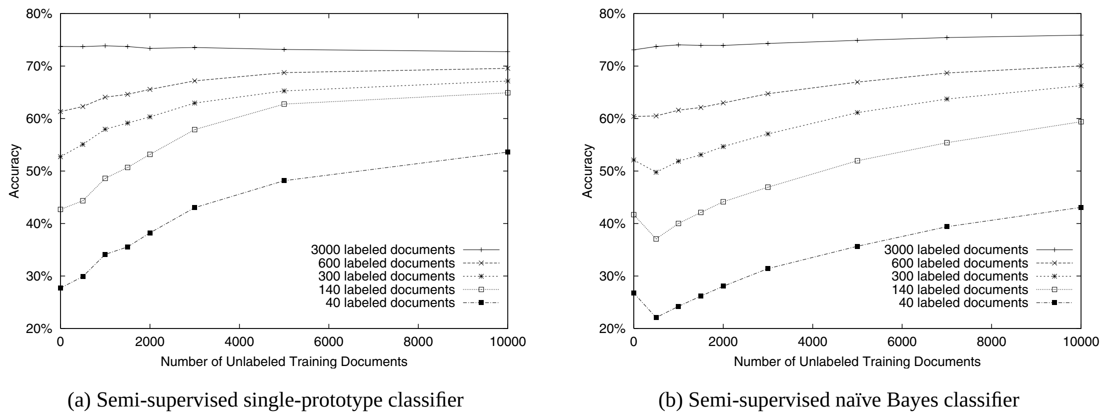


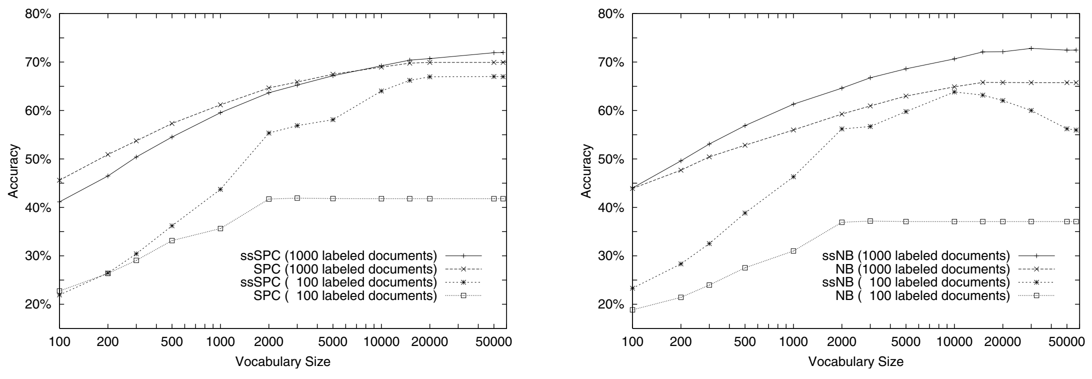
Figure 4.6: Classification accuracies of the semi-supervised single-prototype classifier (ssSPC) and the semi-supervised naïve Bayes classifier (ssNB) with five different numbers of labeled training documents on the 20 Newsgroups dataset when the number of unlabeled training documents is varied. The results for ssNB have been replicated from Nigam *et al.* (2000). Note that ssSPC is superior to ssNB when the number of labeled documents is small. In particular, accuracy of ssNB even degrades if the amount of unlabeled data added is too small.

rather low; see Figure 4.5. Despite initially low accuracy on the unlabeled documents, both ssSPC and ssNB could benefit from unlabeled data. So, how do the single-prototype classifier and the naïve Bayes classifier differ from support vector machines? Both single-prototype and naïve Bayes are instance-averaging classifiers, i.e. each class is modeled through aggregation of class-specific information from the training data. In contrast, support vector machines try to find separating hyperplanes with maximal margin between classes. This involves a complex search which is susceptible to incorrectly labeled training examples. And, as we have seen above, the additional set of unlabeled training examples in combination with their imputed class labels contain many incorrectly labeled training examples. Although ssSVM does indeed benefit from unlabeled data in some situations, it typically struggles when labeled training data is scarce. For this reason, we do not consider support vector machines in the following.

Effect of Unlabeled Set Size

Above, we have seen that the extent to which we may benefit from unlabeled documents depends on the number of labeled training documents available. Naturally, our benefit will also depend on the number of unlabeled documents. To get an impression of this behavior, we now examine the effect of the unlabeled set size. Figure 4.6 shows the classification accuracies of ssSPC and ssNB with five different numbers of labeled training documents on the 20 Newsgroups dataset when the number of unlabeled documents is varied. In most cases, ssSPC is superior to ssNB.

As observed above, adding unlabeled data often helps learning more effective classifiers. Generally, performance gain increases as the amount of labeled data decreases. Also, performance gain increases with the number of unlabeled documents until it reaches a plateau. Only when the amount of labeled data is large may performance degrade. This becomes particularly apparent for ssSPC. Obviously, incorporating a larger set of unlabeled documents amounts to a larger number of incorrectly labeled examples, which



(a) Semi-supervised single-prototype classifier (b) Semi-supervised naïve Bayes classifier

Figure 4.7: Classification accuracies of the semi-supervised and plain single-prototype classifier and naïve Bayes classifier for the 20 Newsgroups dataset when vocabulary size is varied for two amounts of labeled training documents. Performance gain of semi-supervised learning strongly depends on the number of features used to represent text. In particular, performance increases with the number of features. Yet, note that performance of ssNB may degrade when using too large a feature set.

may deteriorate ssSPC's class prototypes. As observed by Nigam *et al.* (2000) for ssNB, adding a small number of unlabeled documents to a small amount of labeled documents actually hurts classification performance. Nigam *et al.* hypothesize that the reason for this is the overly confident estimates of the posterior probabilities of the classes given a document, which effect an almost crisp partition in the unlabeled documents.[86] Also, they argue that this problem disappears when the number of unlabeled documents increases because the unlabeled set then provides a large enough sample to smooth out the sharp discreteness in the predicted class labels. Nevertheless, note that crisp class labels do not keep ssSPC from enhancing performance even when both labeled and unlabeled training set sizes are small.

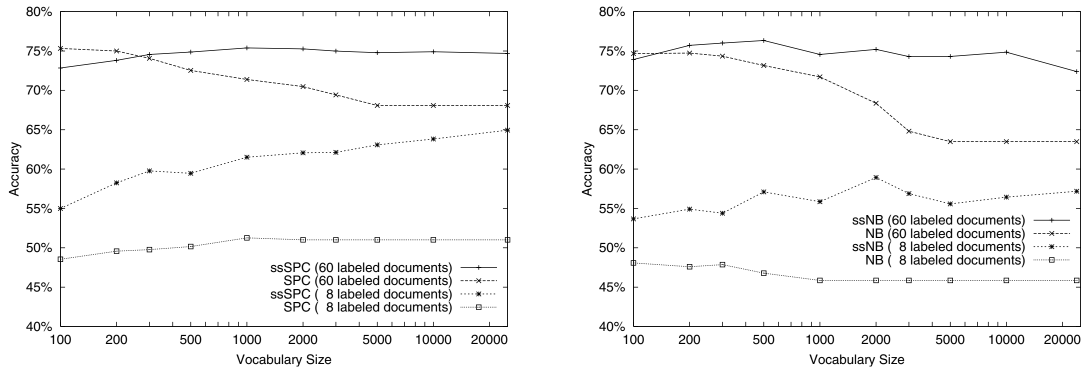
Effect of Vocabulary Size

In Section 4.1.3 we have argued that classification accuracy may be enhanced through exploiting co-occurrence patterns among index terms in unlabeled data. Hence, performance of semi-supervised learning approaches should depend on the number of features used to represent text. In the following we examine classification accuracy of ssSPC and ssNB on the 20 Newsgroups dataset and the WebKB dataset as a function of the number of features.

Smaller vocabulary sizes are obtained by selecting the particular numbers of the most informative features according to the information-gain criterion.[87] Note that this feature selection technique is class-specific and can, therefore, only be applied to those features that occur in the labeled training data. Features occurring only in the unlabeled data are selected in an arbitrary order, which is greatly determined by the sequence in which they have been generated from the training data. Applying more elaborate selection techniques for these features should be an issue for future research.

[86]Due to the word independence assumption, the probabilistic class labels obtained by the naïve Bayes classifier are often extremely close to zero and one; see Section 213 (p. 63 ff.).

[87]See Section 123 (p. 40) for a description of the information-gain criterion.

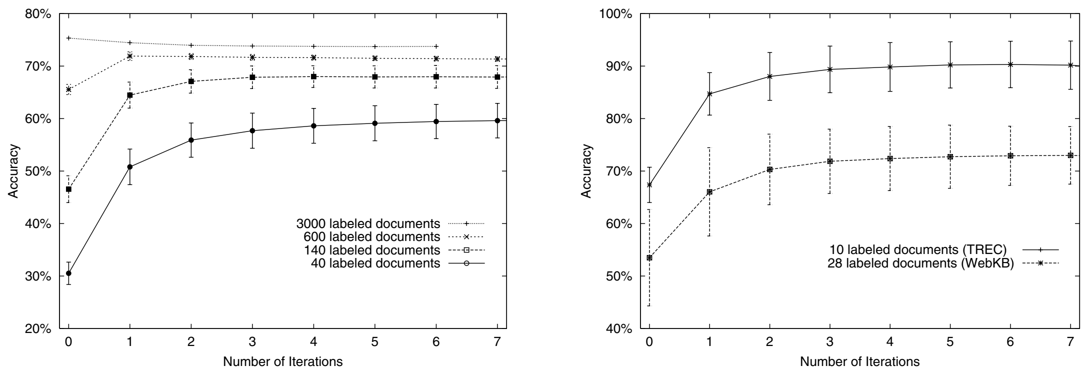


(a) Semi-supervised single-prototype classifier (b) Semi-supervised naïve Bayes classifier

Figure 4.8: Classification accuracies of the semi-supervised and plain single-prototype classifier and naïve Bayes classifier for the WebKB dataset when vocabulary size is varied for two amounts of labeled training documents. Typically, performance gain of semi-supervised learning increases with vocabulary size. Yet, when the base learner is more effective at smaller vocabulary sizes, incorporating unlabeled data does not help a lot. We hypothesize that the benefit from using unlabeled data that might be achieved only with a reasonably large vocabulary set is canceled out by the performance degradation caused by using too large a feature set.

Figure 4.7 shows learning curves of ssSPC and ssNB and their supervised variants on the 20 Newsgroups dataset with two fixed labeled training set sizes when the number of features is varied from 100 to the maximum of 57,944. The learning curves illustrate that performance gain depends on vocabulary size: as a rule, performance gain increases with the number of features. When using too few features, semi-supervised learning hardly benefits from unlabeled data. In particular, for ssSPC with 1000 labeled training documents, incorporating unlabeled documents may not help a lot; the learning curves of SPC and ssSPC are very similar. Nevertheless, ssSPC improves slightly as vocabulary size increases. Performance gain of ssNB over NB is somewhat larger and also increases with the number of features. In contrast, when using only 100 labeled training documents, both ssSPC and ssNB may substantially outperform their supervised variants when the number of features is large enough. Note that performance of SPC and NB reaches a plateau when using 2000 or more features with 100 labeled training documents. The reason for this is that there are, on average, about 2000 distinct features in the labeled documents. Having access to unlabeled documents, however, the number of possible index terms is much larger. Again, performance gain increases with the number of features. Yet, note that performance of ssNB degrades when using too large a feature set. At this point, employment of class-independent feature selection techniques should be considered.

The operating characteristics of semi-supervised and plain single-prototype classifiers and naïve Bayes classifiers are different on the WebKB dataset; see Figure 4.8. The learning curves obtained with eight labeled training documents when the number of features is varied from 100 to the maximum of 23,830 again show that performance gain typically increases with the number of features. With a small amount of training data available for training, performance of SPC and NB does not vary greatly with the number of features. In contrast, when more labeled training data is available, performance of the semi-supervised learners does not vary greatly with vocabulary size. This time, however, performance of SPC and NB decreases substantially as the number of features increases. We hypothesize that the benefit from using unlabeled data which is achievable with a rea-



(a) 20 Newsgroups dataset (b) TREC dataset and WebKB dataset

Figure 4.9: Classification accuracies of the single-prototype classifier as a function of the number of bootstrapping iterations with four different numbers of labeled training documents for the 20 Newsgroups dataset and with one fixed labeled training set each for the TREC dataset and the WebKB dataset. Performance at the first seven iterations after learning the initial classifier is reported. Note that at iteration zero, performance corresponds to that of the plain single-prototype classifier. In addition, the standard error is shown for some learning curves, indicating that, at the worst, dispersion increases slightly.

sonably large vocabulary set is canceled out by the performance degradation caused by using too large a feature set. So, ssSPC and ssNB actually outperform their supervised variants when text is represented through a large number of features. Yet, compared to the performance obtained by SPC and NB when using a more appropriate vocabulary size, ssSPC and ssNB yield hardly any performance gain.

To summarize, results show that performance of semi-supervised learning greatly depends on vocabulary size. Typically, performance gain over plain supervised learners increases with the number of features. Yet, we may only benefit from this performance gain if it is not blotted out by a performance degradation caused by using too large a feature set. As a consequence, semi-supervised learning is especially suitable for datasets on which performance of plain supervised learners does not degrade substantially as vocabulary size increases.

Effect of Number of Iterations and Overfitting

The results presented so far show that semi-supervised learning may help to reduce the need for unlabeled documents. But to what extent does clustering in the unlabeled data contribute to this? Put in other words: Does performance gain increase with the number of iterations? Related to this is the issue of overfitting. Does the semi-supervised learner generalize well beyond the training data or does it only fit the labeled and unlabeled training data well? To find answers to these questions, we now analyze the performance of ssSPC as a function of the number of iterations carried out in the semi-supervised framework. In addition, we compare performance on the unlabeled set and the test set.

Figure 4.9 shows the classification accuracies of ssSPC on all three text corpora at the first seven bootstrapping iterations after the initial classifier has been learned from only the labeled data at iteration zero. For the 20 Newsgroups dataset, learning curves with four different amounts of labeled training data are plotted. For both the TREC dataset and the WebKB dataset, learning curves with only one fixed amount of labeled training

$\frac{n_l}{k}$	**Learner**	20 Newsgroups		WebKB		TREC	
		unlabeled	test	unlabeled	test	unlabeled	test
1	SPC	23.43%	21.78%	43.59%	42.43%	61.28%	65.23%
	ssSPC	51.38%	50.67%	65.59%	63.27%	84.25%	83.63%
10	SPC	55.07%	51.72%	65.59%	65.12%	87.41%	88.37%
	ssSPC	71.02%	68.75%	77.61%	73.67%	92.21%	92.82%
50	SPC	74.21%	69.92%	80.36%	75.89%	94.71%	95.43%
	ssSPC	76.21%	71.97%	80.57%	75.42%	94.86%	94.29%

Table 4.4: Comparison of classification accuracies of both the semi-supervised and the plain supervised single-prototype classifier on the unlabeled set and the test set for all three text corpora with three different numbers of labeled training documents per class, $\frac{n_l}{k}$. Results show that performance on both unlabeled data and test data is very similar, indicating that there is no overfitting problem.

data each are shown. When the number of labeled documents is small, accuracy increases substantially. The gain achieved is largest at the first iteration, i.e. when unlabeled data is incorporated for the first time. Yet, performance is further enhanced at subsequent iterations. So, we see that clustering in the unlabeled data does, in fact, support classifier design. In addition, note that the standard error depicted for some learning curves indicates that, at the worst, dispersion increases slightly compared to the plain supervised learner. As a consequence, variation in the averages reported for the semi-supervised learners is roughly within the same bounds as that for the supervised approaches.

As shown only for the 20 Newsgroups dataset—but as applies likewise for the other two datasets—performance gain tends to be limited to the first bootstrapping iteration when the number of labeled training documents becomes larger. Obviously, with an increasing amount of labeled training data, the initial classifier itself becomes capable of producing a reasonable partition of the unlabeled data. Finally, when the number of labeled documents is large enough to accurately learn a classifier, we have already seen that performance may degrade when incorporating unlabeled data. The learning curve obtained with 3000 labeled documents for the 20 Newsgroups dataset illustrates this effect again. Note that performance degradation increases slightly with the number of bootstrapping iterations, showing that incorrectly labeled examples may yield further misclassifications at subsequent iterations just as correctly labeled examples may enhance performance.

As they do not perform a proper search in hypothesis space but rather aggregate class-specific information, both the single-prototype classifier and the naïve Bayes classifier are not prone to overfitting. To validate their generalization ability anyhow, Table 4.4 shows classification accuracies of ssSPC and SPC for three fixed numbers of labeled training documents per class on the unlabeled training set and the independent test set for all three text corpora. In general, performance on both unlabeled and test data is very similar, indicating that the issue of overfitting does not pose a problem for ssSPC.

Note that plain SPC does not make use of the unlabeled data during the learning process. So, the unlabeled set may be regarded as another independent test set. Nevertheless, note the specific relationship between the labeled training set and the unlabeled set for the 20 Newsgroups dataset and the WebKB dataset. In particular, as the test set for the 20 Newsgroups dataset is created by selecting the last 20% of the articles from each newsgroup by posting date, unlabeled documents are closer in time to the labeled training documents than the test documents. For the WebKB dataset, the four test sets are created in leave-one-university-out fashion. So, while each test set is formed on the basis of web pages from one of the four universities, the corresponding labeled and unlabeled sets are created from web pages of the remaining three universities. As a consequence, unlabeled documents for the WebKB dataset tend to be more similar to the labeled documents than the test documents. For the TREC dataset, there are no such relationships between the labeled and unlabeled training documents.

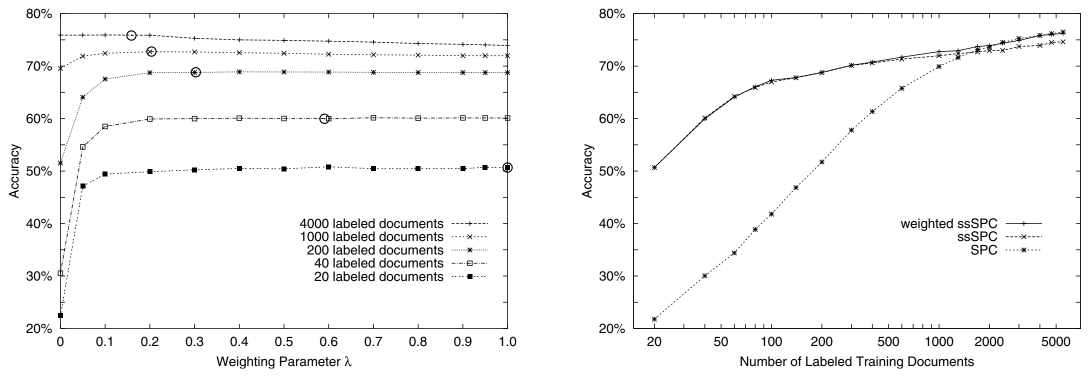
The aforesaid relationships between the labeled and unlabeled sets reveal themselves in the results shown in Table 4.4: for the 20 Newsgroups dataset and the WebKB dataset, accuracy on the unlabeled sets is on average a few points higher than accuracy on the test sets. For the TREC dataset, accuracies on both unlabeled and test sets are very similar; only for those experiments with one labeled training document per class is accuracy on the test set higher. The reason for this might be the slightly different fractions of documents within the classes, which becomes particularly apparent when the number of labeled training documents is small. In spite of these differences, the results show that performance on both unlabeled sets and test sets is very similar. Hence, semi-supervised learning does not overfit the unlabeled training data. Rather, it generalizes well beyond the training data and yields comparable performance on independent test data.

Results for Augmented Framework

Above, we have provided evidence that learning from both labeled and unlabeled data may enhance classification performance substantially when the amount of labeled training data is small. Only when the amount of labeled data becomes too large may performance degrade. As we have discussed earlier, it is unlikely that we will be provided with enough labeled data to actually observe this behavior in practice. So, this type of performance degradation should not pose a serious problem. Still, we typically do not know just when a training set is deemed to be large enough. In the following, we describe results pertaining to the augmented semi-supervised framework which deal with this problem.[88] We instantiate the augmented framework with the single-prototype classifier. The resulting weighted semi-supervised learner, which we refer to as *weighted ssSPC*, is then applied to the 20 Newsgroups dataset.

Figure 4.10 (left) shows the classification accuracy of weighted ssSPC when the weighting parameter λ is varied within its range $[0, 1]$. Recall that λ controls the relative weight of the unlabeled data with respect to the labeled data. So, weighted ssSPC corresponds to SPC and basic ssSPC for the extreme values $\lambda = 0$ and $\lambda = 1$, respectively. The learning curves show that weighted ssSPC is not very sensitive to variations of the weighting parameter. Only values close to zero hinder weighted ssSPC from taking advantage of unlabeled data when labeled data is scarce. Any other value is sufficient to yield substantial performance gain just like plain ssSPC. More interesting is the situation in which the labeled set becomes larger and performance degradation is observed for ssSPC. For example, this is so when learning from unlabeled data in addition to 4,000 labeled training

[88]See Section 4.3.4 (pp. 112 ff.).



(a) Varying weighting parameter λ (b) Varying number of labeled training documents

Figure 4.10: Classification accuracies of weighted ssSPC on the 20 Newsgroups dataset when the weighting parameter λ is varied for five amounts of labeled training data (left). The circles mark the values of λ determined heuristically as a function of the average number of labeled training documents per class as introduced earlier in this chapter. Also, classification accuracy of weighted ssSPC using the heuristically determined weighting parameter λ is compared to basic ssSPC and SPC on the 20 Newsgroups dataset when the number of labeled training documents is varied. Results illustrate that weighted ssSPC mitigates the drawback of ssSPC when learning from many labeled training documents: compared to SPC, performance does not degrade any more.

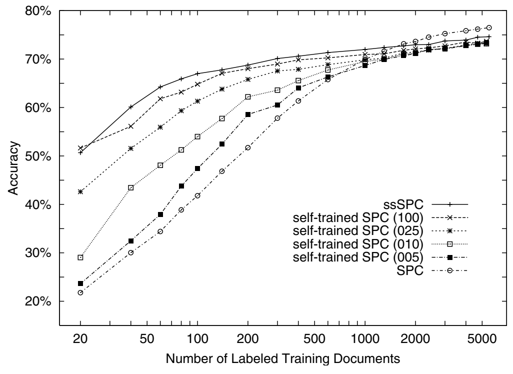
documents. The corresponding learning curve illustrates that performance degradation can be mitigated when given less weight to the unlabeled data. A central problem is how to set the weighting parameter λ. We heuristically determine the weighting parameter as a function of the average number of labeled documents per class available for training as set out in Section 4.3.4. The circles in Figure 4.10 (left) mark the heuristically determined values for λ. Note that all values lie in regions with performance close to the optimum.

Figure 4.10 (right) shows the classification accuracy of weighted ssSPC with heuristically set weighting parameter when the number of labeled training documents is varied. Results are contrasted against plain ssSPC and traditional SPC. When labeled data is scarce, weighted ssSPC shows the same superior results as plain ssSPC. Yet, as the number of labeled documents increases, performance does not degrade. Rather, weighted ssSPC can compete with traditional SPC. Nigam *et al.* (2000) have successfully applied weighted ssNB on the WebKB dataset with weighting parameters determined on the basis of leave-one-out cross-validation. Results with both weighted ssSPC and weighted ssNB show that adjusting the relative weight of unlabeled data with respect to the labeled data allows mitigating of the aforementioned performance degradation.

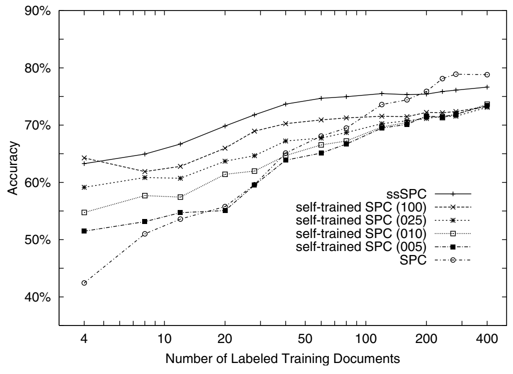
Comparison to Self-Training

So far, we have studied the behavior of the three instantiations of our semi-supervised learning framework and found that it outperforms the corresponding supervised learners in many cases. But how does semi-supervised learning compare with other approaches to bootstrapping classifiers? As has been shown in Section 4.1.4, a direct competitor of semi-supervised learning is self-training.[89] In the following we review the idea of self-training and contrast it with semi-supervised learning.

[89]See Nigam and Ghani (2000), for example.



(a) 20 Newsgroups dataset



(b) WebKB dataset

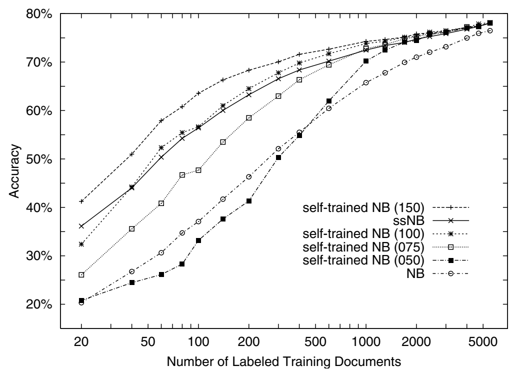
Figure 4.11: Classification accuracies of self-training applied to the single-prototype classifier for the 20 Newsgroups dataset and the WebKB dataset with different number of examples per class added to the training set at each iteration. The results are contrasted with those of the semi-supervised single-prototype classifier and the plain single-prototype classifier.

Self-training is the most straightforward bootstrapping approach to learning from labeled and unlabeled data. Starting with an initial classifier learned from only the labeled data, self-training iterates two steps until all of the unlabeled documents have been converted into training documents. First, the current classifier is used to classify all the unlabeled documents which have not yet been put into the training set. Then, for each class, the top α most confidently classified documents are converted into training documents.[90] As experiments will show, the performance of self-training largely depends on α.

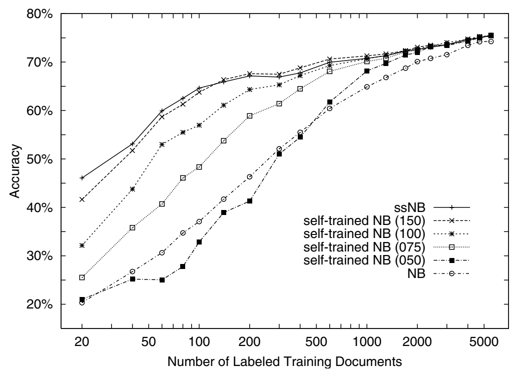
Note that once an unlabeled document is put in the training set for the self-training approach, its imputed class label is not allowed to change in subsequent iterations. In contrast, a semi-supervised learner as set out above uses, at each iteration, all unlabeled documents for which class labels have been imputed, but it allows relabeling of unlabeled documents in subsequent iterations. In this way, both approaches provide a means to avoid misclassifying unlabeled documents: self-training through choosing only the most confidently classified documents and semi-supervised learning by allowing relabeling. So, self-training and semi-supervised training can be considered as opposite ends of a whole spectrum of choices of how to relabel and incorporate unlabeled data.

Figure 4.11 shows classification accuracies of the self-trained single-prototype classifiers with different values of the parameter α on the 20 Newsgroups dataset and the WebKB dataset when the number of labeled training documents is varied. The results are contrasted with ssSPC and SPC. For the 20 Newsgroups dataset, the learning curves show that self-trained SPC achieves about the same performance as ssSPC when α is chosen optimally. With sub-optimal choices, ssSPC clearly outperforms self-training. For the WebKB dataset, self-trained SPC performs worse than ssSPC in all experiments but one. Again, if α is chosen sub-optimal, performance of self-trained SPC may be as bad as SPC. In any way, the results reveal the advantage of semi-supervised learning over self-training: performance is competitive without parameter tuning.

[90]Instead, we might use class-specific parameters α_i, for each $c_i \in \mathcal{C}$, where each α_i is a multiple of the respective class prior. Yet, we refrain from doing so because class prior estimates tend to be unreliable when labeled data is scarce.



(a) 20 Newsgroups dataset (all features)



(b) 20 Newsgroups dataset (10000 features)

Figure 4.12: Classification accuracies of self-training applied to the naïve Bayes classifier for the 20 Newsgroups dataset using all features and the 10000 most informative features according to the information-gain criterion. As above, different numbers of examples per class are added to the training set at each iteration. The results are contrasted with those of the semi-supervised naïve Bayes classifier and the plain naïve Bayes classifier.

Experiments with the naïve Bayes classifier on the 20 Newsgroups dataset show that self-trained NB can outperform ssNB and NB when α is set to an optimal value; see Figure 4.12.[91] The reason for this behavior—in contrast to self-trained SPC and ssSPC—is that self-trained NB is less sensitive to using too large a feature set than ssNB. In particular, self-trained NB incrementally increases the training set which effects a gradual increase in the number of features. In contrast, ssNB is confronted with the complete set of features at any bootstrapping iteration. So, self-trained NB learns with fewer features on average and may therefore produce more accurate classifiers. Note that ssNB is superior to self-trained NB when reducing vocabulary size to 10,000 index terms.[92]

4.5 Conclusions

In this chapter, we have presented a general framework for semi-supervised learning from labeled and unlabeled documents. This is a crucial issue when hand-labeling documents is expensive, but unlabeled documents are readily available in large quantities, as is often the case for text classification tasks. The following summarizes the results of the empirical evaluation:

- Semi-supervised learning approaches can be used to learn accurate classifiers from a large set of unlabeled data in addition to a small set of labeled training documents. Some semi-supervised learners substantially outperform their supervised variants; in other words, they often require less labeled training data to achieve the same level of classification accuracy.

- Instance-averaging approaches such as the single-prototype classifier and the naïve Bayes classifier are particularly suitable as base learners in the semi-supervised learning framework since they are extremely robust to incorrectly labeled data.

[91] Similar results are reported by Nigam and Ghani (2000).

[92] In this case, performance of ssNB does not degrade because too many features are used; see Figure 4.7.

- In addition, more elaborate learning algorithms such as support vector machines may also benefit from the use of unlabeled data. Yet, performance gain shows only when the initial performance of the base learner applied to the labeled data alone is reasonably high. The reason for this is perhaps the complex search involved, which makes support vector machines more susceptible to incorrectly labeled data than the simple instance-averaging approaches.

- Typically, instance-storing approaches such as nearest-neighbor rules do not benefit from semi-supervised learning framework because they do not generalize beyond the training data and, thus, cannot exploit co-occurrence patterns in unlabeled data.

- Some semi-supervised approaches show performance degradation when labeled training data becomes abundant. This drawback can be mitigated by the augmented framework, which allows the weight of the unlabeled data to be discounted.

- Semi-supervised learning is particularly suitable for learning tasks in which larger feature sets do not hinder a learning algorithm from accurately learning a classifier. If too large a feature set causes substantial performance degradation, incorporating unlabeled data may be less useful.

- The semi-supervised framework shows up favorably compared to self-training approaches with the same base learner. In particular, semi-supervised learners achieve similar or even better results without critical parameter tuning.

The semi-supervised learning framework yields to further experimentation. For example, some relevant issues for future research are:

- So far, we have considered the single-prototype classifier, the naïve Bayes classifier, support vector machines, and nearest neighbor rules as base learners in the semi-supervised learning framework. How do other supervised learners such as decision tree induction algorithms perform in the semi-supervised framework?

- In some cases, we have observed performance degradation from using too large a feature set. Yet, the application of class-specific feature selection techniques like the information-gain criterion, which is often used in text classification, may not be statistically reliable when labeled training data is scarce. Using class-independent techniques, such as selecting features with high signal-to-noise ratio,[93] might help to mitigate this drawback. Although initial experiments haven not yet shown any improvement, class-independent feature selection techniques which are common in unsupervised learning should be evaluated in this context.

- So far we have studied semi-supervised learning and self-training. Both approaches to learning from labeled and unlabeled data can be seen as two extremes of a whole spectrum of approaches to converting unlabeled data into labeled training examples and dealing with incorrectly labeled examples. Now, can we combine ideas

[93]See Section 119 (p. 38).

from these approaches so as to further enhance classification performance? For example, does it help to incrementally put unlabeled data into the training set as in self-training and to permit class labels of formerly unlabeled examples to change in subsequent iterations as in semi-supervised learning? Or, is it better to reject unlabeled documents and exclude them from the training set in the semi-supervised learning framework if their classification confidence is low?

Chapter 5

Learning Heterogeneous Classes

Some of the most frequently and successfully applied text learning algorithms have a common drawback: they require that the classes to be learned be homogeneous. In some text classification tasks like information filtering, however, classes tend to be heterogeneous. The violation of the *homogeneity assumption* may hinder accurate learning of the classes. In this chapter, we empirically evaluate the extent to which heterogenous class definitions affect classification performance of some common text learners.

5.1 Introduction

In this section, we take a close look at the heterogeneous nature of the binary relevance class definition in information filtering, and we discuss why this may pose a problem. Subsequently, we introduce the concept of subclasses which allow division of classes into more homogeneous groups of documents. Based on this, we address approaches aimed at exploiting subclass structure. References to related work will be given in the following section.

5.1.1 Problem Description

Supervised learning is based on the underlying assumption that like examples belong to the same class.[1] Yet the fact that similar examples belong to the same class does not imply that all examples of a particular class are similar. Rather, a class may be heterogeneous, consisting only of groups of similar documents. Although in many standard text classification tasks we may assume that the class structure is defined in such a way that the classes are homogenous, especially in information filtering we often find heterogenous class definitions.

To see this, consider that information filtering involves the separation of relevant from non-relevant documents according to a particular user interest as introduced in Chapter 2.

[1]See Quinlan (1999). In information retrieval, this is often formulated as the *cluster hypothesis*: "closely associated documents tend to be relevant to the same requests." (from van Rijsbergen (1979), p. 45).

Depending on the domain, this implies rather broad class definitions, which are prone to be heterogeneous. For example, when filtering news stories, there is often a great number of different topics which a user might like or dislike, i.e. which are either relevant or non-relevant. Assume that each topic describes a homogeneous subset of documents, namely those covering a particular subject. Obviously, as any user-specific grouping of topics may form the two relevance classes, these are unlikely to be homogeneous.

Why does a heterogeneous class structure hinder some text learners from constructing accurate classifiers? We consider learning simple instance-averaging approaches such as the naïve Bayes classifier and the single-prototype classifier, which represent each class by a single entity. These classifiers are among the most frequently and successfully applied approaches to automating text classification. The reason for this is their simplicity, which makes them particularly well suited for learning from a limited amount of sparse document vectors in high-dimensional feature space. Nevertheless, it is also their simplicity which severely compromises class representations and, thus, requires that the classes to be learned be homogenous.[2]

How do simple instance-averaging approaches perform when classes are heterogeneous? Figure 5.1 illustrates the problem of having heterogeneous classes when using a simple instance-averaging approach. We easily see that a certain—though often unknown—level of granularity is necessary to achieve homogeneous classes. For complex topic structures, the two relevance classes of information filtering will not suffice to form homogeneous groups of documents. If the homogeneity assumption is violated, documents may be difficult to classify.

Note that the relevant class of an information filtering task often consists of a small number of topics, whereas the non-relevant class typically consists of a large variety of topics. So, as the non-relevant class tends to be much more heterogeneous, the learning task may be simplified by modeling only the relevant class explicitly. And, in fact, doing so frequently suffices to solve the underlying binary classification task. As for the issue of quality control in information filtering, however, modeling both the relevant and the non-relevant class explicitly is vital; see Chapter 6. Below, we introduce the concept of subclasses to better describe complex text classification problems.

5.1.2 The Concept of Subclasses

A predefined class may be heterogeneous. Yet, typically such a class consists of homogeneous subclasses. And, within these subclasses, the resemblance between documents is relatively high, whereas it may vary greatly between documents belonging to the same class but to different subclasses.[3]

[2]Other classifiers such as nearest-neighbor rules or decision tree learning algorithms are less restrictive with respect to assumptions pertaining to the class structure and may, therefore, better cope with heterogeneous classes. Nevertheless, in our experiments with homogeneous classes, these classifiers did not show themselves to be superior to the naïve Bayes classifier and the single-prototype classifier. See Section 3.3 for a detailed description of common text learning algorithms.

[3]Also see de Kroon *et al.* (1996) and Hsu and Lang (1999) for the concept of subclasses in information filtering and text classification.

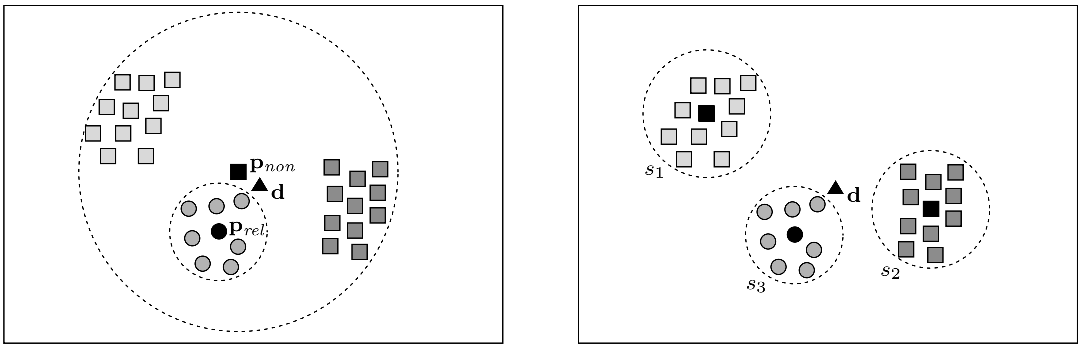


Figure 5.1: When using an instance-averaging approach which represents each class through a single prototype, classifying a new document $\mathbf{d}$ may be difficult (left). Even though $\mathbf{d}$ is more similar to documents of the relevant class (circles), the prototype $\mathbf{p}_{non}$ of the non-relevant class (squares) is closer to $\mathbf{d}$ than the prototype $\mathbf{p}_{rel}$ of the relevant class. Representing homogeneous subclasses through individual prototypes (right) may remedy this problem.

The objective of text classification is to assign documents to their true class as set out in Definition 3.1.1 (p. 23). The introduction of subclasses allows division of classes into more homogeneous groups of documents and provides a more realistic picture of the classification task. Note that the concept of subclasses may be regarded as a type of class hierarchy. Yet, in contrast to exploiting a class hierarchy when learning a classifier as mentioned in Section 260 (p. 75), here we do not assume any topical relationship among subclasses—as a rule, the only common feature among subclasses is that they may belong to the same class. We now extend the definition of text classification so as to include the concept of subclasses:[4]

Definition 5.1.1 (Text Classification with Subclasses)
Assume a space of documents $\mathcal{R}$, a fixed set of k classes $\mathcal{C} = \{c_1, \ldots, c_k\}$, and a fixed set of q subclasses $\mathcal{S} = \{s_1, \ldots, s_q\}$, where $q \geq k$. Both subclasses and classes imply a disjoint, exhaustive partition of $\mathcal{R}$. As above, text classification is a mapping $H : \mathcal{R} \mapsto \mathcal{C}$ from the document space onto the set of classes. The mapping H can be written as the composition of two mappings, $H(\mathbf{d}) = (H_c \circ H_s)(\mathbf{d})$, for $\mathbf{d} \in \mathcal{R}$, where $H_s : \mathcal{R} \mapsto \mathcal{S}$ maps any document onto a subclass and $H_c : \mathcal{S} \mapsto \mathcal{C}$ associates each subclass $s \in \mathcal{S}$ with exactly one class $c \in \mathcal{C}$. Subclasses are assumed to be homogeneous.

Figure 5.2 illustrates the concept of subclasses, dividing the heterogeneous relevance classes into homogeneous groups of documents. Also, it shows the decomposition of the text classification mapping. Note that this concept actually corresponds to the general information filtering framework set out in Section 2.2 (pp. 10 ff.). In this framework, the user interest $\psi \in \mathcal{N}$ is represented by the profile acquisition function $\pi : \mathcal{N} \mapsto \mathcal{P}$, which permits capturing of various topics. A comparison function $\kappa_s(\pi(\psi), \mathbf{d})$ is used to yield a decision vector, which is then mapped by a system decision function τ_s onto zero or unity, representing the non-relevant and relevant classes, respectively. Now, in the notation of Definition 5.1.1, the mapping H_c from the subclasses onto the classes corresponds to the decision function τ_s when assuming a two-class problem, whereas the mapping H_s from

[4]In contrast to Definition 3.1.1, here we use the space of document representations $\mathcal{R}$ instead of the space of actual documents $\mathcal{D}$, i.e. we use the mapping H rather than h. The reason for this is that we assume that documents and their representations can be used interchangeably; see Section 3.2.5 (p. 47).

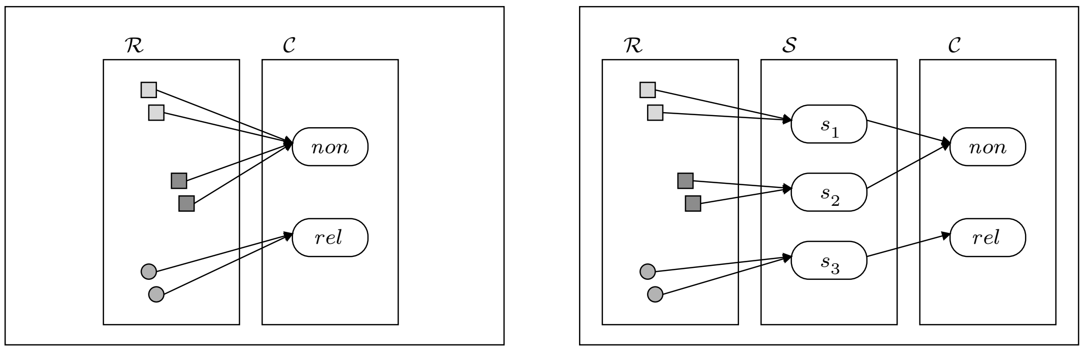


Figure 5.2: The classes defined in information filtering, here rel and non for the relevant and non-relevant documents, are often heterogeneous. So, different kinds of documents from space $\mathcal{R}$ may be mapped onto the same class from space $\mathcal{C}$ (left). Introducing subclasses $s_i \in \mathcal{S}$ allows formation of homogeneous groups, which may then be mapped onto the known classes (right). This reflects the situation of many text classification tasks more realistically.

the document representation space onto the subclasses corresponds to the comparison function κ_s. Note the different meaning of the index s in τ_s and κ_s as opposed to H_s.

Note that in terms of the hypothetical document source set out in Section 3.3.1 (p. 48), the concept of subclasses may be interpreted as follows. Instead of assuming a one-to-one correspondence between classes and class-specific document sources, there may now be multiple class-specific document sources per class. So, the assumed document generation process consists of two steps: first, a generative model would pick any of the classes and, second, any of the respective class-specific document sources, which would then create a particular document.

5.1.3 Exploiting Subclass Structure

How can we exploit knowledge about the homogeneous subclass structure? Possible answers certainly depend on whether we merely know that there are homogeneous subclasses or whether subclass labels are actually provided.

If, on the one hand, subclass labels are given, we may actually consider a classifier which learns the subclasses rather than the classes. When predicting the class label of a new document, the classifier's response at the subclass level is then mapped onto the classes in a separate step. Yet, so far we have seen that hand-labeling documents is tedious and expensive. We argue that this becomes even worse with increasing fineness of the granularity of the predefined classes. Therefore, we assume that the user provides labels only at the class level but not at the subclass level. As a consequence, we will consider supervised learning of the subclasses only as a benchmark approach.

On the other hand, if subclass labels are unknown, a common way of exploiting subclass structure anyhow would be to make use of unsupervised learning approaches to discover subclasses automatically. In contrast to traditional clustering, however, note that we may make use of the given class labels during the clustering process. In particular, the known relevance classes might be used to decide on the fineness of the granularity of the clusters generated, i.e. basically on the number of subclasses to be discovered. References to this kind of approach will be given in the subsequent section.

5.2 Related Work

Several approaches to learning heterogeneous classes through the aid of unsupervised learning can be found in the literature. Some of them have been developed specifically in the domain of text classification, whereas others have a more general character.

Makoto and Takenobu (1995) use hierarchical Bayesian clustering to find clusters which most likely describe a set of documents. These clusters are then used to support text classification tasks. In their information filtering approach, de Kroon *et al.* (1996) also use a Bayesian clustering approach to automatically discover subclasses corresponding to different interests of users. Similarly, Hsu and Lang (1999) use k-means clustering to discover subgroups within documents of the same class. The resulting clusters are used as prototypes for the predefined classes. Merkl (1998) and Klose *et al.* (2000) explore self-organizing maps to find clusters in document collections. These clusters are used to ameliorate document retrieval. However, they might also be used to enhance text classification when the introduction of subclasses helps to form homogeneous classes.

The issue of learning heterogeneous classes is relevant not only in supervised learning but also in a semi-supervised learning framework. In their semi-supervised naïve Bayes approach, Nigam *et al.* (2000) propose that subclasses be identified automatically so as to deal with the situation of having multiple document sources per class, which violates the assumed one-to-one correspondence between document sources and classes.

A different approach is followed by Bensaid and Bezdek (1998). Given some labeled training data in addition to a set of unlabeled data, their semi-supervised clustering approach is designed to purposefully over-partition unlabeled data in as many groups as there are labeled training examples. The idea behind over-partitioning is that it is better to have too many than too few prototypes to represent heterogeneous classes. Through subsequent merging and labeling of cluster representations according to the class labels of the labeled training data, this semi-supervised clustering approach also permits discovery of multiple prototypes for each class, which might then be used to classify new data.

Bezdek *et al.* (1996) and (1998) compare several approaches to multiple-prototype classifier design. In general, these approaches try to find sets of class representatives through clustering and exploiting the given class labels at the same time. As enhancements to the single-prototype classifier, applying these approaches to text classification tasks seems to be a promising solution to learning heterogeneous classes. However, initial experiments have not yet shown significant improvements over the single-prototype classifier.

5.3 Experimental Evaluation

We have seen above that heterogeneous class definitions may hinder learning accurate classifiers. Examples in low-dimensional feature space clearly illustrate this problem. The experiments presented below give some examples that learning accurate classifiers with simple instance-averaging approaches in the face of heterogeneous classes can be done. We compare the single-prototype classifier and the naïve Bayes classifier learned from

Topic	Description	Newsgroups	Documents
1	Politics	*talk.politics.**	$3,000$
2	Sciences	*sci.**	$4,000$
3	Computing	*comp.**	$5,000$
4	Recreation	*rec.**	$4,000$
5	Religion	*alt.atheism, *.religion.**	$2,997$

Table 5.1: Categories derived from the 20 Newsgroups dataset and the number of documents assigned to them. Note that only 19 of the 20 newsgroups are used, omitting the newsgroup *misc.forsale*.

examples at the heterogeneous class level to classifiers learned at the more homogeneous subclass level as set out in Section 5.1.3. Based on the provision of subclass labels, the latter approach enables representing a class not only by one entity but by as many entities as there are subclasses within the respective class. Finally, note that we may certainly find or construct classes which cannot be learned accurately with a simple instance-averaging approach. Yet, the examples will show that the problem of learning heterogeneous classes in high-dimensional feature space may not be as severe as examples in low-dimensional feature space suggest.

5.3.1 Datasets and Experimental Setups

As in Chapter 4, we employ the enhanced version of the rainbow system for the learning and classification task. In particular, we use the single-prototype classifier and the naïve Bayes classifier because they are the two most popular instance-averaging approaches in text classification. The results are obtained for some modification of the 20 Newsgroups and the TREC datasets.[5] The modifications are made so as to address the problem of learning heterogeneous classes.

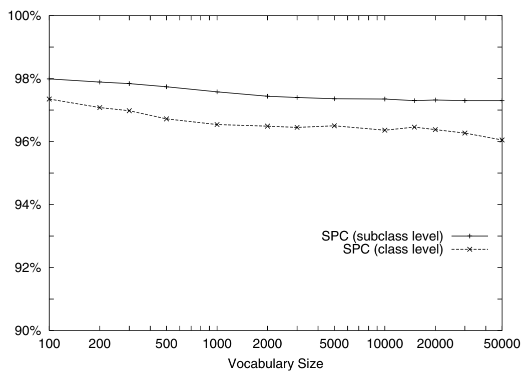
For the 20 Newsgroups dataset, we define five broader categories as shown in Table 5.1. Although they consists of several newsgroups each, the new categories are to a great extent homogeneous because the particular newsgroups grouped together cover related topics. For the filtering task, these categories are seen as subclasses. Categories 1 and 2 make up the relevant class, whereas the non-relevant class consists of categories 3 through 5. So, the two relevance classes tend to be heterogeneous.

For the TREC dataset, categories 001 and 003 constitute the relevant class. Hence, we assume that there are two homogeneous relevant subclasses. To investigate the effect of the number of subclasses making up a class, we conduct experiments with an increasing number of categories put in the non-relevant class. These categories are added in order of decreasing number of documents assigned to them.[6]

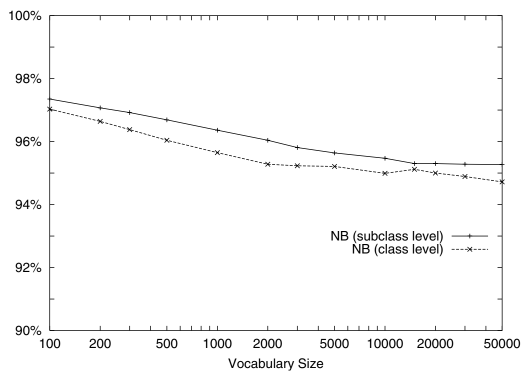
For document representation, we remove stop words and also words which occur fewer than five times in the training set. With the TREC dataset, we use all the features remaining as vocabulary. For the 20 Newsgroups dataset, the number of features is varied so as to examine the effect of vocabulary size on the separability of heterogeneous classes.

[5]See Appendixes A and B for more details on the software and text corpora, respectively.

[6]See Table B.5 (p. 194) for the categories and the number of documents assigned to them.



(a) Single-prototype classifier



(b) Naïve Bayes classifier

Figure 5.3: Classification accuracies of the single-prototype classifier and naïve Bayes classifier for the modified 20 Newsgroups dataset learned at the class level and at the subclass level when vocabulary size is varied. Note the magnified vertical scale. Clearly, exploiting subclass labels yields more accurate classifiers. Although statistically significant in most cases, the performance gain achieved is rather small and may hardly justify the larger amount of user effort required for the provision of subclass labels. Note that accuracy decreases slightly as the number of features increases. But, choosing a large vocabulary size might still be preferable as it typically yields higher recall, which does not show in this graph.

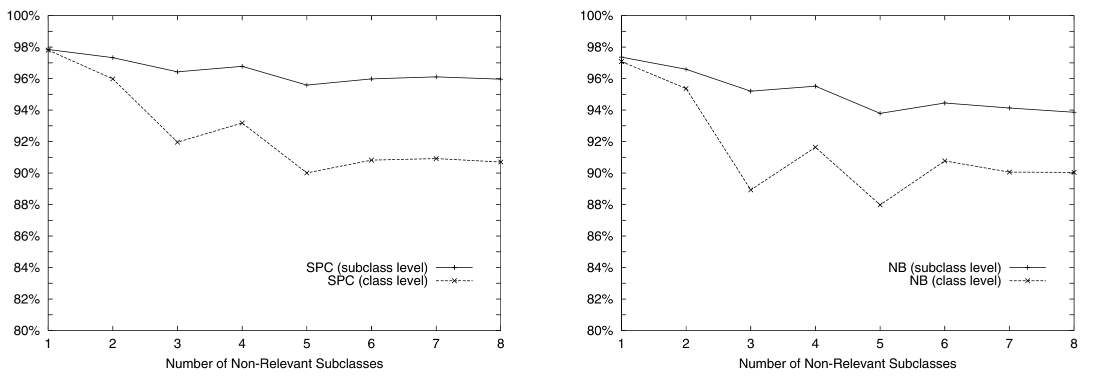
In particular, features are added in decreasing order of information gain values. When conducting the experiments, 10% of the documents are used as training set and 90% as test set. The results reported are averages over ten trials with randomly generated training and test sets.

5.3.2 Results

Figure 5.3 shows the performance of the single-prototype classifier (SPC) and the naïve Bayes classifier (NB) trained at the class and subclass levels on the modified 20 Newsgroups dataset when the number of features is varied. Note that the classifiers trained at the class level represent each relevance class with exactly one entity, whereas the classifiers trained at the subclass level represent each relevance class through as many entities as there are subclasses assigned to the respective class. For the subclass level classifiers, note that any misclassification made between subclasses assigned to the same relevance class is not considered an error since the classifiers' response at the subclass level is mapped onto the corresponding relevance class.

The learning curves show that classification accuracies of both the class level and the subclass level approaches are very similar. In all experiments, learning at the subclass level is superior to learning at the class level. Note that all results for the single-prototype classifier and most results for the naïve Bayes classifier are statistically significant.[7] But, the performance gain achieved is rather small and may hardly justify the need for the larger amount of user effort for the provision of the subclass labels. To summarize, we see that, in this case, the heterogeneous class definition does not hinder learning accurate classifiers with simple instance-averaging approaches. Also, we hardly benefit from exploiting extra knowledge about the subclass structure.

[7]With a two-sided test for the difference of two proportions at a 0.05 significance level. See Dietterich (1998) for some statistical tests for comparing supervised classification learning algorithms.



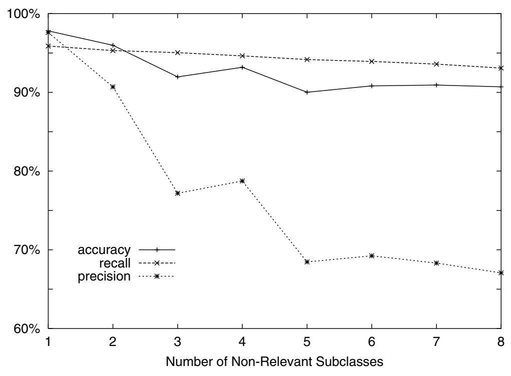
(a) Single-prototype classifier (b) Naïve Bayes classifier

Figure 5.4: Classification accuracies of the single-prototype classifier and naïve Bayes classifier for the modified TREC learned at the class level and at the subclass level when the number of non-relevant subclasses increases. Vocabulary size is fixed for all experiments. Note the magnified vertical scale. Again, exploiting subclass labels yields more accurate classifiers. For learning both at the class and subclass levels, performance decreases as the number of subclasses increases. Yet, this shows to a larger extent when each class is represented by only a single prototype. Still, the results of the class level classifiers are fairly accurate.

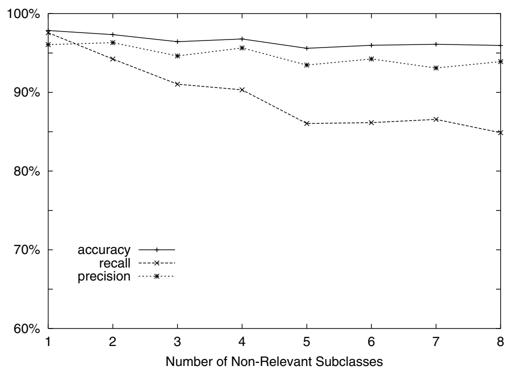
Figure 5.4 shows the performance of the single-prototype classifier and the naïve Bayes classifier trained at the class and subclass levels on the modified TREC dataset when the number of non-relevant subclasses increases. Note that there are exactly two relevant subclasses in all experiments.

When there is just one non-relevant topic, the classifiers learned at the subclass level only slightly outperform the class level classifiers in terms of accuracy. The existence of two relevant subclasses explains this difference. As the number of subclasses increases, at first, the performance gap between the subclass level and the class level classifiers widens. Then, for a larger number of non-relevant subclasses, this gap seems to remain constant. Note that the learning curves strongly depend on the order at which the non-relevant subclasses are added. This may explain not only the volatile behavior midway through the experiments but also the fact that the gap does not widen any further for larger numbers of subclasses. Irrespective of this order, we observe a tendency towards less effective classifiers as the number of subclasses increases. Yet, despite this tendency, the class level classifiers achieve reasonable performance. So, we see that simple instance-averaging approaches may cope fairly well with heterogeneous class definitions.

The difference between the classifiers learned at the class level and those learned at the subclass level gives an impression as to the extent to which automatically discovering subclasses and exploiting this subclass structure for classification tasks might enhance classification performance. In fact, assuming that the subclasses considered in the experiments described above are homogeneous, the performance of the classifiers learned at the subclass level with user-provided subclass labels might be seen as an upper performance bound for approaches that attempt to find subclasses automatically and then represent classes with multiple entities rather than one. Although the class level classifiers achieve reasonably accurate results, the classifiers learned at the subclass level show that there is some room for improvement. It is doubtful whether the performance gain justifies the additional user effort needed for providing subclass labels. So, using clustering approaches to automatically discover subclass structure should be considered instead.



(a) Class level single-prototype classifier



(b) Subclass level single-prototype classifier

Figure 5.5: Classification accuracy, recall, and precision of the single-prototype classifiers learned at the class and subclass levels for the modified TREC when the number of non-relevant subclasses increases. Note the magnified vertical scale. The class level classifier struggles with precision, whereas the subclass level classifier has lower recall.

Above, we have seen that classification accuracy decreases as the number of subclasses increases. Yet, especially when the number of non-relevant documents is large, accuracy may characterize filtering performance only insufficiently, as described in Section 2.3 (pp. 13 ff.). So, let us take a look at recall and precision, which are shown in addition to classification accuracy for both the class level and the subclass level single-prototype classifiers in Figure 5.5.

For the class level classifier, recall remains quite high as the number of non-relevant subclasses increases. It declines more slowly than classification accuracy. Precision, on the other hand, decreases considerably. For the subclass level classifier, recall and precision behave differently: precision, like accuracy, hardly decreases at all, whereas recall drops substantially. This behavior can be explained by looking at the similarity scores of new documents to their true class or subclass prototypes. Table 5.2 shows the average similarities of all relevant and non-relevant test documents obtained from both the single-prototype classifier learned at the class and subclass levels.

When representing a heterogeneous class which consists of several homogeneous subclasses by only one prototype, the similarity of a new document to this prototype will typically be smaller than the similarity to a prototype which represents only a single subclass. So, the average similarity of documents to the non-relevant prototype decreases as the number of non-relevant subclasses increases. Since the number of relevant subclasses

	SPC (class level)		SPC (subclass level)	
q_{non}	rel	non	rel	non
1	0.186	0.215	0.232	0.210
2	0.184	0.203	0.231	0.212
4	0.187	0.196	0.235	0.218
8	0.183	0.187	0.235	0.230

Table 5.2: Effect of the heterogeneous class definition on the average similarity scores of relevant and non-relevant test documents to their true class prototypes when the number of non-relevant subclasses q_{non} increases. Concerning the subclass level classifier, only the similarity score to the most similar subclass prototype is considered. Note that the similarity scores of the class level classifier tend to decrease slightly as the number of subclasses increases. When allowing for multiple prototypes per class, the similarity scores do not decrease; rather, they may increase.

remains constant, there is an increasing number of documents which will be classified as relevant. For this reason, we observe a large drop in precision and only a small change in recall for the classifier trained at the class level. For the subclass level classifier, the average similarities to the most similar non-relevant subclass prototype in fact increases. This explains why recall declines, while precision and accuracy remain almost unchanged. Hence, we see that explicit modeling of subclasses while resulting in high accuracy and precision classification of documents may compromise recall.

5.4 Conclusions

In this chapter, we have studied some examples based on two text collections to evaluate the extent to which heterogeneous class definitions affect text classification with simple instance-averaging classifiers. This is a key issue as the relevance classes of the filtering task tend to be heterogeneous. The following summarizes the empirical results.

Typically, heterogeneous class definitions cause classification accuracy of instance-averaging classifiers to degrade. This performance decrease shows that there is some room for enhancing text classification if classes are heterogeneous. Nevertheless, the examples have also shown that instance-averaging approaches may still learn accurate classifiers even though the homogeneity assumption is violated.

Learning at the subclass level rather than at the class level is a means to enhance classification accuracy in the face of heterogeneous classes, as it allows representation of classes with multiple entities. But, these classifiers may compromise recall in case there is a large number of non-relevant subclasses. So, while learning at the subclass level might be suitable for some applications, it is, in its current form, most likely not acceptable for the filtering task. Yet, note that confidence mapping or adjusting the classification thresholds for the classification scores may remedy this problem.

So far, we have assumed that subclass labels are provided by the user. This enabled us to evaluate the extent to which representing classes by as many prototypes as there are homogeneous subclasses helps enhance classification performance. Yet, in practical applications, we often cannot assume that subclass labels are provided. Thus, an issue for future research is to apply clustering approaches in order to automatically uncover subclass structure. This is certainly a challenging task because documents are typically represented by high dimensional and also sparse vectors: a situation which often causes clustering approaches to struggle when trying to find meaningful clusters.

Clustering algorithms may find groups of documents which are to some extent similar. And, these similarities among documents may actually support browsing through large document collections. Yet, whether these document clusters also enhance classification performance, is another issue. Generally, we cannot guarantee that the clusters discovered will correspond to what we are looking for: homogeneous groups of documents within the existing class structure which make learning classes easier or even possible. A similar problem has already been encountered in Section 4.3.1 (pp. 94 ff.) and has led to the development of semi-supervised learning approaches. So, likewise, existing cluster algorithms should be modified so that they exploit the predefined class structure.

Chapter 6

Quality Control

Effectively employing information filters, or classifiers in general, requires that the new documents which are to be classified come from the same set of class-specific document sources from which the training documents have been obtained. However, in a long-term application, these document sources tend to change over time for various reasons. With time, dynamic aspects may cause a classifier to become less effective than expected and may, thus, necessitate its adaptation to the changing environment. Since obtaining new training data is expensive, we aim at adapting a classifier only if necessary. So, coping with dynamic aspects falls into two subtasks: detecting that changes have occurred and adapting to these changes accordingly. This chapter focuses on detecting changes in a document stream which is filtered either with little or with no user feedback through application of techniques taken from the field of quality control.

6.1 Introduction

This section discusses in detail problems associated with the *stationarity assumption*, which justifies classifying new data through application of classifiers learned from training examples observed beforehand. Having realized that document streams are dynamic rather than stationary, we describe different types of changes which we may observe in a real-world application and discuss how these changes may affect classification performance. Finally, we consider methodologies for coping with dynamic aspects.

6.1.1 Problem Description

In the preceeding, we have studied learning algorithms which allow construction of classifiers to predict the class labels of new documents. Irrespective of the learning approach, the application of classifiers to determine the class label of new documents is based on the essential assumption that training documents and new documents come from the same source, or distribution, in statistical terms.[1] In other words, assuming that training and

[1]See Schürmann (1996), p. 309.

new documents are both generated by class-specific document sources, the application of classifiers learned from examples requires that these hypothetical document sources be *stationary*.[2] By stationary we understand the fact that new documents to be classified at different points in time are similar to the classes as initially defined through training data. That is, the distributions of documents as given by the class-specific document sources are assumed to be time invariant. Yet, even though this stationarity assumption may hold initially, it is likely to become invalid in a long-term application. For various reasons, both the topics and contents of new documents can be expected to change over time. The document sources are *dynamic*, or *non-stationary*, rather than stationary.[3] As time progresses, any changes in the learning scenario may cause a classifier to become less effective than expected. Then, a classifier should be adapted accordingly in order to maintain classification performance.

6.1.2 Dynamic Aspects

Take the task of information filtering, which amounts to separating relevant from non-relevant documents according to a particular user interest. Living in a dynamic world, we know that this classification task is subject to change. Changes in the learning scenario are often referred to as *concept drift*.[4] The problem of concept drift is ubiquitous. But what do conceptual changes look like?

In the supervised learning setting, the concept that describes the filtering task is represented through examples. So, we must distinguish between changes to a concept proper and changes in the concept representation. Note that we do not consider learning a different concept per se. The concept to be learned in information filtering will always be the documents that a user likes to read. Any concept change that we do consider will show up in the training examples, yielding the following two different types of concept change, which might be observed in a long-term application:[5]

- *Class labels of examples change.* Differently labeled examples amount to changes in the user interest. This is, perhaps, what is most commonly understood as concept change. Of course, we are still trying to learn to identify documents that a user likes to read. Yet, the topics that the user is interested in are changing. For instance, a user might become interested in other topics because of particular news-breaking events or a new job. As a consequence, the mapping from documents onto the two relevance classes will change, necessitating adaptation.

[2]See Russell and Norvig (1995), p. 553, or Mitchell (1997), p. 203, for example. Note, however, that in time series analysis, the notion of *stationarity* for stochastic processes is used in a precisely defined statistical context, e.g. see Hamilton (1994), pp. 45–46, or Greene (1997), pp. 827–830.

[3]In addition, note that not only are the document sources dynamic, but also user interests might change over time. However, we focus on changes within the document sources.

[4]According to Mitchell (1997), p. 21, learning a concept means inferring a boolean-valued function from training examples of its input and output. So, an exhaustive binary classification problem, where each item to be classified is mapped onto exactly one of two possible classes, corresponds to learning a concept. As for information filtering, the concept may be defined as 'documents that a user finds interesting', yielding the two classes *relevant* and *non-relevant* depending on a particular user interest.

[5]Also see Lam *et al.* (1996), pp. 318–319, for a similar distinction among changes.

- *Distributions of examples change.* Training examples represent what a user is interested in. As the space of documents is huge and the number of training examples is limited, they generally only partly cover the concept. Owing to changes in the environment, the distributions of documents may change over time. For instance, because of social or engineering progress or political events, the content of new documents is constantly evolving. Even the meaning of single words may evolve over time. In addition, we may observe words which have not been used before in a particular text corpus. Accordingly, the use of words in specific topics may change, causing a commensurate change in word distribution which, in turn, affects topic-specific document distributions. Here, we refer to these types of changes as *content change*. As a result, we may obtain documents from regions in feature space that have previously not been populated. Hence, it may be difficult to classify new documents. Also, the concept representation would differ when training examples were gathered at different points in time. In this sense, content change may also be regarded as concept change. However, this is arguable because content change does not imply that the topics a user is interested in have changed. Rather, new training documents only reveal information about the concept that was unknown before. In any case, the current classifier requires adaptation.

Irrespective of the type of change that we have discussed so far, we may characterize changes by the rate at which they occur. We distinguish gradual changes (*concept drift*) and sudden changes (*concept shift*). Typically, content changes are gradual, whereas changes in user interests may also be abrupt. The rate at which changes occur affects the ability to detect changes. We will have more to say on this later in this chapter.

Changes in user interest and content changes are a crucial issue in information filtering because they may substantially deteriorate classification performance. Therefore, coping with these changes is a key challenge. Note that content changes typically concern all the users involved in a particular domain. In contrast, evolving interests are always a user's own concern.

Most likely, users will know when they are interested in different topics. So, when provided with user feedback in the form of class labels for new documents,[6] the difficulty of coping with changing user interests is to implement an adequate adaptation strategy rather than detecting concept change. In contrast, perceiving changes in a constantly evolving environment is a different issue. Many users will fail to notice changes and may miss essential information. So, a challenging task will be to detect changes in the environment and alert the user to be able to trigger an appropriate adaptation strategy.

As we will see in the following section, much work has been done concerning changes in user interests in information filtering. Yet, much less work addresses coping with content changes due to changes in the environment. In the following, we focus on content changes, i.e. on changes within the set of topics which a user may either like or dislike and their respective distributions. At the same time, we assume that user interest remains constant.

[6]Recall that we do not address human-machine interaction in this dissertation. Rather, we assume that we are provided with appropriate means to present documents to users and receive feedback upon request.

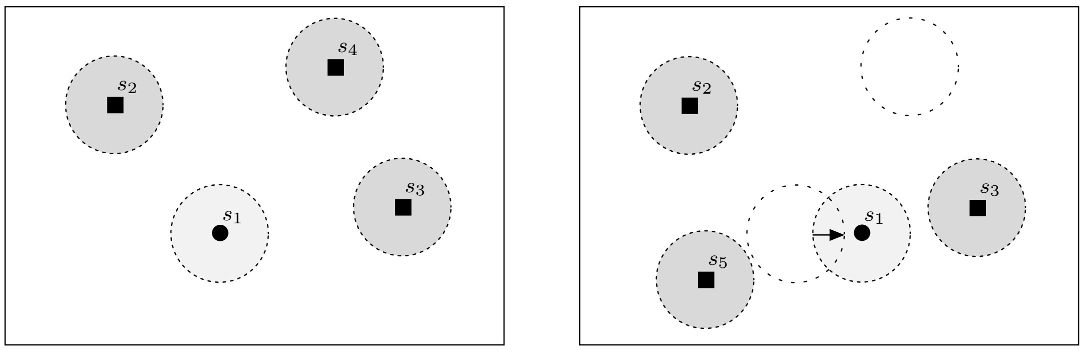


Figure 6.1: The effect that content change may have on the set of topics. Initially, there are four topics: topic s_1 is relevant and topics s_2 through s_4 are non-relevant (left). After some change has occurred, the relevant topic s_1 has drifted somewhat to the right, the non-relevant topic s_4 has vanished, and another non-relevant topic s_5 has emerged. Owing to the changes, relevant documents of topic s_1 may be confused with the non-relevant topic s_3. Also, non-relevant documents of the new topic s_5 may be classified as relevant because of their similarity to documents of topic s_1 before the changes occurred.

A Taxonomy of Content Changes

As set out in Chapter 5, the two relevance classes of the filtering task tend to be heterogenous and typically consist of more homogeneous subclasses, which we will also refer to as *topics*. As previously, we make the simplifying assumption that each document can be uniquely associated with exactly one topic. Also, each topic belongs to either the relevant or non-relevant class. With regard to these topics, we now consider possible changes which, in combination, describe the dynamic nature of the information filtering task.

Now, looking at a stream of documents, we may observe the following changes with respect to the set of topics:

- Existing topics change.
- Existing topics vanish.
- New topics emerge.

Figure 6.1 shows the impact that content change may have on the set of topics. Note that a changing topic may be considered as the superposition of two similar topics with one of them vanishing and the other just emerging. Hence, it suffices to consider vanishing and emerging topics.

Impact on Classification Performance

Having realized that the set of topics is subject to change in the manner set out above, the next step is to analyze how these changes affect classification performance.

We consider vanishing and emerging topics. In general, vanishing topics should not pose a serious problem with regard to classification performance as there will not be any more documents belonging to this topic that can be misclassified. Only when an old topic hinders a classifier from predicting class labels of other documents because of their resemblance to other topics, should action be taken to allow accurate classification of the remaining topics. In addition, note that due to a large number of topics which have vanished, a classifier may become too complex. In this case, eliminating the related training examples should be considered. Nonetheless, as some old topics might revive after a while, we might be better off keeping some training examples of the currently out-dated topics.

Dealing with emerging topics, on the other hand, is a much greater challenge with regard to classification performance. How do emerging topics affect classification performance? A classifier may predict the class labels of documents which belong to a new topic either correctly or incorrectly based on their resemblance to existing topics. If the majority of the new documents is classified correctly, the appearance of a new topic will probably not be recognized. And, with respect to classification performance, this is not necessary. So, more crucial are documents that are misclassified. Assuming forced recognition, i.e. each document must be assigned to any one of the predefined classes, two types of errors may be observed: a non-relevant document may be classified as relevant or a relevant document may be classified as non-relevant.

If, on the one hand, documents of a new non-relevant topic are presented to users, we may assume that they will complain. Although this is annoying for users, we will most likely receive feedback in the form of some true class labels. Subsequently, adaptation of classifiers may be based on this feedback. If, on the other hand, documents of a new relevant topic are erroneously withheld from users, they might miss essential information unless they are informed otherwise. This problem is crucial to the application of information filters. How can we make sure that relevant documents are not permanently misclassified because of changes?

Assume that, under the stationarity assumption, we are capable of estimating classification performance of a filtering system. This performance estimate will become useless if changes occur. Hence, it is essential to continuously evaluate performance in order to guarantee a specific level of classification performance in a long-term application. Yet, without the knowledge of the true class labels for all or at least some of the new documents, this poses a difficult problem, as we will further dicuss in the following section.

6.1.3 Coping with Dynamic Aspects

In a long-term application, a classifier may require adaptation to a changing environment to cope with violations of the *stationarity assumption*. As a rule, we do not know how and when changes will occur: they occur while a filtering system is already in operation. So, anticipating change and considering possible changes during training of classifiers is not feasible. For this reason, we consider the following two methodologies for coping with dynamic aspects:[7]

- *Adaptation at regular intervals.* A straightforward approach to handling dynamic aspects is to update or relearn a classifier at regular intervals irrespective of whether changes have really occurred. Undoubtedly, this methodology enables us to deal

[7]See Lanquillon and Renz (1999) and Lanquillon (1999b).

with a changing environment. Especially for information filtering, however, this methodology is not practical for two reasons. First, any form of adaptation requires that new training examples be available. Yet, regular provision of new training data is tedious and expensive.[8] In fact, requiring the true class label of each new document after it has been classified reduces the task of information filtering to absurdity because its aim is to reduce a user's information load. So, we cannot assume that we have enough new training data to permit frequent adaptation of a classifier. As the amount of new training data required depends on the frequency with which a classifier is adapted, the need for new training data may be reduced by choosing a suitably large interval between two successive adaptations. But, choosing an appropriate interval length is a difficult task. As we typically do not know how the document stream will change, any choice of the interval length—reaching from not adapting at all and, thus, completely ignoring changes to continuously updating after each new document (incremental learning)—appears arbitrary. To sum up, we may regard adaptation of a classifier at regular intervals as a brute-force approach to coping with dynamic aspects because it wastes human resources. Consequently, this methodology is not feasible where labeled data is scarce and expensive.

- *Adaptation only if need be.* A more elaborate methodology for handling dynamic aspects is to adapt a classifier to changes only if classification performance needs to be maintained. Deciding whether adaptation is necessary requires being able to detect changes. Hence, coping with dynamic aspects falls into two subtasks: detecting changes and adapting to these changes accordingly. As we have seen above, we always require some user feedback in form of class labels for new documents for adaptation. The reason for this is that we cannot otherwise know how to correctly classify new documents from regions in feature space which have not been populated previously. However, as we aim at adapting a classifier only if necessary, the amount of new training data may be greatly reduced. So, the benefit of this methodology largely depends on the amount of feedback required to detect changes. Clearly, we may easily detect changes when provided with true class labels for the new documents after they have been classified. But as this is not feasible in practice, the challenge is to detect changes either with little or with no feedback. Note that generally the risk of failing to recognize changes and the risk of false alarms, i.e. erroneously predicting that a change has occurred, increase as the amount of available feedback decreases. To sum up, attempting to detect changes and adapting only if necessary may make coping with dynamic aspects feasible. Yet, feasibility is achieved at the risk of failing to detect change and, thus, withholding relevant documents from the user.

As it allows minimization of the user feedback required and is, therefore, feasible, we favor the methodology that attempts to detect changes and adapts the information filter only if necessary to maintain classification performance. We primarily focus on the change detection task; see Section 6.3. In Section 6.4 we briefly address approaches to adapting to changes.

[8] Also see Section 4.1.1 (p. 80).

6.2 Related Work

In machine learning, changing concepts are often handled by time windows of fixed or adaptive size on the training data, i.e. by learning from only a certain number of the most recent examples.[9] Another approach to dealing with concept changes is by weighting training examples or parts of the hypothesis according to their age or usefulness for the classification task.[10] Note that most approaches to dealing with concept changes assume that the feedback for the examples classified becomes available after classification. Though absolutely necessary to handle changes in user interests, providing feedback for a large number of documents is still prohibitive. So, applications of these approaches may fail to work in practice when true class labels are not available. Our approach to treating changes in the environment will try to do with little or no feedback. Below, we briefly discuss some approaches to dealing with dynamic aspects when learning a concept, focusing mainly on approaches applied to the information filtering task.

Kuh *et al.* (1991) extend computational learning theory to include concepts that can change or evolve over time. They conclude that, for some concept classes, learning changing concepts is provably efficient, if the rate or the extent of change is limited in particular ways. Also, Helmbold and Long (1994) show that a window of a certain minimal fixed size allows learning of concepts for which the extent of change is appropriately limited. Note, however, that concept changes of real-world problems may not obey the restrictions imposed. For instance, user preferences for reading news articles may change almost arbitrarily often and radically.[11] Also, note that the window sizes theoretically suggested are typically very large and, thus, impractical. So, applications based on time windows usually rely on much smaller window sizes.[12]

The choice of the window size is perhaps the most crucial issue for any approach which learns from only some of the most recent training examples. Typically, we cannot determine an appropriate constant window size for all possible situations. As using a small window permits fast adaptability to changes and a larger window leads to enhanced generalization ability in phases without concept change, adjusting the window size to the current extent of change is, in general, superior to a fixed window size. Most approaches apply simple heuristics to determine the current window size as a function of classification performance. Typically, window size decreases if changes might allow discarding of out-dated examples. In contrast, window size may increase when the concept is stable, so as to provide a more representative training set. In any case, detecting changes is the first, and probably most important step, towards coping with dynamic aspects. We will have more to say on this in the following section.

Allan (1996) explores relevance feedback techniques for automatic correction of user profiles so that they more accurately reflect user interests. To better cope with changing user interests, he downweights training documents according to their age. Note that this approach follows the methodology of adapting a classifier at regular intervals irrespective

[9] See Mitchell *et al.* (1994) or Widmer and Kubat (1996), for example.

[10] See Maloof and Michalski (1995) or Nakhaeizadeh *et al.* (1998), for example.

[11] See Klinkenberg and Joachims (2000).

[12] See Widmer and Kubat (1996), Klinkenberg and Renz (1998), or Lanquillon (1999a), for example.

of whether changes have actually occurred. For his adaptive web page recommendation service, Balabanoviĉ (1997) follows a similar approach: existing user profiles are discounted through an aging factor in order to cope with changing user interests.

Other approaches are in line with the methodology of adapting to changes only if necessary to maintain classification performance. For example, Lam *et al.* (1996) propose a two-level learning approach to coping with non-stationary user interests. While a lower level is responsible for learning user profiles, a higher level uses Bayesian analysis to detect shifts in user interests. Once a shift is detected, the lower-level learning algorithm is suitably adapted.

Klinkenberg and Renz (1998) describe an adaptive information filtering system which permits dealing with changes in user interests. Their approach tries to cope with dynamic changes by, first, detecting changes and, then, adapting the information filter only if necessary through learning from some of the most recent training documents. To detect changes, they monitor performance measures, which requires that the true class labels of new documents become available in order to be evaluated. To mitigate this problem, Klinkenberg (1999) tries to operate his adaptive filtering system with only partial user feedback. For adaptation, Klinkenberg employs simple heuristics to determine appropriate window sizes. Klinkenberg and Joachims (2000) propose a method for recognizing and handling concept changes with support vector machines which is both theoretically well-founded and, at the same time, effective and efficient in practice. Nevertheless, note that their approach still requires the true class labels of new documents to become known after they have been classified.

Tauritz and Sprinkhuizen-Kuyper (1999) and Tauritz *et al.* (2000) present an adaptive information filtering system which is concerned with filtering in changing environments and considers changes in user interests as well as content changes in the document stream. They employ the concept of evolutionary computation to cope with a changing environment. Although results are promising with respect to classification performance, in its current form, their approach suffers from the same drawback as the aforementioned approaches: it requires the user to provide feedback for the documents classified.

In his work on autonomous text classification systems, Lewis (1995) suggests continously monitoring the performance of a classifier and adapting to changes if necessary. This corresponds to the approach that we will follow later in this chapter for handling dynamic aspects. Lewis focuses on estimation of performance measures for probabilistic classifiers without user feedback. However, he does not provide any evaluation for a dynamically changing document stream. In our work, we will try, among other things, to make use of these estimates to detect changes in a real-world application.

A fairly new field of study is *topic detection and tracking (TDT)*, which aims at finding and following news events in a stream of news stories. The task is divided into three key subtasks: segmenting a stream of recognized speech into distinct stories, identifying those news stories which are the first to discuss a new event occurring in the news, and, given a small number of sample news stories about an event, finding all the follow-up stories in the stream.[13] Although the TDT task and our work appear similar in that they attempt

[13]See Allan *et al.* (1998).

to discover novelty, i.e. changes in a document stream, they do, in fact, differ in several respects. In the TDT task, events are defined as clusters localized in space and time and associated with some specific action. This contrasts with the definition of a topic in text classification, which refers to a cluster of documents about the same subject. For example, *'the election of the president of the United States of America in the year 2000'* may be seen as an event, whereas *'presidential elections'* in general might be a topic. Another distinction is that TDT is defined as an unsupervised learning problem: classes are not known and fixed in advance. As a consequence, there are no relevance classes with respect to particular user interests which are to be learned. While TDT aims at detecting events as early as they occur, we focus on detecting changes when they have a significant effect on classification performance. The tracking of events once they have been discovered shows some similarities to the issue of learning classes when labeled training data is scarce. It is different, though, since events are much more short-lived, and several distinct events may be similar to the same topic but may still have to be separated.[14]

6.3 Detecting Changes

In the preceeding, we have seen that continuously adapting a classifier so as to cope with a changing environment is typically not feasible. In contrast, trying to detect changes first and, then, adapting to them if necessary is practical. Only this methodology allows comprehensive minimization of the user effort required. Detecting changes necessitates continuous observation of characteristic values of the filtering process such as classification performance or document properties. We focus on performance indicators which can be evaluated either with little or with no user feedback. We propose to use techniques from statistical quality control to detect changes in terms of deviations from the expected values of the indicators observed.

6.3.1 Statistical Quality Control

In order to detect any concept change within a stream of data, at least one characteristic value should be observed and compared to previous values regularly. We refer to this process as *monitoring*. Regarding the change detection problem from a statistical perspective, the sequence of observed characteristics represents a discrete random process which has some inherent variation. Now, the key difficulty is to distinguish between normal variation caused by stochastic fluctuations—usually called noise—and variation due to real concept changes. If the data stream is noisy, it may contain inconsistent examples, so-called outliers, which do not fit in the current concept but are spurious and may never appear again. Note, though, that examples which seem to be outliers might also herald concept change. However, in a classification system learned from examples, it typically takes more than one new example to change the concept.

[14]For example, see Yang *et al.* (1999) and Yang *et al.* (2000) for some approaches to the TDT task.

> *Methods designed to react quickly to the first signs of concept drift may be misled into overreacting to noise. This results in unstable behavior and low predictive accuracy. On the other hand, an incremental learner that is designed primarily to be highly robust in the face of noise runs the risk of not recognizing real changes in the target concepts and may adjust to changing conditions very slowly, or only when the concepts change radically.*[15]

The problem of distinguishing between noise and real change is broadly studied in the statistical literature about quality and process control.[16] There, this problem is known as distinguishing between chance causes and assignable causes of variation. We employ two well-known techniques to monitor quality indicators for the information filtering process: a variant of the *Shewhart control chart* and the *cumulative sum control chart*.

A control chart is defined as a plot of a certain characteristic value, such as the sample mean or a measure of variability, which is taken sequentially in time.[17] In addition, control charts include bounds, or control limits, which help to determine whether a particular sample is within acceptable limits of random variation. Through these limits, control charts try to distinguish between variation caused by expected stochastic perturbations and variation caused by unexpected changes. If the plotted characteristic is outside the limits, we conclude that something has happened to the process observed: possibly some concept change has occurred. In this case, some action should be taken to cope with this change. For classification problems, concept changes typically require adaptation of the classifier used; see Section 6.4.

Shewhart Control Chart

In 1924, Walter A. Shewhart developed the statistical control chart concept as a change detection method for the continuous inspection of the quality of a manufactured product.[18] His approach is known as the *Shewhart control chart*. Below, we describe a variant of this approach as applied later in this chapter for the detection of changes in document streams.

Suppose that a characteristic value ν_t, computed at different points $t \in \mathbb{N}$ in time, is monitored as time progresses. Typically, the characteristic describes samples of the data stream taken from particular periods of time, such as the classification accuracy of some recent documents classified. Let the mean μ be the expected value for the characteristic value with standard deviation σ. For the moment, assume that both μ and σ can be estimated on the basis of some historical data. The values ν_t are plotted in time-series fashion versus time with appropriate warning and action limits indicating that the process has changed.[19] See Figure 6.2 for an illustration of the Shewhart control chart.

The action limits are equivalent to the classical control limits and are basically suitable for the detection of grave changes, i.e. concept shift. Typically, there is an upper action

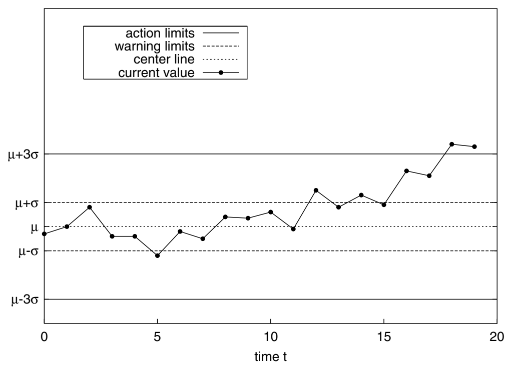
[15]From Widmer and Kubat (1996), p. 82.

[16]See Montgomery (1997) for a thorough introduction to statistical quality control.

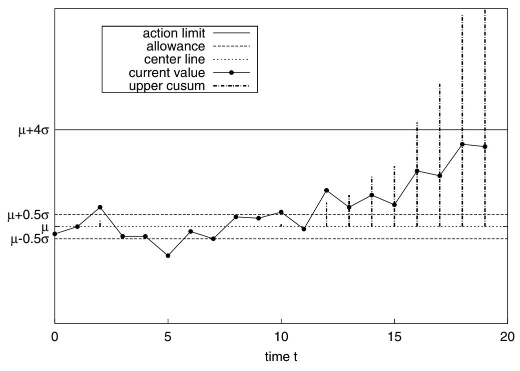
[17]See Hogg and Ledolter (1992), pp. 187–196, for example.

[18]See Montgomery (1997), p. 9.

[19]See Montgomery (1997), pp. 132–138, for some basic principles of the Shewhart control chart.



(a) Shewhart control chart



(b) Cusum control chart

Figure 6.2: Illustration of a Shewhart control chart (left) and a cusum control chart (right). In both charts, the dots show the values of the characteristic observed at time t. Typically, these points are connected to better visualize trends. Initially, the characteristic falls around the mean value, which is also referred to as the center line. About halfway through the chart, we observe an upward drift in the characteristic values. The upper action limit of the Shewhart control chart recognizes this change at $t = 18$. If warning limits were used, the change might be detected earlier. The upper cusum shown as impulses in the cusum control chart exceeds its action limit even earlier than the warning limits of the Shewhart control chart. Note that the number of consecutive non-zero cusum values may be used to determine the time point at which the change might have started. Here, the change is likely to have started at $t = 12$.

limit and a lower action limit, which are often set at $\mu + 3\sigma$ and $\mu - 3\sigma$, respectively.[20] The theoretical explanation for the choice of these action limits is as follows. Suppose that the characteristic values ν_t follow a normal distribution with mean μ and variance σ^2. Then, the probability that any ν_{t_i} falls between the action limits is extremely close to unity, namely about 0.9973. Hence, it is very rare that any ν_{t_i} would fall outside the action limits if the process is stable. If, on the other hand, ν_{t_i} falls outside the action limits, this is taken as evidence that there is some change in the data stream. Although the distribution of the characteristic around its mean value is often unknown, in many practical situations, such as those addressed in the following, it may be considered approximately normal according to the *central limit theorem*.[21] Therefore, the three-sigma action limits are often a reasonable choice. Nonetheless, we may choose any other confidence level at which to suspect change. So, in general, we may set the action limits at $\mu \pm \alpha\,\sigma$, with $\alpha > 0$. Setting $\alpha = 2.33$, for example, corresponds to a one-sided 99% confidence interval for observing values either smaller or greater than the action limits, assuming that the characteristic values approximately follow a normal distribution. In general, note that smaller values of α increase the risk of a false alarm, i.e. indicating change when there is none, whereas larger values of α increase the risk of failing to recognize changes.

That small and persistent changes often remain undetected is a well-known drawback of the basic Shewhart control chart with its control limits. The reason for this is that the Shewhart control chart considers only the current value of the characteristic observed and ignores any information given by its entire sequence.[22] Below, we discuss another control chart which may help remedy this problem. To increase sensitivity of the Shewhart control

[20]For some characteristic values, we might only use one of the action limits. When observing classification accuracy, for example, a lower control limit may suffice since deviations beyond the upper action limit are certainly acceptable and would not require adaptation.

[21]See Box and Luceño (1997), p. 27, and Montgomery (1997), p. 62.

[22]See Montgomery (1997), pp. 313–314.

chart, warning limits may be used in addition to the action limits. Warning limits are set somewhat closer to the mean than the action limits, for example at $\mu \pm \sigma$ or $\mu \pm 2\sigma$. In general, we may set the warning limits at $\mu \pm \beta\,\sigma$ with $0 < \beta < \alpha$. If the characteristic value falls outside the warning limits but is still inside the action limits, the process might still be in control. Yet, the situation might also indicate a trend towards a conceptual change. To make use of the warning limits, the position of successive characteristic values is often considered. For example, a certain number of warnings in a row might indicate that the process is out of control and thus trigger an action. In this work, we use one-sigma warning limits in addition to the action limits and also suspect conceptual change if the characteristic observed falls outside the same warning limit three times in a row.[23]

Having determined the probability distribution governing the characteristic values to be monitored, a key difficulty in using Shewhart control charts is to estimate appropriate values for μ and σ. In general, these values are unknown: they can only be estimated on the basis of historical data. Yet, when a process is begun, there may not be enough historical data available. To mitigate this problem, we might collect additional data while the characteristic is monitored and adapt μ and σ accordingly. Note that this may be a drawback when we are already in a phase of concept drift: the change may not be detected as the action and warning limits may adapt with the change. Note that any problem concerning the estimation of μ and σ largely depends on the characteristic value monitored and will, thus, be deferred until Section 6.3.2.

Cumulative-Sum Control Chart

A *cumulative-sum* (or *cusum*) *control chart* considers the entire sequence of characteristic values. In particular, the cusum chart accumulates deviations from the expected value over time and may, thus, detect small changes which occur successively. A similar approach to detecting small changes is the exponentially weighted moving-average control chart.[24]

As previously, assume that we monitor the characteristic ν_t over time t with mean μ and standard deviation σ. The cusum control charts accumulates deviations from the mean which are above target with one statistic C^+ and deviations from the mean which are below target with another statistic C^-. The statistics C^+ and C^- are called the one-sided upper and lower cusum, respectively. At time t, they are defined recursively as[25]

$$C_t^+ = \max[0, \nu_t - (\mu + \delta\,\sigma) + C_{t-1}^+] \tag{6.1}$$
$$C_t^- = \max[0, (\mu - \delta\,\sigma) - \nu_t + C_{t-1}^-] \tag{6.2}$$

where the starting values are $C_0^+ = 0$ and $C_0^- = 0$. The term $\delta\,\sigma$ is often referred to as slack value or allowance and allows us to ignore deviations from the mean to some small extent. So, in effect, the upper and lower cusums accumulate deviations from the mean which are greater than the allowance, with both statistics reset to zero upon becoming negative.

[23] See Taylor *et al.* (1997), for example. Also, see Box and Luceño (1997), p. 62, for more action rules.
[24] See Box and Luceño (1997), pp. 71–74, or Montgomery (1997), pp. 313–341.
[25] See Montgomery (1997), p. 318.

We assume that a change has occurred at time t_i if either one of the cusums exceeds the action limit $\gamma\,\sigma$, i.e. if $C^+_{t_i} > \gamma\,\sigma$ or $C^-_{t_i} > \gamma\,\sigma$.[26] Typically, δ is set to 0.5. Yet, there seems to be only small grounds for selecting γ in the literature. In this work, we choose $\gamma = 4$.[27] For example, this would signal change for two observations in a row with a deviation of at least two and a half times the standard deviation from the mean. In addition, note that the cusum chart is not only able to detect changes but also allows us to determine when the particular change might have begun to occur: we just have to find the time point at which the cusum rose above zero for the last time prior to exceeding the action limit. See Figure 6.2 for an illustration of the cusum control chart using only the upper cusum.

Combined Cusum-Shewhart Control Chart

As we have seen above, the cusum control chart permits detection of concept drift. Yet, it is less effective than the Shewhart control chart in detecting concept shift. In an attempt to make use of the individual strengths, we will also consider using both control charts in combination so as to enhance change-detection ability. In particular, we apply both the Shewhart control chart and the cusum control chart and suspect concept change if either one or both charts signal change.[28]

6.3.2 Quality Indicators for Text Classification

In the preceeding, we have introduced approaches to monitoring characteristics so as to detect changes in some underlying process. Now, we derive quality characteristics from the information filtering process. Specifically, the key challenge is to define indicators which do not just allow detection of changes but whose computation is also feasible.

Batch Processing

In information filtering, we are concerned with classifying documents from an incoming stream according to a given concept, i.e. documents which a user likes to read. As we have discussed previously, it typically takes more than a single document to change a concept. Hence, we cannot detect changes with indicators based on a single document. Rather, we require computation of indicators on the basis of sets of documents which are clustered in time. In particular, we assume that the document stream is divided into batches of documents with respect to their chronological order. Then, for each batch, any quality indicator for the information filtering process may be computed on the basis of all documents in the respective batch. Note that this batch processing will average single observations, thus reducing noise, so that the influence of outliers is only slight.

[26] Again, for some characteristics, it will suffice to use either the upper or the lower cusum. For example, when monitoring classification accuracy, we are interested solely in the lower cusum because enhanced accuracy typically does not require adaptation to concept change.

[27] See Montgomery (1997), p. 322.

[28] See Montgomery (1997), p. 325.

Even if information filtering is regarded as an online process, where each document is classified as it arrives,[29] constructing batches is quite natural since, for example, all documents arriving in the course of a day or a week could be grouped together. Note that the computation of indicators with respect to the batches does not conflict with the online presentation of documents to the user: both processes can be carried out independently.

Obviously, batches may differ in the number of documents they contain. For the sake of simplicity, we assume that all batches have the same size. Note, though, that determining an appropriate batch size is not at all trivial. If the batch size is chosen too small, some indicators observed might still be biased and, thus, inaccurate. Yet, if it is too big, the reaction to concept changes might be too slow. In fact, choosing a suitable batch size depends on the composition and the behavior of the data and on the application at hand.

A Taxonomy of Quality Indicators

Our quality indicators will be derived from batches of documents. But what kind of quality indicators shall we use? We consider three categories of characteristics:[30]

- *Text properties:* The quality indicator characterizes the current batch of documents from the stream, e.g. through class distributions or term frequencies. For example, concept changes might affect the average number of index terms occurring in a document. Basically, indicators based on text properties concern the preprocessing of documents, i.e. the transformation of plain text into feature vectors as set out in detail in Section 3.2. For this reason, they allow detection of changes even before documents are actually classified. So, a user may be alerted that the upcoming classification decisions may be uncertain. Unless they require the true class labels to be known, the computation of this type of quality indicator is feasible.

- *Classifier properties:* The quality indicator characterizes the classifier or, in other words, the concept representation which has been established on the basis of the current batch of documents as training data.[31] These indicators require new classifiers to be learned at regular intervals. Yet, as we have seen above, this is generally not feasible because new training documents are often not available. Therefore, we will not further consider indicators based on classifier properties.

- *Classification properties:* The quality indicator is based on final or intermediate classification results, e.g. performance measures such as classification accuracy. Naturally, making use of performance measures is the most straightforward approach to assessing the quality of the underlying filtering process, unless efficiency plays a major role. The computation of most performance measures is not practical as it typically requires the true class labels of the documents classified. Hence, we will have to use alternative performance measures.

[29] See Section 15 (p. 7).

[30] See Klinkenberg and Renz (1998), Lanquillon and Renz (1999), and Lanquillon (1999b).

[31] See Rissland *et al.* (1995) for an approach to measuring structural change in concept representations.

The discussion above has already ruled out the use of classifier properties since our primary goal is to avoid building new classifiers at regular intervals unless changes in the concept have actually occurred. This leaves us with indicators based on text or classification properties. In this work, we confine ourselves to using indicators based on classification properties, i.e. on general forms of performance measures. The reason for this is that any concept change which significantly affects the document representation and, thus, any indicator based on text properties should also affect indicators based on classification properties because classifiers are trained to distinguish among classes through class-specific differences in the document representations. So, in fact, we may see exploiting classification properties as a particular means to spotting differences in text properties.

Assessing Performance Indicators

We consider general forms of performance measures as quality indicators for the information filtering process. In Section 2.3 (pp. 13 ff.), we have described some of the most common performance measures used in text classification and information filtering: *accuracy*, *error rate*, *precision*, and *recall*.

To reiterate, recall is the probability that a relevant document will be presented to the user. Accuracy and error rate express the probability that a document will be classified correctly and incorrectly, respectively. For their computation, these performance measures require the true class labels of the documents classified. As provision of true class labels is often not feasible, we cannot in general evaluate these performance measures. Only the computation of precision, i.e. the probability that a document presented to the user is indeed relevant, may be feasible as it requires feedback solely for documents presented to the user. Yet, precision as a single performance measure does not suffice to fully characterize classification performance. The reason for this is that precision does not recognize relevant documents which are withheld from users. Therefore, we may be able to detect that a particular topic is vanishing, but we may not recognize that a new relevant topic is emerging. To cope with this problem, we would have to evaluate further performance measures like recall.[32] As this reflection shows, we have to derive alternative quality indicators to overcome the evaluation problem pertaining to the computation of most of the common performance measures.

Below, we discuss performance measures which can be evaluated both with little and with no user feedback. In particular, we first consider computing common performance measures on the basis of samples of the documents classified rather than on all the documents classified. Next, we dicuss an approach to estimating expected performance based on posterior probabilities of class membership. Finally, we propose a method which observes the uncertainty in classification decisions. Although these approaches assume the two-class information filtering task, they may easily be extended to more than two classes.

Estimation of Performance Measures from Document Samples. Direct computation of many common performance measures is not feasible as it requires the true class labels

[32] See Section 42 (p. 15).

of the documents classified. Similar evaluation problems may arise when controlling the quality of a manufacturing process. For example, inspecting each item produced may not be practical either because it is too expensive to check each item in large volume production or because the items produced might be destroyed upon inspection. To assess quality in these cases, typically, a random sample is drawn from the population of all items produced. Any quality indicator evaluated on this sample is then used as an estimate for the quality indicator pertaining to the entire population.

Concerning information filtering, we may draw a sample of documents from each batch and require feedback from the user only for the documents in this sample. Since we may assume that feedback for documents presented to the user will be obtained, it may suffice to draw a sample only from documents which, being classified as non-relevant, have been withheld from users. Now, any performance measure may be evaluated on the basis of partial user feedback and, then, be used as an estimate for the performance measure evaluated for all documents of a batch.

Assume that we evaluate, for each batch, the error rate e_{sample} on the document sample drawn from the respective batch. To monitor the sample error, we consider the values obtained for different batches as the characteristic for the filtering process at different points in time. For the application of the Shewhart and the cusum control charts, we have to determine appropriate estimates for the mean and the standard deviation of the sample error. A document can be classified either correctly or incorrectly. Hence, the error rate follows a binomial distribution. Suppose we are given q sample errors evaluated on earlier batches. For $i = 1, \dots, q$, let n_i denote the number of documents in the batch at time i. We estimate the target value of the sample error by the weighted mean of the previous sample error rates $e^{(i)}_{\text{sample}}$, giving[33]

$$\bar{e}_{\text{sample}} = \frac{1}{\sum_{i=1}^{q} n_i} \sum_{i=1}^{q} n_i \, e^{(i)}_{\text{sample}} \tag{6.3}$$

For the batch at time t with size n_t, the standard deviation is estimated by

$$s_{e_{\text{sample}}} = \sqrt{\frac{\bar{e}_{\text{sample}}\,(1 - \bar{e}_{\text{sample}})}{n_t}} \tag{6.4}$$

With these estimates, the action and warning limits of the control charts can be drawn accordingly. For example, the upper warning and action limits of the Shewhart control chart are set at $\bar{e}_{\text{sample}} + s_{e_{\text{sample}}}$ and $\bar{e}_{\text{sample}} + 3\, s_{e_{\text{sample}}}$, respectively.

Typically, the difficulty of this approach lies in providing sample error rates of earlier batches. This becomes particularly apparent when setting up the monitoring process: there may be no batches prior to the first one observed. As a remedy, p-fold cross-validation with the training data might be applied in order to estimate the target value $\bar{e}_{\text{sample}}$. In this case, the error rates on each of the p held-out sets of the cross-validation procedure can be used in place of the sample error on previous batches. Also, we may consider adapting the mean and the standard deviation according to the sample error rates

[33]See Lanquillon (1999a), for example.

Relevant?	User says *yes*.	User says *no*.
System says *yes*.	$A = \sum_{i=1}^{n} s_i \, Z_i$	$B = \sum_{i=1}^{n} (1 - s_i) \, Z_i$
System says *no*.	$C = \sum_{i=1}^{n} s_i \, (1 - Z_i)$	$D = \sum_{i=1}^{n} (1 - s_i) \, (1 - Z_i)$

Table 6.1: Contingency table for n documents with random variables as relevance judgements (following Lewis (1995), p. 248).

on recently processed batches so as to obtain more accurate estimates. For the estimation of mean and standard deviation, we will consider the sample error rates on recent batches only if they are within the warning limits of the Shewhart control chart.

To sum up, monitoring the sample error rate instead of the true error rate allows substantial reduction in user feedback. Yet, users must still read and hand-label documents which they are not interested in. Clearly, smaller samples require less feedback from the user. Note, though, that sample size is inversely related to the ability to detect change: choosing a small sample size increases the risk of false alarms or failing to recognize changes. Below, we dicuss performance measures which do not require any user feedback.

Estimation of Expected Performance. Now, assume that we do not have any user feedback at all, i.e. we do not have any true class labels for the documents classified. In addition, assume for now that a classifier responds to a new document with estimates of the posterior class probabilities, based on which classification decisions are made. If we use a non-probabilistic classifier, we can transform the classifier's response into posterior probabilities through confidence mapping.[34]

To estimate the expected performance of a classifier on new data, Lewis (1995) models the unknown user judgement, i.e. the unknown true class labels, by a Bernoulli (0/1) random variable Z_i with parameter p_i giving the probability that Z_i will take on the value 1 for each document $\mathbf{d}_i$ classified by the filtering system. The event $Z_i = 1$ occurs if the current document is actually relevant, and $Z_i = 0$ otherwise. It is assumed that each document is judged independently of all others. In addition, the value of each p_i is assumed to be the classifier's estimate of the posterior probability of the relevant class for document $\mathbf{d}_i$. In other words, we assume that the more confident the classifier is in its classification decision, the larger the probability will be that the user would make the same decision. Yet, in reality, this assumption if often violated: the classifier's estimates of the posterior class probabilities may be rather poor; otherwise it would not yield any classification errors.

While the true class labels for new documents are unknown, we certainly know the class assigned to a new document by the classifier. So, rather than using a random variable, let the simple Boolean variable s_i denote this classification decision with $s_i = 1$ if the classifier considers document $\mathbf{d}_i$ as being relevant, and $s_i = 0$ otherwise.

In Section 2.3 (pp. 13 ff.), we have expressed common performance measures in terms of the entries of a contingency table. With the true class labels unknown, we cannot directly construct the contingency table. Nevertheless, for a set of n documents $\mathbf{d}_i$, we may define its entries in terms of random variables based on the classification decision s_i and

[34]See Section 3.3.4 (pp. 75 ff.).

the random variables Z_i, which replace the relevance judgements of a user; see Table 6.1. Rather than based on the entries a, b, c, and d of the normal contingency table, a performance measure is now expressed in terms of the random variables A, B, C, and D.[35] For example, for a set of n documents, the expression for the error rate is given by

$$ERROR = \frac{B+C}{n} = \frac{\sum_{i=1}^{n}(1-s_i)\,Z_i + \sum_{i=1}^{n} s_i\,(1-Z_i)}{n} \tag{6.5}$$

To apply these expressions to the monitoring process, we have to get rid of the random variables. Let $g(\mathrm{s},\mathbf{Z})$ be a function to evaluate an arbitrary performance measure in terms of the system decisions $\mathrm{s} = (s_1,\ldots,s_n)$ and the user judgements $\mathbf{Z} = (Z_1,\ldots,Z_n)$ for the n documents. Lewis (1995) provides a general expression for the expected performance measure $E[g(\mathrm{s},\mathbf{Z})]$ as the sum of performance values that would be obtained for each of the 2^n possible combinations of the user judgements for all n documents, weighted by the probability of each judgement as determined by the classifier, yielding

$$\begin{aligned} E[g(\mathrm{s},\mathbf{Z})] &= \sum_{\mathbf{z}\in\{0,1\}^n} \mathrm{P}(\mathbf{Z}=\mathbf{z})\,g(\mathrm{s},\mathbf{z}) && (6.6)\\ &= \sum_{\mathbf{z}\in\{0,1\}^n} (\prod_{i=1}^{n} p_i^{z_i}\,(1-p_i)^{1-z_i})g(\mathrm{s},\mathbf{z}) && (6.7) \end{aligned}$$

For a large number n of documents, it may not be feasible to directly evaluate this expression. Yet, for the expected error rate, for example, a simpler expression is given by[36]

$$e_{\text{expected}} = \frac{1}{n}\sum_{i=1}^{n}((1-2s_i)\,p_i + s_i) \tag{6.8}$$

With $s_i \in \{0,1\}$ and $p_i \in [0,1]$, the values of the summands in this expression can be written in a simpler form as

$$\begin{aligned} (1-2s_i)\,p_i + s_i &= \begin{cases} p_i & \text{if } s_i = 0 \\ 1-p_i & \text{otherwise} \end{cases} && (6.9)\\ &= |s_i - p_i| && (6.10) \end{aligned}$$

As a result, this summand quantifies the uncertainty in each classification decision by measuring the difference between the class assigned and the posterior probability based on which the classification decision has been made. For the two-class filtering problem, this difference is minimal if the classification decision is made by thresholding the posterior probability at 0.5, which corresponds to the maximum a posteriori decision rule.[37]

Note that this expression has some obvious similarity to the RAD reject-criterion commonly applied in pattern recognition, which rejects decisions if the distance between the class predicted and the classification scores exceeds a predefined threshold θ:[38]

$$\mathrm{RAD}_i^2 = |s_i - p_i|^2 > \theta \tag{6.11}$$

[35] See Lewis (1995).

[36] See Lewis (1995).

[37] See Section 210 (p. 62).

[38] See Schürmann (1996), pp. 293–298, and Schürmann (2000).

We will have more to say on rejecting classification decisions due to uncertainty for the quality indicator described next.

For the monitoring process, consider that the expected error rate is a measure of the average uncertainty of a set of classification decisions rather than an error rate. Hence, we do not suppose a binomial but a normal distribution for this characteristic. As above, let there be q expected error rates $e^{(i)}_{\text{expected}}$ on earlier batches with size n_i for $i = 1, \ldots, q$. The target value for the expected error rate is evaluated as the weighted mean of the previous expected error rates, yielding

$$\bar{e}_{\text{expected}} = \frac{1}{\sum_{i=1}^{q} n_i} \sum_{i=1}^{q} n_i \, e^{(i)}_{\text{expected}} \tag{6.12}$$

with a standard deviation estimated by

$$s_{e_{\text{expected}}} = \sqrt{\frac{1}{q-1} \sum_{i=1}^{q} (e^{(i)}_{\text{expected}} - \bar{e}_{\text{expected}})^2} \tag{6.13}$$

Warning and action limits of the control charts are plotted according to these estimates. For the provision of values for the expected error rate on earlier batches, consider the same reflections as made for the sample error rate.

To summarize, expected performance measures permit evaluation of quality indicators without user feedback. Yet, the values obtained will typically differ significantly from the traditional performance measures, which are based on complete user feedback for all documents classified. The reason for this behavior is the fact that the classifier's estimates of the posterior class probabilities given a document are often not very reliable. An expected performance measure may well be used as a quality indicator of a filtering system. As we are not interested in accurate estimates of performance measures but in change detection instead, we may consider applying this indicator to non-probabilistic classifiers without confidence mapping. For example, assume similarity-based classifiers: the classification scores received from such a classifier might be used in place of the posterior probabilities. In this case, Equation (6.8) could be formulated to express the average dissimilarity of new documents to the class assigned rather than the average uncertainty.[39]

Virtual Rejects: Observing Uncertainty in Classification Decisions. Again, we assume that we do not have knowledge of the true class labels for the documents classified. Yet, instead of measuring the average uncertainty of a set of classification decisions as done above for the expected error rate, we now quantify uncertainty through counting decisions made with a confidence below a certain threshold. In other words, we take into account decisions that should be rejected for reasons of uncertainty. Nevertheless, as we assume forced recognition, i.e. each new document must be classified as either relevant or non-relevant, we talk about *virtual rejects*: documents whose classification decisions would be rejected if forced recognition were not assumed.

[39] See Lanquillon (1999b).

Assume that a classifier responds to a new document with classification scores for each class based on which the classification decision is made. Hence, we require that each class of the classification task is appropriately represented. Note that we have seen in Section 364 (p. 132) that the learning task in information filtering may be simplified by modeling only the relevant class explicitly since the non-relevant class is often heterogeneous and, thus, difficult to represent. To detect changes, however, we have to model both the relevant and the non-relevant classes explicitly. This is "because to recognize a thing as new, one must be able to distinguish it from what is old. But to be distinguishable, some of the things we perceive must be similar enough as classes or patterns for us to compare with another thing to tell if it is the same or different."[40] Put in other words, we cannot recognize a document as new solely through the dissimilarity to the relevant class. We also need to know the resemblance to the non-relevant class.

In addition, the application of the reject indicator to be introduced below is based on the assumption that changes in the document stream cause classification decisions to become more uncertain as the resemblance of new documents to the known classes decreases. Otherwise, we would not be able to recognize changes. For the evaluation of the reject indicator, we focus on classifiers which respond to a new document with a classification score in the unit interval. It is essential that these scores are not normalized so that they add up to unity. The reason for this is that normalization will hinder the reject indicator from identifying uncertain classification decisions as they may appear more confident than they actually are. So, we may use, for example, any type of similarity-based classifier such as the single-prototype classifier. Also, the naïve Bayes classifier might be used if the estimates of the posterior probabilities of the classes given a document are not normalized.

Equipped with classification scores for each class, assume that we classify a new document as belonging to the class which yielded the maximum classification score. This decision rule remains unchanged. However, we also introduce a reject-criterion which signals that a classification decision is too uncertain. In particular, let $c = H(\mathbf{d}) \in \mathcal{C}$ denote the class predicted for document $\mathbf{d}$ by some classifier H. We virtually reject the classification decision if the classification score s_c for the class predicted is below a certain threshold.[41] Note that the threshold parameter may be class-dependent. So, the reject-criterion is defined by the predicate function $\varrho(\mathbf{d})$ which evaluates as one if document $\mathbf{d}$ should be rejected, and otherwise zero:

$$\varrho(\mathbf{d}) = \begin{cases} 1 & \text{if } s_c < \theta_c \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

The key challenge is to define the threshold parameters θ_{c_i} for each possible class $c_i \in \mathcal{C}$.

We will derive class-dependent threshold values based on the following motivation. Let H be a classifier that explicitly models each relevance class and consequently responds to a new document with a classification score for each class. By individually thresholding the scores for the relevant and the non-relevant classes, we can obtain two new classifiers H_{rel} and H_{non}. The idea is to measure the disagreement between H and either H_{rel} or

[40]From Klapp (1986), p. 81.

[41]This reject-criterion is similar to the RAD reject-criterion, which is often applied in pattern recognition; see Schürmann (1996), pp. 293–298.

Algorithm 6.1 Computation of Reject Thresholds for Filtering Task

```
Input: set of training documents $D = D_{non} \cup D_{rel}$, learning algorithm $L$
1: perform leave-one-out or $p$-fold cross-validation with $L$ on $D$ so as to obtain unbiased
   classification score vectors $\mathbf{s}_i = (s^{(i)}_{non}, s^{(i)}_{rel})$ for each training document $\mathbf{d}_i \in D$
2: split score vectors into two sets $S_{non} = \{\mathbf{s}_i | \mathbf{d}_i \in D_{non}\}$ and $S_{rel} = \{\mathbf{s}_i | \mathbf{d}_i \in D_{rel}\}$
3: for all $c \in \{non, rel\}$ do
4:     define subsets $S^{<}_{\bar{c}}(\theta_c) = \{\mathbf{s} \in S_{\bar{c}} | s_c < \theta_c\}$ and $S^{>}_{c}(\theta_c) = \{\mathbf{s} \in S_c | s_c \geq \theta_c\}$
5:     determine $\theta_c$ such that
           $$\frac{|S^{<}_{\bar{c}}(\theta_c)|}{|S_{\bar{c}}|} \approx \frac{|S^{>}_{c}(\theta_c)|}{|S_c|}$$
6: end for
Output: class-specific reject thresholds $\theta_{non}$ and $\theta_{rel}$
```

H_{non}, depending on the decision of H. That is, if H predicts the relevant class, we would look at the response of H_{rel}. On the other hand, if H predicts the non-relevant class, we would consider the response of H_{non}. The classification of any document for which the two classifiers considered disagree is virtually rejected. So, the main task is to determine threshold parameters θ_{rel} and θ_{non} which would permit reasonable classification of documents if the classifiers H_{rel} and H_{non} are used in isolation.

According to the aforementioned motivation, determining class-specific threshold values is straightforward. For each class, we have to apply confidence mapping so as to transform any type of classification scores into confidence values as set out in Section 3.3.4 (pp. 75 ff.). Based on these confidence maps, the threshold values can be determined as those classification scores which yield a confidence value of 0.5 when trying to separate the respective class from the other classes. Note that confidence mapping yields not only the threshold values which we require but also a complete mapping from the range of classification scores onto the unit interval. Hence, it is more complex and expensive than actually necessary. In the following, we describe a simpler approach to determining only the class-specific threshold parameters.

Following the concept of confidence mapping, a confidence value of 0.5 corresponds to a classification threshold for which the odds of classifying a document as either relevant or non-relevant are even. Since in many filtering tasks the non-relevant documents by far outnumber the relevant documents, we will consider relative numbers of documents rather than absolute numbers. In other words, we assume that misclassifying a document is inversely proportional to class frequency.[42] As a consequence, finding the classification threshold which yields a confidence of 0.5 corresponds to finding the point of intersection of the density functions of the two overlapping distributions of classification scores. Algorithm 6.1 outlines how these threshold values can be approximated.[43]

To achieve reasonable results, it is essential to obtain some unbiased classification scores for documents which have not been used for training. For this reason, we apply either

[42]This is equivalent to using normalized histograms for confidence mapping; see Section 3.3.4 (p. 76).

[43]See Lanquillon (1999b).

leave-one-out or p-fold cross-validation on the training data.[44] Then, having split the classification scores into two subsets according to the true class labels of the respective documents, we proceed separately to determine the thresholds θ_{non} and θ_{rel} for the non-relevant and the relevant classes, respectively. For example, consider the evaluation of θ_{non}.[45] Now, the objective is to find a value θ_{non} which yields the best separation between relevant and non-relevant documents with regard to the relative number of documents in each class. In other words, we require that the largest fraction of documents with non-relevant classification scores below θ_{non} is indeed relevant and that the largest fraction of documents with non-relevant classification scores above θ_{non} is indeed non-relevant. Note that the number of relevant documents with a non-relevant score below θ_{non} increases and the number of non-relevant documents with a non-relevant score above θ_{non} decreases as θ_{non} increases. Hence, there will be a point at which the fractions of these documents become approximately equal. The value of the classification score at this point best separates the relevant and non-relevant classes with respect to the relative numbers of documents and is, thus, used as the reject threshold θ_{non}.

Having determined the threshold parameters for the reject-criterion, we can evaluate a quality indicator based on the virtual rejects on each batch of documents, i.e. based on the fraction of classification decisions which should be rejected due to uncertainty. Note that this is similar to monitoring the fraction of non-conformities in a manufacturing process, seeing a document whose classification should be rejected as non-conforming with some specified standards.[46] For a set of n documents, the reject characteristic ν_{reject} is evaluated as the fraction of those documents which should be rejected, yielding

$$\nu_{\text{reject}} = \frac{1}{n} \sum_{i=1}^{n} \varrho(\mathbf{d}_i) \tag{6.15}$$

With regard to the change detection process, we can evaluate the target value $\bar{\nu}_{\text{reject}}$ as weighted mean from earlier batches as described above for the sample error rate and the expected error rate. Since the classification of a document should be either rejected or accepted, we assume a binomial distribution of the reject indicator. Hence, the standard deviation for the current batch of size n_t is estimated by

$$s_{\nu_{\text{reject}}} = \sqrt{\frac{\bar{\nu}_{\text{reject}}\,(1 - \bar{\nu}_{\text{reject}})}{n_t}} \tag{6.16}$$

With the estimates $\bar{\nu}$ and $s_{\nu_{\text{reject}}}$ for the target value and standard deviation for the reject indicator, the action and warning limits of the control charts can be drawn as described above. The reflections pertaining to the initialization phase of the monitoring process for the sample error also apply for the reject indicator.

[44] For the extreme case, with n training documents, performing leave-one-out cross-validation is the same as n-fold cross-validation. If leave-one-out cross-validation is computationally not feasible, p-fold cross-validation with $p \ll n$ is preferred.

[45] For the evaluation of θ_{rel}, we would proceed similarly.

[46] See Montgomery (1997), pp. 251–275, for more details on this type of control chart.

To sum up, the reject indicator monitors the fraction of classification decisions which should be rejected due to uncertainty. Therefore, it permits monitoring of the filtering process without user feedback. Unlike the expected performance measures discussed above, note that this indicator requires that the reject thresholds be determined empirically for each situation. The reason for this is that the thresholds depend on the document stream and the classifier applied. As a consequence, we see that the ability to detect changes largely depends on the choice of the classifier: classifiers with poor estimates of the posterior class probabilities given a document, such as the naïve Bayes classifier, may yield poor change recognition ability. In contrast, similarity-based classifiers like the single-prototype classifier are more suitable as their response realistically reflects the resemblance of new documents to the predefined classes.

6.4 Adapting to Changes

An information filtering system must be adapted to concept changes in order to maintain classification performance. Once changes have been detected, there are basically two methodologies for adapting a classifier used to filter documents: the existing classifier can be updated on the basis of some recent examples, or a new classifier can be learned from scratch based solely on a currently representative set of training examples.

Note that, irrespective of the methodology chosen, we assume that the classifier can only be adapted at discrete time points, namely between two batches of documents. Hence, the classifier remains unchanged while processing the documents of any given batch.

6.4.1 Updating an Existing Classifier

This methodology brings up the question of how to combine the knowledge inherent to the existing classifier with new training examples so as to yield a more effective classifier. Often, this is done by *incremental learning algorithms*. Note, though, that the design goal in incremental learning is typically to produce a classifier which does not depend on the sequence in which the training examples are presented to the learning algorithm.[47] For our problem, however, the design goal is different: the chronological order of the training documents should have an effect on the result of a learning algorithm because the sequence in which training documents become available may reveal essential information about their relevance and validity. Nevertheless, principles from incremental learning approaches might be adopted so as to cope with a changing environment.

Approaches to updating a classifier crucially depend on the learning algorithm used to produce that classifier, i.e. they are classifier-dependent. When using nearest-neighbor learning approaches, for example, updating is as simple as adding new training examples and possibly deleting some out-dated examples from the training set. In this case, updating should rather be viewed as learning a new classifier from scratch because the focus is on providing a more representative training set, which is not trivial, as we will

[47] See Utgoff (1989) or Utgoff (1994).

see below. Updating those approaches which simply aggregate class-specific information into prototypes for each class, such as the naïve Bayes classifier and the single-prototype classifier, is also straightforward: the class prototypes can be updated incrementally by adding information about new documents whenever they become available. Also, old examples may be removed simply by subtracting information pertaining to the respective documents. For other classifiers, by contrast, updating can be more difficult a task. For example, consider a support vector machine, which tries to find a decision surface with maximal margin between the two classes to be separated.[48] Changes in the class structure expressed through new training examples may require a decision surface which differs drastically from the one learned previously. So, depending on the extent of change, learning a classifier from scratch may be easier than updating an existing classifier.

Also, note that changes in the document stream may necessitate updating not only a classifier but also the document representation. If changes in the document preprocessing phase are required so as to better cope with the changing environment, the classifier learned based on the out-dated document representation may no longer be valid. In this case, learning a new classifier rather than updating an existing classifier might be preferable. For the sake of simplicity and classifier-independence, we focus on learning a new classifier from scratch as discussed in the following. Yet, note that updating approaches may provide more ground for further minimization of user effort and should, thus, be taken into account in future research.

6.4.2 Learning from Scratch

This methodology avoids the problems related to updating an existing classifier and the document representation by running through the entire learning process from scratch. Hence, the difficulty is shifted from processing updating strategies to providing a truly representative set of training examples each time a classifier is to be adapted to changes.

Provision of a representative set of training examples is certainly not trivial. As introduced in Section 6.2, many machine learning approaches try to solve this problem either by using a time window or by weighting training examples according to their age or usefulness with respect to the classification task and removing out-dated examples. Most approaches, theoretically or heuristically motivated, assume that the true class labels are available for all new examples once they have been classified. Clearly, we cannot assume that we have obtained the true class labels for documents which have been withheld from users. Hence, most approaches are not practical for the filtering task.

A simple approach, which is often very effective in practice, is to assume that the examples of the most recent batch are representative of the current situation. Then, it suffices to use the current batch as the training set based on which the new classifier is learned. When learning a new classifier so as to adapt to changes, we use the current batch of documents as training examples and require that the user provide true class labels for these documents. At this point, we assume that only plain supervised learning algorithms are used. Below, we also consider the application of semi-supervised learning.

[48]See Section 224 (pp. 66 ff.).

6.4.3 Further Minimization of User Effort

Trying to detect changes through quality indicators which do not require expensive user feedback and adapting to the changes recognized may greatly reduce user effort. But still, each learning phase requires new labeled documents which users must provide. Is there more potential to minimize user effort in the learning step?

Obviously, at this point, further minimization of user effort concerns the availability of labeled training data. So, all the approaches to reducing the need for labeled training data described in Chapter 4 may be considered. Instead of plain supervised learning from the most recent batch of documents, we may label only a small fraction of the most recent documents and apply semi-supervised learning. Also, we may consider using further batches of documents as unlabeled data.

When requesting class labels for the new documents of the most recent batch, the user need not start from scratch and hand-label all documents. Instead, the current classifier's response may be used to support labeling of documents. In particular, we require the true class labels for at least those documents whose classification decisions are uncertain. In case the reject indicator is used, for example, all documents which would be rejected could be hand-labeled. Note that this approach can be regarded as *active learning*.[49] Also, *document clustering* may assist users in providing class labels: labels may be required only for some prototypical documents rather than for each document.

As the main focus of this chapter is on change detection, we confine ourselves to adapting to changes through learning a new classifier from scratch based on labeled documents from the most recent batch. Yet, the combination of quality control approaches with semi-supervised learning will be an issue in Chapter 7.

6.5 Experimental Evaluation

This section provides empirical evidence that applying quality control techniques to detect changes and adapting a filtering system to changes only if necessary allows maintenance of classification performance in changing environments with greatly reduced user effort. We present experimental results on a subset of the TREC dataset and on the Reuters dataset with the single-prototype classifier.[50] Further results, also with other text corpora, are reported in Lanquillon and Renz (1999) and Lanquillon (1999b) and (1999c).

6.5.1 Datasets and Experimental Setups

As previously, we employ the enhanced version of the rainbow system for the learning and classification task of the filtering system. In addition, we use a script to simulate the batch processing, including the monitoring process, and the adaptation of the filtering

[49] See Section 289 (pp. 87 ff.). Also see Lewis and Catlett (1994) and McCallum and Nigam (1998b).

[50] See Appendix B for more details on the text corpora used.

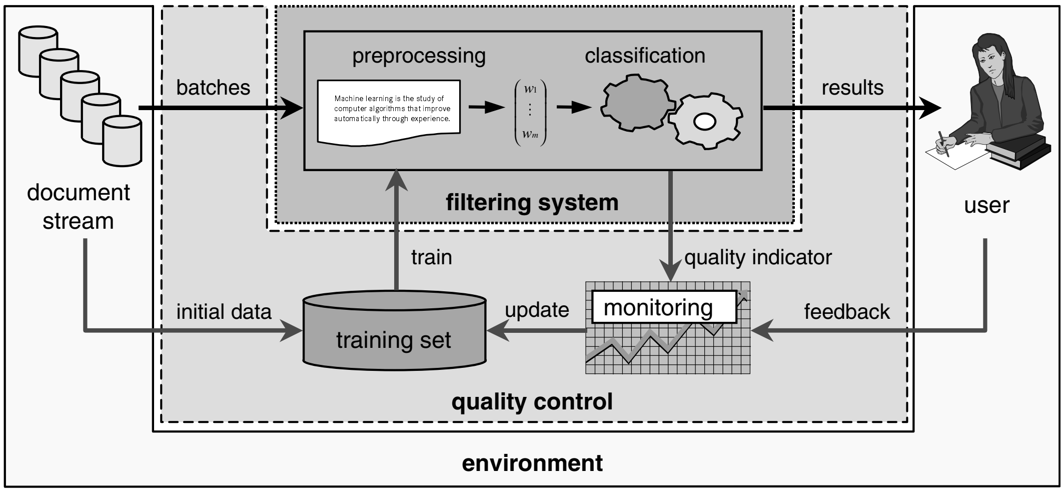


Figure 6.3: Framework of an adaptive information filtering system based on a standard filtering system and a quality control unit. The quality control unit tries to detect changes in the document stream and can trigger adaptation of the filtering system.

system.[51] Figure 6.3 shows a general framework of an adaptive filtering system based on a standard filtering system and a quality control unit which controls change detection and adaptation of the filtering system. The figure also depicts the interactions between the two components and the environment, i.e. the document stream and the user.

The standard filtering system is the core of the adaptive filtering framework because it carries out the main task of the system: classifying documents from a stream as either relevant or non-relevant for a particular user. First, the filtering system is learned based on some initial training data. When it is in operation, its performance is checked by monitoring quality indicators derived from the filtering process as set out in Section 6.3. Relevant documents are presented immediately to the user. Yet, for the monitoring process, the document stream is divided into batches of documents according to their chronological order. For the evaluation of some quality indicators and if changes are suspected, the monitoring process may require feedback from the user in terms of true class labels for some or all of the documents of the current batch. A representative training set is maintained through storage of new documents for which the true class labels have been provided by the user. If the monitor has detected some change, the filtering system is adapted based on the current training set as described in Section 6.4.

In Chapter 7, we will look at a dataset which may contain some real changes as it has been gathered over a longer period of time. At this point, however, we prefer simulated changes over real changes because they permit controlled evaluation of the quality control unit. So, in particular, we consider two artificial change scenarios: a concept shift on the subset of the TREC dataset and a concept drift on the Reuters dataset.

For the first experiment with concept shift, we use all ten of the categories contained in the subset of the TREC dataset; see Appendix 429 (p. 194). Categories 001 and 003 are defined to be relevant, while the remaining 8 categories represent non-relevant topics. The documents are split randomly into 21 batches. Each batch contains 331 documents with some of the documents being discarded. The first batch serves as the initial training set

[51]See Appendix A for more details on the software.

while the remaining 20 batches represent the temporal development. Concerning relevant documents, batches 0 to 13 contain only documents of topic 003. Batch 14 comprises documents of both relevant topics. From batch 15 on, there are only documents of topic 001. Each batch contains the same composition of documents of the non-relevant topics. So, there are 58 relevant and 273 non-relevant documents of different topics in each batch.

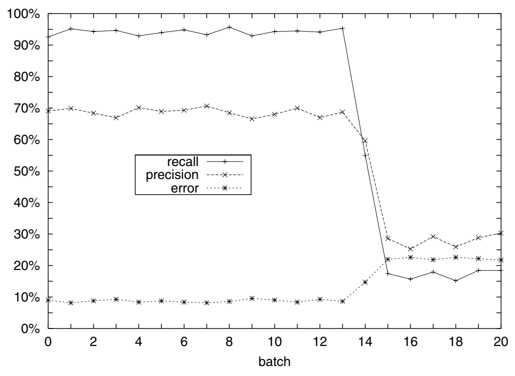
For the second experiment with concept drift, we use a subset of the Reuters collection; see Appendix 428 (p. 193). Two of the largest categories, *corporate acquisitions (ACQ)* and *earnings (EARN)*, are used as relevant topics. All other categories are combined to represent the non-relevant class. The text corpus is split randomly into 41 batches, each containing about 500 documents. In all batches, a constant number of relevant documents of topic ACQ is present. Starting from batch 15, a gradually increasing number of documents from the other relevant topic EARN is added. Documents from the non-relevant class are distributed among the batches in such a way that the batch sizes are approximately equal. Again, the first batch serves as the initial training set and the batches remaining represent the temporal development of the document stream.

For document representation, we remove stop words and also words which occur fewer than five times in the training set. From the remaining set, the $2,000$ most informative words according to the information gain measure are selected as vocabulary. Initial experiments have shown that this is a reasonable choice: as a rule, using more words slightly increases precision but decreases recall on the datasets provided. As learning algorithm we choose the single-prototype classifier not only because it is effective and efficient but also because the similarity scores based on which it assigns class labels to new documents are very suitable for the change detection task; see Section 415 (p. 162).

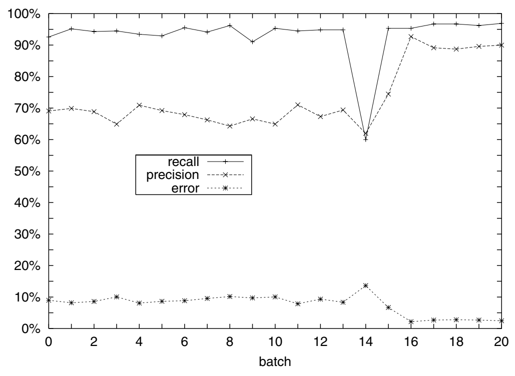
In the following, we present results on both change scenarios with five approaches. The first approach, *no adaptation*, serves as baseline approach. It is learned only once when the filtering system is set up and is then left unchanged throughout the whole process. Second, we pretend to have user feedback for all documents classified and we *relearn* the classifier of the filtering system after each batch, irrespective of whether changes have occurred or not. Since provision of the class labels is intractable, the relearn approach is only a theoretical benchmark: ideally, a feasible adaptive approach should be as effective as the relearn approach but require much less user effort. So, the other three adaptive approaches are feasible as they try to detect changes and adapt the classifier only if necessary to maintain classification performance. As quality indicators, the three alternative performance measures described in Section 6.3.2 are used: sample error, expected error, and virtual rejects. The indicators are monitored with the combined cusum-Shewhart control chart set out in Section 6.3.1. All results reported are averages over ten trials with batches created with different random seeds.

6.5.2 Results

Concept Shift on the TREC Dataset. Figure 6.4 shows the performance of the baseline and the benchmark approaches with no adaptation and with adaptation after each batch, respectively. At first, the performance of both approaches is acceptable. After the change



(a) No adaptation to changes



(b) Relearning after each batch

Figure 6.4: Performance of the baseline and benchmark approaches for concept shift on the TREC dataset.

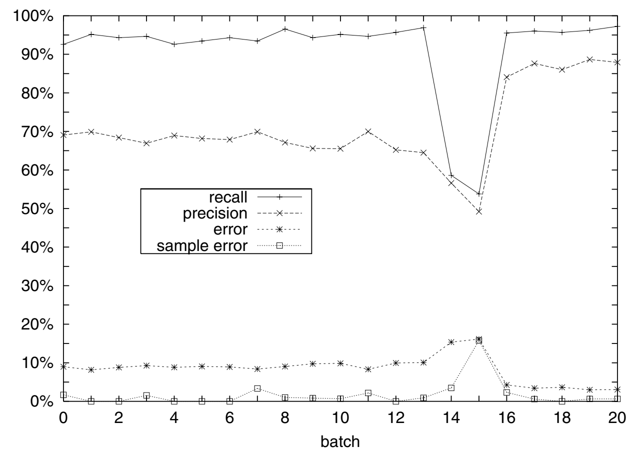
has occurred, however, the performance of the no adaptation approach decreases significantly because it cannot identify documents of the new topic as relevant. The performance of the relearn approach quickly recovers from the performance decrease in batch 14. Note that, after the change, precision is higher than before because the new relevant topic 001 can be separated more effectively from the non-relevant class than the old relevant topic 003. Also, the results show that it suffices to use the documents of only the most recent batch as training set. Obviously, coping with changes is simple when new labeled training data is available to learn a new classifier at regular intervals.

Figure 6.5 shows the performance of the three feasible adaptive approaches. With greatly reduced user effort, these approaches recover to some extent from the performance decrease caused by the concept shift at batch 14. The sample error rate is evaluated based on a randomly drawn subset of 5% of the documents per batch. Adaptation to the concept shift is almost as effective as that of the relearn approach. When monitoring the expected error rate, which does not require any user feedback, adaptation to the concept shift is much slower: concept shift is detected only with a delay of several batches. Also without user feedback, the reject indicator detects the concept shift very reliably and achieves fast adaptation, which can compete with the relearn approach. In most trials, monitoring the reject indicator signals change either when the change actually occurs or with a delay of one batch, i.e. when the change has reached its full extent.

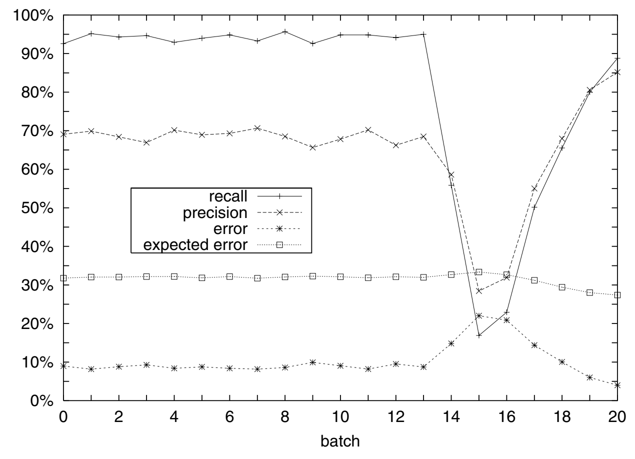
Table 6.2 compares, for all five approaches, the average performance per trial over all batches. Also, it shows the average number of adaptations made per trial while processing the entire set of batches and the average fraction of documents for which user feedback in terms of class labels has been required in order to detect changes and to adapt to changes.

Approach	Recall	Precision	Error	Adapted	Feedback
No adaptation	69.21%	56.03%	13.05%	0.0	0%
Relearning after each batch	93.20%	73.40%	7.52%	20.0	100%
Monitoring sample error rate	91.22%	70.91%	8.36%	2.6	17%
Monitoring expected error rate	80.32%	64.94%	10.30%	1.2	6%
Monitoring reject indicator	89.80%	70.85%	8.33%	1.3	6%

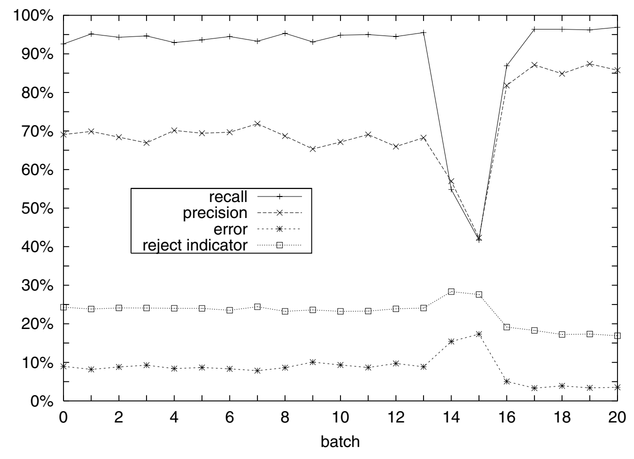
Table 6.2: Average results per trial over all batches for the concept shift on the TREC dataset.



(a) Monitoring sample error rate

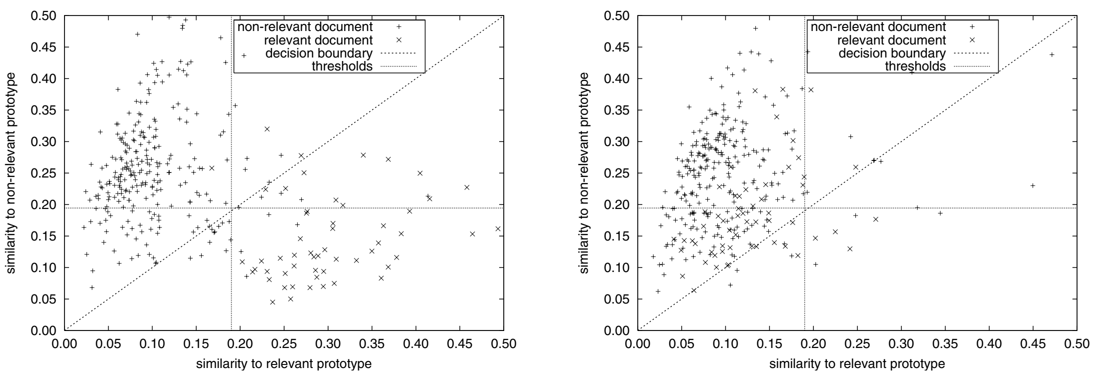


(b) Monitoring expected error rate



(c) Monitoring reject indicator

Figure 6.5: Performance of the adaptive approaches for concept shift on the TREC dataset.



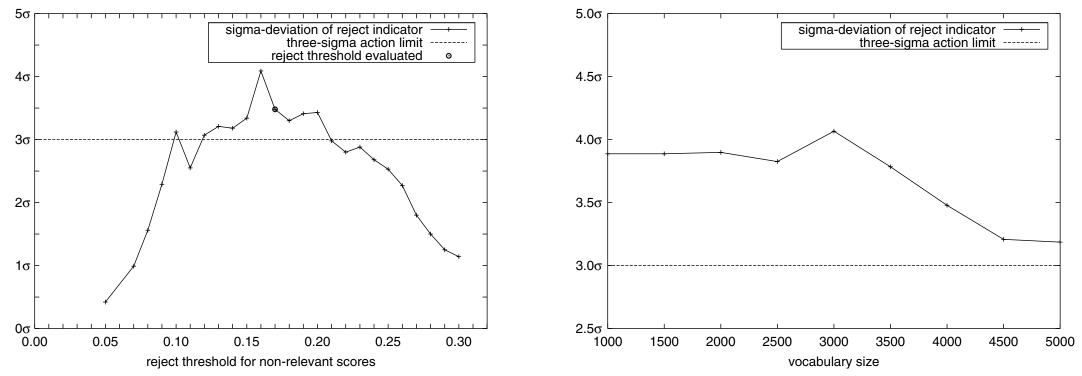
(a) Before concept shift (b) After concept shift

Figure 6.6: Classification scores before and after the concept shift on the TREC dataset. The diagonal line indicates the decision boundary as obtained from classifying documents according to the maximum classification score. Documents in the bottom left corner below the reject thresholds should be rejected. Owing to the concept shift, the similarity of new documents to the relevance prototypes decreases. This causes a significant increase in the number of documents which fall within the reject region.

No adaptation and relearning after each batch are two extremes which require no or complete user feedback, respectively. On average, monitoring the sample error rate signaled more than twice as many changes per trial than monitoring the expected error rate or the reject indicator. Although it requires some user feedback, monitoring the sample error rate causes a large number of false alarms. The larger number of adaptations and the small amount of user feedback for each batch yield a relatively high fraction of documents for which feedback is required. As a consequence, monitoring the sample error rate is not as much of a remedy in reducing user effort as the other two adaptive approaches. Monitoring both the expected error rate and the reject indicator requires only a small amount of feedback in order to obtain new training documents. For this change scenario, monitoring the reject indicator yields faster adaptation than monitoring the expected error rate.

To further justify the application of the reject indicator in detecting changes, we now take a closer look at the functionality of the reject indicator. The results presented so far have shown that monitoring the reject indicator allows effective change detection without user feedback. Figure 6.6 shows the classification scores of a batch of documents to the relevance prototypes before and after the simulated concept shift on the TREC dataset. For this change scenario, the figure illustrates that the classification scores returned by the classifier decrease because of the change. Hence, the number of uncertain classification decisions, which should be rejected, increases. As a result, we see that the assumption concerning the application of the reject indicator, namely that documents belonging to a new topic yield lower classification scores, is indeed reasonable.

Figure 6.7 depicts the operating characteristics of the reject indicator when the reject threshold θ_{non} for the non-relevant class and the number of features is varied. For the variation of the non-relevant reject threshold, note that only documents which are classified as non-relevant with a score below the threshold are considered as rejects; decisions in favor for the relevant class are not considered. The operating characteristic for θ_{non} shows that the value determined by Algorithm 6.1 yields an acceptable threshold parameter: the deviation of the reject indicator from the mean is above the three-sigma action limit of the Shewhart control chart. If θ_{non} is chosen too small, too few uncertain classification



(a) Varying reject threshold θ_{non} (b) Varying number of features

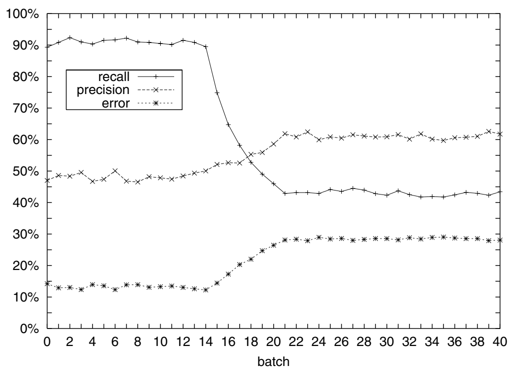
Figure 6.7: Operating characteristics of the reject indicator after the concept shift on the TREC dataset as occurred when the reject thresholds (here only the threshold θ_{non} for the non-relevant class) and the number of features are varied. The vertical axes indicates the deviation of the reject indicator from the mean in multiples of the standard deviation (sigma). Both mean and standard deviation are estimated based on values of the reject indicator observed prior to the concept shift. For example, the Shewhart control chart would signal that a change has occurred when the deviation is above the three-sigma action limit. Algorithm 6.1 yields a value of about $\theta_{non} = 0.17$, which falls within the acceptable range, i.e. the deviation of the reject indicator caused by the concept shift is above the three-sigma action limit. Using the algorithmically determined thresholds, the ability to detect changes through monitoring the reject indicator tends to decrease when the number of features increases. Note that the curves are fairly erratic because the data points are obtained from a single trial only.

decisions are considered as rejects. In contrast, when using too large a value of θ_{non}, the deviation decreases because too many correct classification decisions fall in the reject region. In both extreme cases, the reject indicator cannot be used to detect changes.[52]

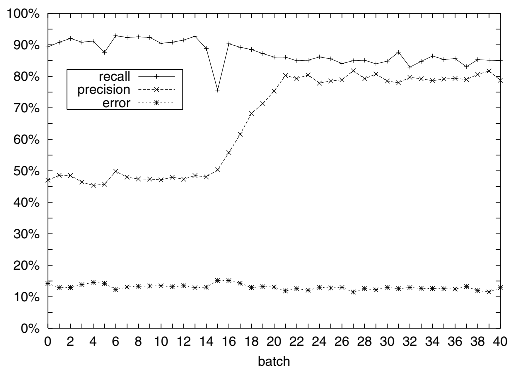
The ability to detect changes through monitoring the reject indicator depends not only on the choice of the reject thresholds but also on the number of features used to represent text. The reason for this is that vocabulary size affects the classification scores returned by the classifier. In particular, when increasing vocabulary size by adding words in decreasing order of information-gain values, the chance of finding words in documents which are less specific for any of the classes increases. So, an increasing number of words which also occur in any document of a new topic may emerge because of concept changes. As a result, classification scores assigned to documents of a new topic may differ to a smaller extent from the classification scores assigned to documents of the known topics, and change detection may be more difficult. As illustrated in Figure 6.7 (right), the deviations of the reject indicator from the mean tend to decrease as the number of features increases.

Concept Drift on the Reuters Dataset. The performance of the baseline and the benchmark approaches with no and complete user feedback, respectively, are shown in Figure 6.8. At first, acceptable recall can be achieved only at relatively low precision. Both approaches reach a higher level of precision when the second relevant topic EARN arises. The reason for this behavior is that the fraction of non-relevant documents per batch decreases and, consequently, there are fewer documents which can be misclassified as relevant. Again, in the presence of concept change, the relearn approach manages to maintain an acceptable level of recall, while the performance of the non-adaptive approach is not acceptable.

[52]Also see Lanquillon and Renz (1999).



(a) No adaptation to changes



(b) Relearning after each batch

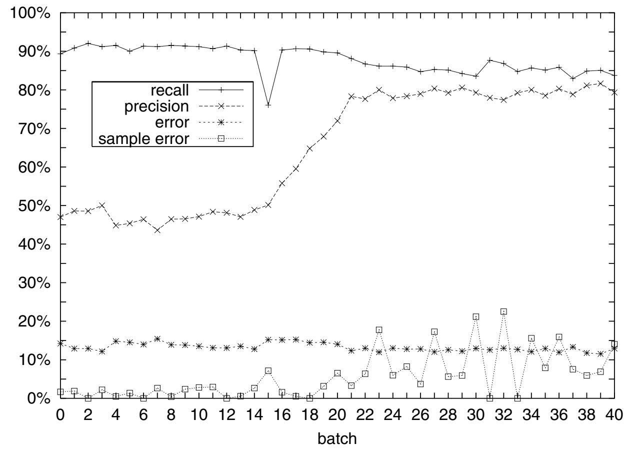
Figure 6.8: Performance of the baseline and benchmark approaches for concept drift on the Reuters dataset.

Figure 6.9 shows the performance of the three adaptive approaches when monitoring the alternative performance measures. All of them can cope to some extent with the concept drift beginning at batch 15. For monitoring the sample error, we again draw a 5% sample of documents per batch. Adaptation to the concept drift is fast and effective: performance is similar to relearning after each batch. Yet, the sample error rate is very volatile and, thus, causes the filtering system to be adapted frequently. As a consequence, this approach requires too much feedback to be practical. Increasing the sample size would reduce volatility. This would, in turn, reduce the need for user feedback concerning adaptation. With respect to change detection, however, the user feedback required would increase. When monitoring the expected error or the reject indicator, adaptation to the concept drift is slower: it takes several batches for the gradual concept change to be recognized. From a statistical perspective, this is acceptable as the concept change has to reach a specific level of significance before becoming distinguishable from noise. Similar to the concept shift scenario, the reject indicator allows faster adaptation than the expected error rate. It signals more changes and, thus, requires more user feedback for the adaptations triggered. In particular, monitoring the reject indicator suggests that the filtering system be adapted twice per trial. Yet, note that only very few of the changes detected are actually false alarms. Most of them are detected along the period of concept drift and are acceptable.

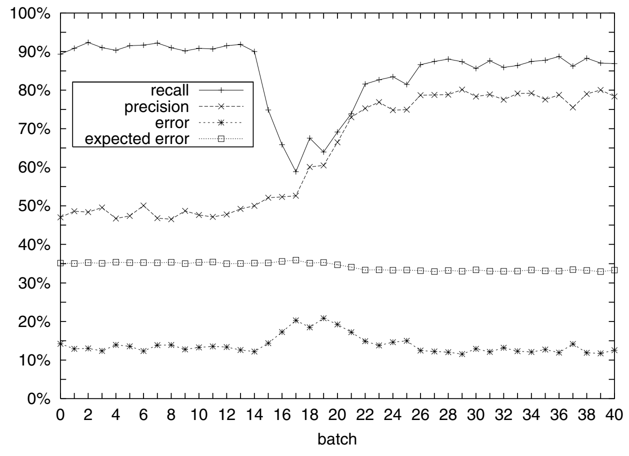
Table 6.3 compares the average performance over all batches, the average number of adaptations, and the average fraction of the documents for which feedback has been required for all five approaches. With respect to both effectiveness and efficiency, monitoring the reject indicator outperforms the other approaches for this change scenario.

Approach	Recall	Precision	Error	Adapted	Feedback
No adaptation	61.98%	55.58%	21.94%	0.0	0%
Relearning after each batch	87.40%	65.96%	13.02%	40.0	100%
Monitoring sample error rate	87.76%	65.38%	13.24%	13.3	37%
Monitoring expected error rate	84.67%	64.30%	13.89%	1.4	4%
Monitoring reject indicator	86.56%	64.50%	13.51%	2.0	5%

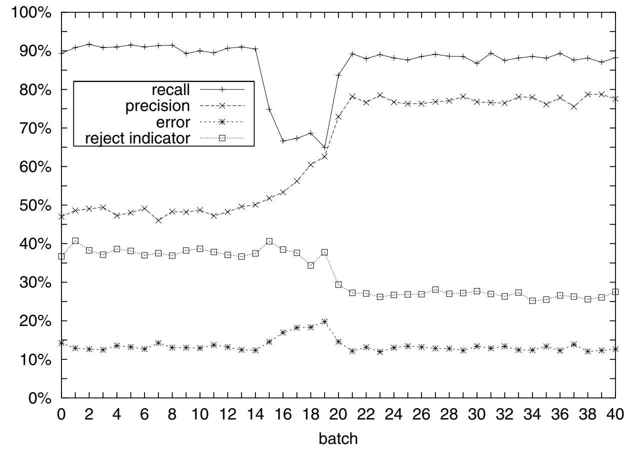
Table 6.3: Average results per trial over all batches for the concept drift on the Reuters dataset.



(a) Monitoring sample error rate



(b) Monitoring expected error rate



(c) Monitoring reject indicator

Figure 6.9: Performance of adaptive approaches for concept drift on the Reuters dataset.

6.6 Conclusions

This chapter was geared at providing techniques which allow efficient maintenance of classification performance in a long-term filtering application. We favor a methodology which adapts a filtering system to changes only if necessary because it permits comprehensive minimization of user effort. This methodology falls into two subtasks: change detection and appropriate adaptation. We have focused on detecting changes with little and no user feedback. Below, we summarize the results of the empirical evaluation:

- Both concept shift and drift in a document stream can be detected reliably with either little or no user feedback when monitoring any of the alternative performance measures presented: sample error rate, expected error rate, and virtual rejects.
- In particular, detecting changes with the reject indicator yields fast adaptation to changes with no user feedback for the detection process. Only adaptation of the classifier requires feedback for the provision of new training documents. In contrast, using the sample error rate, which requires some small amount of user feedback per batch, also yields fast adaptation but at the expense of some false alarms. In these cases, the adaptive filtering system is less efficient.
- In order to detect changes without user feedback, it is essential that the documents of a new topic differ in some recognizable way from the existing topics. As the non-relevant class of the filtering task is often heterogeneous, documents of a new relevant topic may have too much resemblance to the non-relevant class to be recognized as new. Then, changes may remain undetected.

With the aforementioned methodology, we have applied a framework for coping with dynamic aspects, which are inherent to many real-world classification tasks. We have carried out some basic experiments. Some relevant issues remain for future research:

- Derive quality indicators based on text properties to permit classifier-independent change detection which is, for example, insensitive to problems pertaining to class representations when classes are heterogeneous.
- Evaluate the quality control approach proposed with other classifiers such as the naïve Bayes classifier or support vector machines.
- Implement more elaborate adaptation strategies such as active learning to further reduce the need for user feedback. Note that semi-supervised learning with the single-prototype classifier will be considered in the following chapter.
- Analyze the effect of varying the batch size. Also, study alternatives to batch processing, such as evaluating quality indicators on the basis of a moving window of the most recently classified documents.

In this dissertation, we have confined ourselves on detecting changes in the content of document streams. Yet, it is also of paramount importance to recognize changes in user interests. Hence, the ultimate goal should be to cope with both types of changes.

Chapter 7

Empirical Evaluation

The aim of this chapter is two-fold. On the one hand, we evaluate the quality control approach set out in Chapter 6 on a new text collection, which was not used previously in this work. In contrast to the experiments with artificial change scenarios conducted so far, we now consider changes which occur naturally over time. On the other hand, we integrate the quality control approach with the semi-supervised text learning framework introduced in Chapter 4. Here, the objective is to further minimize the user effort needed to adapt a classifier to changes in the environment so as to maintain its classification performance. The integrated system is also evaluated using the new text collection.

7.1 System Integration

In Chapter 6, we have discussed methodologies for coping with dynamic aspects in classification tasks. We favor the approach which attempts to detect changes and adapts a classifier only if necessary to maintain classification performance since solely this approach permits comprehensive minimization of the user effort required. Experiments with simulated change scenarios have shown that changes can be detected reliably without user effort. Yet, adapting to changes still requires the provision of true class labels for some new documents. So far, we have considered only plain supervised learning approaches which learn a new classifier from scratch if changes have been detected: a solution which typically requires many labeled training examples. Hence, as already discussed in Section 6.4.3, there is enhanced potential to further reduce the user effort required. At this point, we consider deploying the semi-supervised learning framework from Chapter 4 to reduce the need for labeled training data.

As in the previous chapters, we use the enhanced version of the rainbow system and the Perl script, which implements the quality control unit of the adaptive filtering system as set out in Section 420 (p. 166).[1] Our rainbow version includes an implementation of the semi-supervised learning framework. And, since the quality control unit uses rainbow as the core classification unit, it has access to the semi-supervised learning framework.

[1]See Appendix A for more details on this software.

Using semi-supervised learners within the adaptive filtering system poses a new problem: the number of labeled training examples may be too small to allow cross-validation for the estimation of parameters needed to set up the monitoring process.[2] To remedy this problem, we use the same set of labeled documents in all the runs of the cross-validation process and split only the unlabeled training data into disjunct subsets. Since we do not know the true class labels of the unlabeled training examples, we evaluate the monitoring parameters on the basis of the predicted instead of the true class labels.

7.2 Datasets and Experimental Setup

To enable evaluation of the quality control approach with both plain supervised and semi-supervised learning on data containing changes which occur naturally over time, we have collected a new dataset over a period of almost one year. This text collection, referred to as the Yahoo! Dailynews dataset, consists of news articles from eight different categories.[3]

For the filtering task, we consider four of the eight categories as relevant: health, science, sports, and technology news. The remaining four categories, business, entertainment, politics, and world news, form the non-relevant class. Admittedly, this division hardly reflects common user interest. Users are typically not interested in broad categories such as health or science in their entirety. Rather, they are more likely to be interested in subtopics from these categories. Note, however, that division of the documents into categories according to subtopics requires labels which are not naturally available at the news source. We therefore stick to the division at the broad category level.

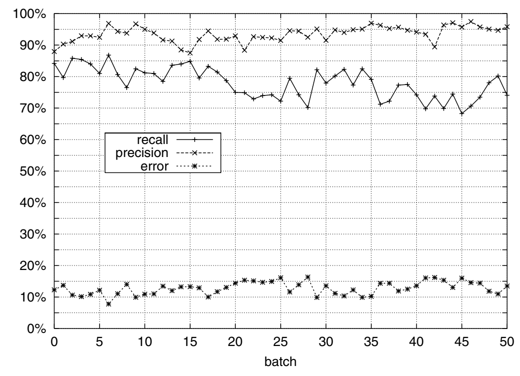
Our quality control methodology requires that performance measures be evaluated on batches of documents. We split the document stream into batches on a weekly basis. As the dataset was gathered over a period of 51 weeks, we obtained 51 batches, each containing, on average, nearly 500 documents, which are distributed almost evenly between the two relevance classes. The first batch of documents serves as the initial training data, from which all approaches are initially learned. Over the remaining 50 batches, classification performance is expected to decrease for reason of naturally occurring changes.

When tokenizing the news articles, HTML tags are skipped. For document representation, we remove stop words and words which occur fewer than five times in the training set or in less than three training documents. From the remaining set of features, the $3,000$ most informative words according to the information gain measure are selected as vocabulary. Initial experiments have shown that this yields reasonable results in terms of both recall and precision for the single-prototype classifier, which is used as the core text learner in all experiments to be conducted in this chapter.

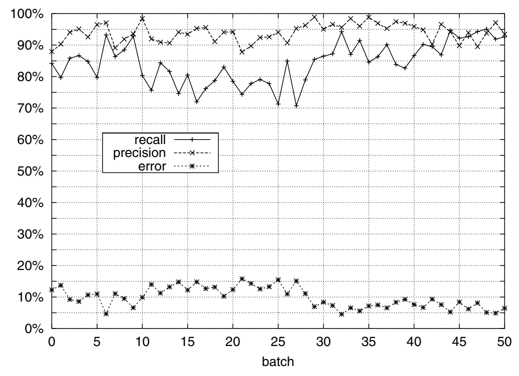
As in Chapter 6, we evaluate both the baseline approach, *no adaptation*, which remains unchanged after the initial learning phase, and the benchmark approach, which relearns the classifier after each batch. Concerning the adaptive approaches, we only consider the approaches which monitor the expected error rate and the virtual reject indicator as set out

[2]See Section 6.3 (pp. 149 ff.).

[3]See Appendix B for more information on this dataset.



(a) No adaptation to changes



(b) Relearning after each batch

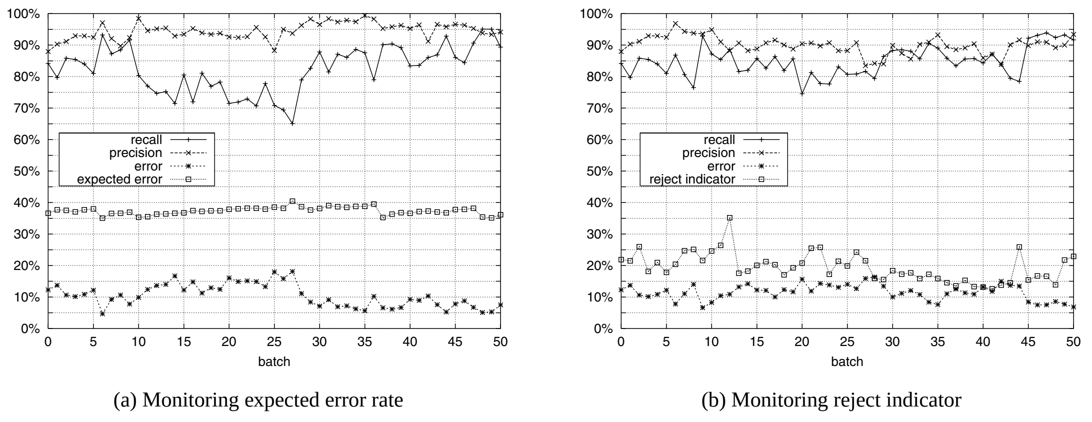
Figure 7.1: Performance of the baseline and benchmark approaches on the Yahoo! Dailynews dataset using the plain single-prototype classifier. As the results with the baseline approach show, we observe a slight decrease of recall over time, while precision increases slightly. This behavior is very likely due to concept drift in the stream of news articles. Note that the curves are fairly erratic as the results are obtained from only a single trial.

in Section 6.3.2 (pp. 153 ff.). Note that we do not evaluate the adaptive approach which monitors the sample error rate as this approach did not prove to be practical. Whenever a classifier is learned, we consider training documents from only the most recent batch. This is certainly the most simple adaptation strategy; see Section 6.4 (pp. 163 ff.).

When using the plain single-prototype classifier learned from all the documents of a batch, we conduct only one trial per approach. The reason for this is that, in this case, the batches can be created according to only a single chronological order. For approaches with the semi-supervised single-prototype classifier, however, the results presented are averages over ten trials. Here, we run multiple trials to decrease the influence of the random selection of the labeled training set on the classification performance. We randomly select 10% of the documents per batch as labeled training data. The remaining 90% of the documents per batch are used as unlabeled training data. Also, we run ten trials of the *no adaptation* approach with the plain single-prototype classifier with only 10% of the documents per batch as training data to evaluate the extent to which semi-supervised learning enhances filtering performance when labeled training data is scarce.

7.3 Results

Figure 7.1 shows the performance of the baseline and the benchmark approaches with the plain supervised single-prototype classifier. The performance of the approach with no adaptation after the initial learning phase reveals that the concept changes slightly over time: while precision barely increases, recall gradually decreases by some points. Specifically, within the first 15 batches, recall falls within a range of about 80% to 85%. Then, it decreases to about 72% over the next ten batches. Except for some phases with higher recall between batches 25 and 35 and towards the end, we observe recall values which are, on average, about ten points below the initially achieved performance. The reason for the drop in recall is a slight concept drift which occurred naturally over time. For example, between May and July 2000, i.e. around batches 17 through 29, there is



(a) Monitoring expected error rate (b) Monitoring reject indicator

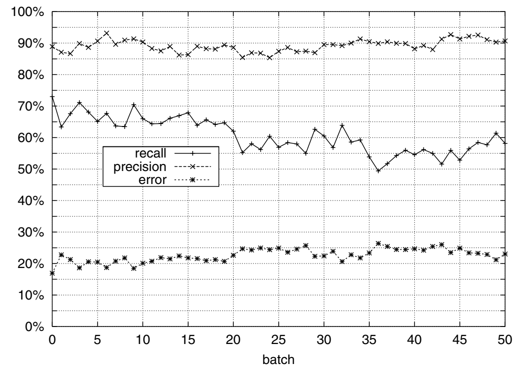
Figure 7.2: Performance of the adaptive approaches for naturally occurring concept change on the Yahoo! Dailynews dataset using the plain single-prototype classifier. Batches at which change are suspected and adaptation is triggered are marked by solid triangles. In most of the cases in which the adaptive approaches recommended adaptation, we actually observe a decrease in recall. Monitoring the reject indicator outperforms monitoring the expected error rate in terms of recall, but at the expense of precision.

an exceptionally large number of articles on HIV and AIDS in the health and science categories. And, the initial training data does not contain any documents related to this topic. Nonetheless, the concept change in the Yahoo! Dailynews dataset is much less radical than the simulated changes examined in Chapter 6.

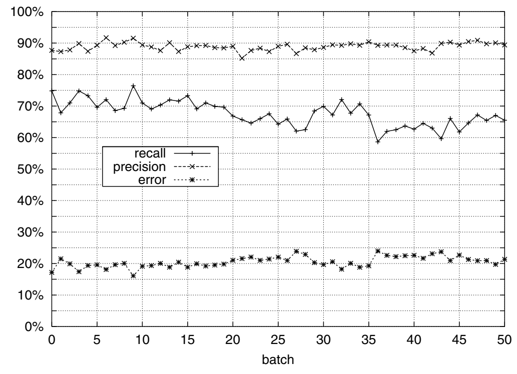
As it is relearned after each batch, the benchmark approach enhances recall for most of the batches. The improvement shows towards the end in particular, whereas between batches 10 and 27, the relearn approach struggles with learning the two relevance classes: in a few cases, recall is slightly worse than that of the baseline approach. The precision of the baseline and the benchmark approaches is very similar. Although the benchmark approach generally outperforms the baseline approach, the results also show that learning from only the most recent batch may not always be appropriate. So, additional training documents from older batches might be worth considering when adapting classifiers.

Figure 7.2 shows the performance of the two adaptive approaches which monitor the expected error rate and the virtual reject indicator, respectively. The adaptive approaches both detect changes quite reliably when performances deteriorates. Compared to the baseline approach, recall is enhanced through adaptation to the concept drift. With respect to the benchmark approach, the adaptive approaches achieve similar results. Yet, the user effort required has been reduced substantially. In terms of recall, monitoring the reject indicator is superior to monitoring the expected error rate: faster adaptation is achieved with fewer learning phases. Yet, monitoring the reject indicator deteriorates precision by some points on average when compared to the other three approaches; also see Table 7.1.

Note that the adaptive approach which monitors the expected error rate struggles when learning the relevance classes between batches 10 and 27. The same problem was observed for the benchmark approach in Figure 7.1. As a consequence, monitoring the expected error rates cannot be charged with this deficiency. Rather, the poor performance in this period is due to the simple adaptation strategy: the training documents in the most recent batch are not always fully representative of the documents to be classified subsequently. Unlike the benchmark approach and monitoring the expected error rate, moni-



(a) No adaptation to changes with SPC



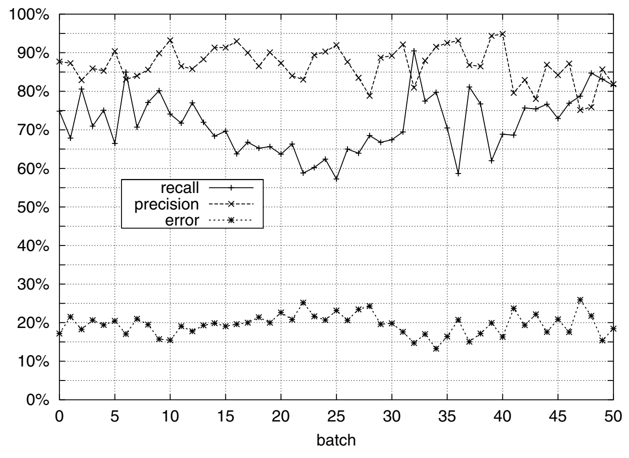
(b) No adaptation to changes with ssSPC

Figure 7.3: Performance of the baseline approach for naturally occurring concept change on the Yahoo! Dailynews dataset using the plain supervised single-prototype classifier (SPC) and the semi-supervised single-prototype classifiers (ssSPC) when only 10% of the training data are labeled.

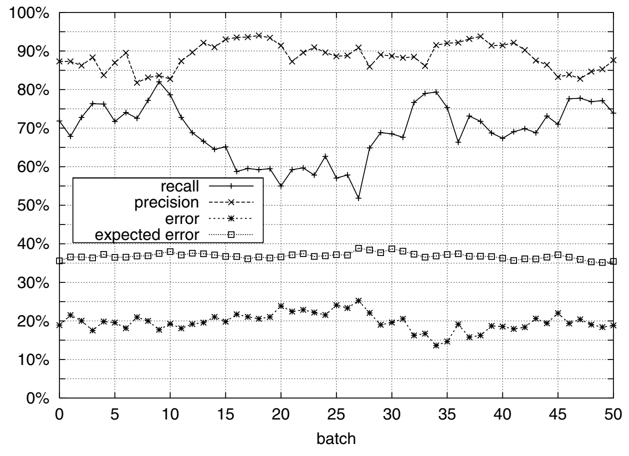
toring the reject indicator does not seem to have this difficulty in classifying documents in the aforesaid or any other period. As both adaptive approaches are able to recognize changes, monitoring the reject indicator very likely adapted to changes at batches whose documents were more representative for the situations at hand.

So far, we have learned classifiers in the quality control framework with plain supervised learning approaches. In the following, we turn to semi-supervised learning of classifiers. Figure 7.3 shows the performance of the baseline approach, i.e. no adaptation to changes after the initial learning phase, in combination with the plain supervised single-prototype classifier (SPC) and the semi-supervised single-prototype classifier (ssSPC) when only 10% of the documents in a batch are labeled by the user. Note that only ssSPC uses the remaining 90% of the documents per batch as unlabeled training data. As expected, the performance of the baseline approach with SPC decreases substantially when the number of labeled training documents is reduced to only 10% of the documents per batch; also see Figure 7.1 and Table 7.1. Semi-supervised learning helps increase performance in terms of recall, while precision and error rate hardly deteriorate at all. But, we do not reach the level of performance achieved by SPC with the entire batch as labeled training data.

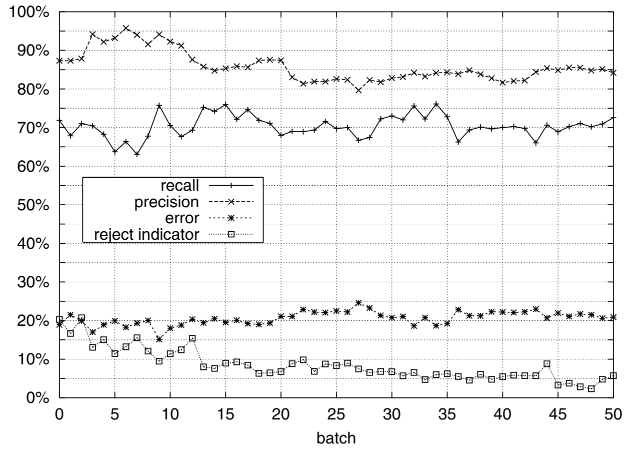
Figure 7.4 shows the performance of the benchmark approach and the two feasible adaptive approaches which monitor the expected error rate and the virtual reject indicator when using the semi-supervised single-prototype classifier with 10% of the documents per batch being labeled. Relearning after each batch outperforms the baseline approach in terms of recall most of the time. Similar to the benchmark approach with the plain supervised single-prototype classifier, the relearn approach struggles with learning the relevance classes from only the most recent batch, especially around batches 15 through 30. In this period, the documents of a batch are not as representative for the learning scenario as at other times. Obviously, semi-supervised learning cannot remedy this problem. Precision and error rate of the benchmark approach are similar to those of the baseline approach. So, recall can be enhanced while only slightly detracting from precision and accuracy; also see Table 7.1. Only towards the end does the precision of the benchmark approach drop substantially below baseline precision.



(a) Relearning after each batch



(b) Monitoring expected error rate



(c) Monitoring reject indicator

Figure 7.4: Performance of the benchmark approach and the adaptive approaches for naturally occurring concept change on the Yahoo! Dailynews dataset using the semi-supervised single-prototype classifier. Note that, here, we cannot mark the points of adaptation for the adaptive approaches because the curves are drawn from average results over ten trials, and the adaptation in each particular trial may have taken place at different batches.

As in the supervised case, monitoring the expected error rate with the semi-supervised learner is similar to the benchmark approach in that it struggles with learning the classes around batches 15 through 30. Although it adapts to changes nearly ten times per trial and enhances recall at the end, the overall performance is hardly superior to the baseline approach; see Table 7.1. We draw the same conclusion as in the supervised setting: the documents of some batches are not representative enough to enable reasonably accurate learning of classifiers. Therefore, finding a more representative training set by incorporating training examples not only from the most recent batch but also from even older batches may help enhance classification performance.

With only about two adaptations per trial, the adaptive approach which monitors the virtual reject indicator copes well with concept drift and maintains recall at a fairly constant level. Yet, maintaining recall is achieved at the expense of lower precision. In most trials, one of the adaptations takes place at either batch 11 or batch 12. Looking at the performance of the relearn approach reveals that this is the time at which the other approach begin to struggle. Through effective detection of changes and efficient adaptation fostered by semi-supervised learning, this approach handles the naturally occurring concept drift with exceptionally little user feedback; see Table 7.1.

Table 7.1 summarizes the results of all quality control methodologies—*no adaptation*, relearning, and the adaptive approaches which monitor the expected error rate and the virtual reject indicator—with both the supervised and the semi-supervised single-prototype classifiers as core text learner. In addition to the common performance measures, the table shows the fraction of training documents per batch for which true class labels are required, the number of adaptations triggered after the initial learning phase, and the total amount of user feedback necessary to realize each approach depending on the fraction of labeled training documents and the number of adaptations performed. As they are trained with much more labeled data, the first four supervised approaches outperform the other approaches. Semi-supervised learning helps enhance classification performance when labeled data is scarce. In both cases, the feasible adaptive approaches enable us to cope with the naturally occurring concept drift through a limited number of adaptations.

The performance curves which illustrate the approaches with only 10% labeled training documents are fairly deceptive. That is to say, the performance achieved varies greatly around the average values depicted. For example, average recall per trial of the baseline

Approach	Learner	Labeled	Recall	Precision	Error	Adapted	Feedback
No adaptation	SPC	100%	77.83%	93.60%	12.80%	0.0	0.0%
Relearning	SPC	100%	84.48%	94.09%	9.72%	50.0	100.0%
Expected error rate	SPC	100%	82.33%	94.64%	10.40%	9.0	18.0%
Reject indicator	SPC	100%	84.87%	89.99%	11.45%	5.0	10.0%
No adaptation	SPC	10%	60.47%	89.17%	22.70%	0.0	0.0%
No adaptation	ssSPC	10%	67.49%	88.95%	20.61%	0.0	0.0%
Relearning	ssSPC	10%	71.62%	86.86%	19.51%	50.0	10.0%
Expected error rate	ssSPC	10%	68.97%	88.67%	19.75%	9.9	2.0%
Reject indicator	ssSPC	10%	70.33%	85.87%	20.63%	2.1	0.4%

Table 7.1: Average results over all batches on the Yahoo! Dailynews dataset for all approaches.

approach may lie more than 20% below or above the mean. The reason for this is the randomly drawn training set, which is not large enough to always provide a sufficient number of examples for all the categories contained in the relevance classes. In particular, a 10% training sample contains nearly 50 documents. So, on average, each of the eight categories is represented by about six documents. Since these documents are drawn randomly, there may very well be training sets which contain hardly any documents of a specific category. Depending on whether such a category is relevant or non-relevant, either recall or precision may be low. Yet, if each category is sufficiently represented by training examples, recall and precision may both reach reasonably high levels.

7.4 Conclusions

In the following, we summarize the results of the empirical evaluation with both the plain supervised and the semi-supervised single-prototype classifiers on the Yahoo! Dailynews dataset, which contains naturally occurring concept drift.

Reducing training set size so that hand-labeling the training documents is practical causes classification performance to decrease substantially. Clearly, there is always a trade-off between the performance attainable and the willingness to provide feedback in terms of class labels. Semi-supervised learning helps narrow the gap between the two extreme performance levels achieved when only very few or a sufficient number of labeled training examples—which is often not feasible—are provided.

With both supervised and semi-supervised learning, monitoring the expected error rate and monitoring the virtual reject indicator enables efficient and fairly reliable detection of concept changes. On average, monitoring the reject indicator triggers fewer adaptations and should therefore be preferred.

Through systematic adaptation to changes, the adaptive approaches can cope with concept drift and enhance classification performance in terms of recall with substantially reduced user effort. Yet, the gain achieved is much smaller than that of the results with the simulated change scenarios examined in Chapter 6. The reason for this is that the naturally occurring concept change shown here is much more subtle than the artificial changes.

The performance curves of the relearn approach and the approach which monitors the expected error rate illustrate that the most recent batch is not always truly representative for learning the relevance classes. Hence, an essential issue for future research is to study more elaborate adaptation strategies which learn from training documents taken not only from the most recent batch but also from even older batches.

The results obtained with the semi-supervised learners show that performance varies greatly depending on the random selection of the training set. Although the average performance over several trials may be acceptable, it may not be so for any particular trial. And, when actually employing semi-supervised learners in practice, we only have a single trial. So, to ensure provision of a representative set of labeled training documents which would guarantee a higher level of classification performance, the training documents should be selected with care instead of being drawn randomly.

Chapter 8

Conclusions

In this chapter, we briefly summarize the main contributions of this dissertation to the areas of text classification and information filtering. Also, we discuss some of the open issues and possible future developments of our work in particular and of information filtering as a whole.

8.1 Contributions

In this work, we have viewed information filtering as a binary text classification problem, which is typically solved by means of supervised learning algorithms. We have identified the following three assumptions, which are often violated in real-world applications:

1. Constructing accurate classifiers through supervised learning algorithms typically requires a large number of labeled training examples, whose manual provision may easily become prohibitive.

2. Some of the most frequently applied text learning algorithms require that classes be homogeneous, so that each class can be represented by a single entity. Yet, in reality, the two classes of the filtering task tend to be heterogeneous.

3. Predicting the class labels of new examples through classifiers learned from some given training data assumes that the data sources do not change over time. In a long-term application, however, document sources are prone to change.

In the course of this dissertation, we evaluated the extent to which violations of these assumptions affect classification performance. In addition, we provided some solutions to remedy the problems observed.

With respect to the application goal of this dissertation set out in Chapter 1, compared to standard approaches, the techniques proposed enable both construction of reasonably accurate filtering systems and maintenance of classification performance in a long-term application at substantially reduced user effort.

Semi-Supervised Learning

Chapter 4 was dedicated to reducing the need for labeled training examples in supervised learning. Providing a sufficient amount of labeled training data for text learning problems is often prohibitive since users have to read and hand-label many documents: a tedious and time-consuming task. Having surveyed methodologies for reducing the amount of labeled training data needed to accurately learn text classifiers, we proposed a general framework for semi-supervised learning from both labeled and unlabeled training data. In contrast to labeled documents, unlabeled documents are often inexpensive and readily available in large quantities. Nigam *et al.* (2000) have previously studied semi-supervised learning with the naïve Bayes classifier. Our work generalizes this approach so that it can be used in combination with any learning algorithm. Empirical evaluations with three real-world text collections showed that, by additionally using unlabeled data, some semi-supervised learners require less labeled training data than their supervised variants to achieve the same level of classification performance. The semi-supervised framework proved to be especially suitable for simple averaging approaches like the single-prototype classifier and the naïve Bayes approach. Specifically, the semi-supervised single-prototype classifier outperformed the semi-supervised naïve Bayes classifier in most cases. Semi-supervised learning with support vector machines was only successful when performance of the plain support vector machine learned initially from only the labeled data was reasonably high. Simple instance-storing approaches like nearest-neighbor rules did not benefit from the semi-supervised learning framework because they do not generalize beyond the training data during the learning phase.

Learning Heterogeneous Classes

In Chapter 5, we shed some light on the learnability of heterogeneous classes by means of simple instance-averaging approaches. As they represent each class through a single entity, these approaches require that the classes to be learned be homogeneous. Certainly, homogeneity of classes is a matter of the granularity at which they are defined. In information filtering, we deal with extremely coarse-grained relevance classes: documents a user either likes or dislikes. Hence, the classes to be learned tend to be heterogeneous. But, how do instance-averaging approaches perform when the homogeneity assumption is violated? Empirical results on two text collections showed that, for common text learning tasks, the simple text learners considered may accurately learn heterogeneous classes. As a rule, classification performance decreased only slightly compared to that of more elaborate approaches, which exploit additional knowledge about the homogeneous subclass structure. This finding was confirmed in the information filtering experiment conducted in Chapter 7. Consequently, violations of the heterogeneity assumption for problems in high-dimensional feature space need not be as severe as examples in low-dimensional feature space suggest. And, using simple instance-averaging approaches to learn text classifiers should be considered even for complex learning tasks with heterogeneous classes. But, we can enhance classification performance through approaches which automatically uncover homogeneous subclasses in a heterogeneous class structure and which utilize the subclasses discovered in order to learn more effective classifiers.

Quality Control in Information Filtering

Chapter 6 dealt with dynamic aspects in the long-term application of information filtering systems. We proposed techniques which enable efficient maintenance of classification performance in the face of a changing environment. In particular, our methodology for quality control in information filtering attempts to detect changes without expensive user feedback and adapts a filtering system only if necessary to maintain classification performance. Therefore, this approach permits comprehensive minimization of the user effort required. Experiments with artificial change scenarios showed that concept drift and shift in a document stream can be detected reliably. And, subsequent learning of a new classifier with new training data enhances performance. In addition, experiments with naturally occurring changes conducted in Chapter 7 proved that our change detection approach also works reliably in a more realistic setting.

8.2 Limitations and Future Work

In the course of this dissertation we conducted many experiments so as to empirically evaluate the approaches proposed. Yet, these approaches call for further experimentation:

- In Chapter 4, we carried out experiments with the semi-supervised learning framework in combination with the single-prototype classifier, the naïve Bayes classifier, support vector machines, and the nearest-neighbor rule. How do other supervised learners such as decision tree or rule set induction algorithms perform in the semi-supervised framework?
- In Chapters 6 and 7 our quality control framework was evaluated only with the single-prototype classifier. How do other learning algorithms perform in this setting?

Although our work provides some solutions to the problems outlined above, it, in turn, raises other issues which could be explored in future research:

- In some experiments with the semi-supervised learning framework, we observed performance degradation from using too large a feature set. Yet, the application of class-specific feature selection techniques like the information-gain measure, which is often used in text classification, may not be statistically reliable when labeled training data is scarce. In addition, they cannot be applied to features which occur only in the unlabeled data. So, an essential issue is to explore class-independent feature selection techniques which are common in unsupervised learning.
- Our semi-supervised learning framework describes a certain way of learning from both labeled and unlabeled data. As there are other approaches to doing this, such as self-training, an interesting issue is to evaluate whether hybrid approaches can further enhance classification performance.

- Although we have seen that heterogeneous classes can be learned with simple instance-averaging approaches, there is still potential to improve classification performance through more elaborate approaches. In particular, representing each class by multiple entities rather than a single entity could be studied. Here, clustering approaches could be applied to uncover homogeneous subclasses in the heterogeneous class structure. Knowledge about the subclasses could then be incorporated into learning a classifier so as to enhance classification performance.

- So far, in the experiments with our quality control framework, we only considered learning a new classifier from scratch on the basis of a certain set of the most recent examples. Specifically the results in Chapter 7 showed that this adaptation strategy does not always suffice to accurately learn a new classifier. So, a vital issue is to implement more elaborate adaptation strategies which may incorporate a larger set of examples and may build on existing classifiers.

- In Chapter 7, we employed semi-supervised learning to further reduce the amount of user feedback needed for an adaptive filtering system. Here, the heterogeneous class structure caused a problem when the randomly selected labeled training set was not large enough to ensure that each subclass was represented by at least a few examples. In these cases, performance substantially deteriorated compared to plain supervised learning with a sufficiently large training set. Therefore, to ensure provision of a representative training set, which is necessary to guarantee a high level of classification performance, the training documents should be selected with care rather than drawn randomly.

In this dissertation, we have confined ourselves to enhancing some relevant aspects of text classification to improve information filtering. The following issues for future research aim at broadening this restricted view of adaptive information filtering:

- Our quality control approach enables detection of changes in the document stream, i.e. in the environment. Yet, it is also of paramount importance to recognize changes in user interests. Hence, the ultimate goal should be to create an adaptive filtering system that can cope with changes both in the environment and in user interests.

- Rather than considering only textual documents and viewing information filtering as a binary text classification task, we will have to process general multi-media documents. This necessitates generation of classifiers for different types of information, which may be seen as a task of the emerging field of *multi-media mining*.

Appendix A

Software

The experiments conducted in the course of this dissertation were carried out with an enhanced version of the rainbow text classification system. The conventional rainbow system and also the libbow library on which it is based were written by McCallum (1996). Our rainbow version includes, among other things, an implementation of the single-prototype classifier and an interface to the SVM^{light} implementation of support vector machines written by Joachims (1999a). Also, we have added implementations of the semi-supervised learning and self-training frameworks as set out in Chapter 4. The source code of all components of our rainbow version is written in C and has been compiled and tested on Linux and Solaris platforms. In addition to our enhanced rainbow system, we provide a Perl script which implements the adaptive classification environment for coping with dynamic aspects in classification tasks as set out in Chapter 6. In the following, we describe the programs and components in more detail.

Libbow

The *bag-of-words library*, libbow, is useful for writing statistical language modeling, text retrieval, classification, and clustering programs.[1] The sources are publicly available from Andrew McCallums's home page at `http://www.cs.cmu.edu/~mccallum/bow`. We have implemented our system based on libbow Version 0.95 dated November 26, 1999.

The library provides many facilities which are essential for implementing text classification systems. For example, this includes very efficient data structures and functions for reading and parsing text files, generating *bag-of-words*-style vector representations of the text files according to several different methods, pruning the vocabulary by word counts or information gain and removing common English stop words, learning different classification models from training documents such as a K-NN classifier, a tfidf classifier, or a naïve Bayes classifier with various different options, and classifying both test documents and new documents with the classifiers generated. A complete overview of the functionality can be found at the libbow home page.

[1]See McCallum (1996).

Rainbow

The rainbow front-end is a program for performing statistical text classification based on libbow. Its source code, as well as that of other front-ends for document retrieval and document clustering, is included in the libbow distribution. A brief tutorial for using rainbow is available at `http://www.cs.cmu.edu/~mccallum/bow/rainbow`. Online information on how to use the rainbow features with their command-line options can be obtained by invoking rainbow with the `--help` option.

As a rule, using rainbow to perform text classification tasks is a two-step process: first a set of documents is read and written to disk as a compact model containing document statistics and, second, this model is used to output diagnostics of the documents read or to perform text classification of test or new documents. Rainbow provides the entire libbow functionality relevant for text classification through command-line options. To evaluate the rainbow output, the libbow package provides some Perl scripts for access of different classification performance measures.

SVM–Light

SVM^{light} is an implementation of support vector machines written by Joachims (1999a), which is geared for pattern recognition problems. It is publicly available at Thorsten Joachims's home page at `http://ais.gmd.de/~thorsten/svm_light`. We are using SVM^{light} version 3.01 dated October 26, 1999.

With the author's permission, we have slightly modified the source code of SVM^{light} so that its output conforms with that of the rainbow system. Basically, this involved writing all output to the error stream rather than to the standard output. In addition, we have changed the main functions of the learning and classification modules to general functions which can be invoked from within other programs. In particular, the entire functionality of the SVM^{light} package is compiled into a library which is linked to our rainbow system.

Rainbow Enhancements

The rainbow version used for the experiments in this dissertation consists mainly of four new features: an implementation of the single-prototype classifier, an interface to the SVM^{light} package, and implementations of the semi-supervised learning and the self-training frameworks. Information on how to use these features can be obtained online by invoking our rainbow system with the `--help` option.

Implementation of the Single-Prototype Classifier

We have implemented the single-prototype classifier as set out in Section 197 (pp. 58 ff.), basically building on existing parts of the rainbow source code. In particular, we used the document vector weighting options of the K-NN classifier implementation, which follows the weighting scheme of the Smart information retrieval system as set out in Section 3.2.4 (pp. 44 ff.). The learning and classification facilities of the single-prototype classifier are based on the implementation of the tfidf classifier.

Interface to SVM-Light

We provide an interface to the SVM^{light} package which enables us to use its learning and classification modules from within the rainbow system. The SVM^{light} section of rainbow's online help lists the command-line options which can be used for learning support vector machines with the SVM^{light} package. Please refer to the SVM^{light} documentation for more details on using its core learning and classification modules.

In particular, the rainbow facilities for indexing documents are used to construct document vectors according to the same weighting options employed for the single-prototype classifier. The interface invokes the SVM^{light} learning and classification modules with suitably modified data structures. The return values are, in turn, appropriately modified so that they can be processed within rainbow.

Note that our SVM^{light} interface enhances the basic SVM^{light} functionality as it permits handling classification problems with $k > 2$ classes through composition into k binary classification tasks. For the combination of the binary classifiers' responses, we provide a module for confidence mapping as set out in Section 3.3.4 (pp. 75 ff.).

Implementation of the Semi-Supervised Learning Framework

We have implemented the semi-supervised learning framework introduced in Chapter 4 based on Kamal Nigam's implementation of his semi-supervised naïve Bayes algorithm,[2] which is included in the libbow source code as the *em* classifier. As a generalized version of the *em* classifier, the semi-supervised learning framework is referred to as the *emx* classifier within our rainbow system. The corresponding section of rainbow's online help provides detailed information on how to use the semi-supervised learning framework.

The main difference between the *emx* classifier and the basic *em* classifier is that, instead of being hard-wired to the naïve Bayes classifier, *emx* permits any classifier registered in the rainbow system to be specified as the base learning algorithm. In addition, it provides different options for converting the base classifier's responses for the unlabeled data into either hard, probabilistic, or possibilistic class labels as set out in Section 312 (p. 96).

Implementation of the Self-Training Framework

For comparison with the semi-supervised learning framework, we have also implemented the self-training framework as set out in Section 300 (p. 90). Like the semi-supervised

[2]See Nigam *et al.* (2000).

learning framework, the self-training framework allows any other classifier to be used as base learning algorithm. Unlike semi-supervised learning, however, unlabeled data is always given hard labels. The number of unlabeled examples which are converted to labeled training examples at each iteration can be specified through a command-line option. Refer to the section of the *self* classifier in rainbow's online help for further information on how to use the self-training framework.

ACE: The Adaptive Classification Environment

The adaptive classification environment, ACE, is a Perl script that implements the quality control unit of the adaptive information filtering framework as set out in Section 420 (p. 166). The goal of ACE is to enable experiments for the evaluation of the quality control techniques introduced in Chapter 6. Note that, in its current form, ACE is not designed to be used as a part of an adaptive filtering system for everyday use.

ACE uses the enhanced rainbow system as the underlying core classification unit. As set out in Section 420 (p. 167), text classification problems can be solved with three different methods: no adaptation after the classifier as been learned initially, relearning after each batch irrespective of whether changes have occurred, and adapting to changes only if necessary to maintain classification performance. The adaptive approach necessitates the detection of changes. For this purpose, ACE allows continuous monitoring of performance characteristics as described in Section 6.3 (pp. 149 ff.). In addition to the common performance measures recall, precision, and error rate, ACE permits computation of the alternative performance measures introduced in Section 6.3.2 (pp. 153 ff.): sample error rate, expected error rate, and virtual rejects.

For the evaluation of the quality control unit, ACE assumes that batches of documents are provided rather than accepting documents from a real document stream. A batch is simply a file which contains references to documents together with their true class label. Note that the true class labels are used solely for the purpose of evaluation. They are not used for monitoring purposes within the quality control unit unless theoretical benchmarks are considered.

Batches can be either constructed manually or generated automatically according to a configuration script which determines how relevant and non-relevant subclasses are mixed over time to form sets of relevant and non-relevant documents. As a rule, automatic batch generation is used to set up experiments with simulated change scenarios as in Chapter 6, whereas manual batch provision is used to run experiments with naturally occurring changes as in Chapter 7.

Appendix B

Text Corpora

In the course of this work, we have used five different text corpora: the 20 Newsgroups dataset, the WebKB dataset, a modification of the Reuters dataset, a subset of the TREC dataset, and the Yahoo! Dailynews dataset. The collections contain documents in English. The first four datasets have been widely used to evaluate approaches to automating text classification and related tasks. The fifth dataset has been collected especially for this work over a longer period of time so as to evaluate the extent to which the ability of coping with dynamic aspects is essential in real-world applications. Note that only the first three datasets are publicly available. Table B.1 summarizes the main characteristics of the five text corpora. Below, the datasets are described in more detail.

Name	Document Type	Classes	Documents	Unique Words
20 Newsgroups	newsgroups articles	20	$19,997$	$119,504$
WebKB	university web pages	7	$8,282$	$91,503$
Reuters	business news articles	3	$21,559$	$42,520$
TREC	business news articles	10	$7,026$	$76,971$
Yahoo! Dailynews	various news articles	8	$25,042$	$94,352$

Table B.1: Overview of the five text corpora used for experiments in this dissertation.

The 20 Newsgroups Dataset

The 20 Newsgroups dataset, assembled by Ken Lang, is a collection of approximately $20,000$ newsgroup articles divided almost evenly among the 20 different UseNet discussion groups shown in Table B.2. The dataset is publicly available from Jason Rennie's home page at `http://www.ai.mit.edu/~jrennie/20_newsgroups.html`. It has been widely used for experiments in text learning applications.[1]

Except for a small fraction of the document collection, each article belongs to exactly one newsgroup. So, the task is typically to classify an article into the one of the 20 newsgroups

[1]The dataset was first cited by Lang (1995).

comp.graphics	*alt.atheism*
comp.os.ms-windows.misc	*soc.religion.christian*
comp.sys.ibm.pc.hardware	*talk.religion.misc*
comp.sys.mac.hardware	
comp.windows.x	*talk.politics.guns*
	talk.politics.mideast
sci.crypt	*talk.politics.misc*
sci.electronics	
sci.med	*rec.autos*
sci.space	*rec.motorcycles*
	rec.sport.baseball
misc.forsale	*rec.sport.hockey*

Table B.2: UseNet newsgroups used as categories in the 20 Newsgroups dataset grouped by related topics.

to which it was posted. About 4% of the articles were cross-posted among two of the newsgroups. For the sake of simplicity, we ignore this fact. Note that when processing the newsgroup articles, the newsgroup headers should be removed because they contain the correct class labels and would lead to perfect classification of the articles.

The WebKB Dataset

The WebKB dataset was collected by the text learning group at the Carnegie Mellon University as part of an effort to create a crawler that explores previously unseen computer science departments and classifies web pages into a knowledge-base ontology.[2] It is publicly available at `http://www.cs.cmu.edu/~webkb/`.

The text corpus contains $8,282$ web pages gathered from university computer science departments, including the entirety of four departments, and also an assortment of pages from other universities. The pages are divided into seven categories: *student, faculty, staff, course, project, department,* and *other*. Table B.3 gives an overview of the seven categories and the number of documents assigned to them. Typically, the task is to classify a web page into the appropriate one of the seven categories. Yet, in the experiments conducted in this dissertation, only the four most populous non-other categories are used: *student, faculty, staff,* and *course*.

Category	4 Universities	Miscellaneous	Total
student	558	1,083	1,641
faculty	153	971	1,124
course	244	686	930
project	86	418	504
staff	46	91	137
department	4	178	182
other	3,071	693	3,764
Total	4,162	4,120	8,282

Table B.3: Categories of the WebKB collection of university web pages and the number of documents they contain.

[2]See Craven *et al.* (1998).

The Reuters Dataset

The well-known and frequently used Reuters-21578 (1997) text categorization test collection, distribution 1.0, is available from David D. Lewis's professional home page at `http://www.research.att.com/~lewis/reuters21578.html`.

The conventional Reuters-21578 collection contains $21,578$ news stories. Originally, there are different sets of categories according to, for example, organizations, people, places, and topics. Each document is assigned to one or more of these categories. Note that in many evaluations of text classification applications and related tasks, only some of the most populous categories are actually learned.[3]

Although it is recommended that the predefined splits into test and training sets be employed, we use a modified version of the Reuters-21578 collection—in this work referred to as the Reuters dataset—to evaluate approaches which aim at coping with dynamic aspects. In particlar, we consider only two of the predefined categories: acquisitions (*ACQ*) and earnings (*EARN*). Note that 19 documents belong to both of these categories. These documents are omitted since we focus on text classification tasks where each document is assigned to exactly one class. All documents which are not assigned to either of the two aforementioned categories are combined to form a third category, referred to as *OTHERS*. Hence, we obtain a text corpus with $21,559$ documents, each belonging to exactly one of the three categories shown in Table B.4.

Category	Description	Documents
ACQ	acquisitions	$2,429$
EARN	earnings	$3,968$
OTHERS	all other categories	$15,162$
	Total	$21,559$

Table B.4: Categories of the modified Reuters dataset. Note that 19 documents assigned to both ACQ and EARN have been omitted.

The TREC Dataset

The TREC dataset used in this dissertation is, in fact, a subset of the data used for different tasks at the *Sixth Text REtrieval Conference (TREC-6)*.[4] We have obtained access to the data by taking part in the TREC-6 filtering track.[5]

The TREC-6 data volumes consist of several gigabytes of English business news from various sources such as the Associated Press newswire (AP), the Foreign Broadcast Information Service (FBIS), and the Wall Street Journal (WSJ). For the filtering task, articles have been assigned to none, one, or more of 47 different topics. In the experiments conducted in this dissertation, we consider only articles which have been assigned to exactly one of the ten topics shown in Table B.5, yielding a text corpus with $7,026$ documents.

[3]For example, see Joachims (1997a).
[4]See Voorhees and Harman (1998).
[5]See Bayer *et al.* (1998).

Category	Description	Documents
001	Antitrust Cases Pending	400
003	Joint Ventures	842
004	Debt Rescheduling	355
005	Dumping Charges	483
006	Third World Debt Relief	528
054	Satellite Launch Contracts	343
100	U.S. Protectionist Measures	808
128	Privatization of State Assets	635
142	Impact of Gov. Regulated Grain Farming on Intl. Relations	2,422
180	Ineffectiveness of U.S. Embargoes/Sanctions	210
	Total	7,026

Table B.5: Categories selected from the TREC dataset for experiments in this dissertation and the number of documents they contain.

The Yahoo! Dailynews Dataset

We have assembled the Yahoo! Dailynews dataset from Reuters news articles provided at `http://dailynews.yahoo.com` for the news categories *business, entertainment, health, politics, science, sports, technology,* and *world*. The aim of providing this dataset is to enable evaluation of the extent to which real-world changes in data streams affect the classification performance of the filtering task.

The articles were gathered daily over a period of 51 weeks from Monday, January 3, 2000, until Sunday, December 24, 2000. On regular working days, there were about ten articles per category and day. On weekends and holidays, some categories contained fewer articles. In total, the Yahoo! Dailynews dataset contains some more than $25,000$ news articles, which are distributed among the eight categories as shown in Table B.6.

Category	Description	Documents
bs	business news	3,394
hl	health news	2,754
pl	politics news	3,458
re	entertainment news	3,013
sc	science news	2,561
sp	sports news	3,557
tc	technology news	2,747
wl	world news	3,558
	Total	25,042

Table B.6: News categories contained in the Yahoo! Dailynews dataset and the number of documents they contain.

Notation

Abbreviations

1-NN	One-nearest-neighbor rule
1-NP	One-nearest-prototype rule
EM	Expectation-Maximization
K-NN	K-nearest-neighbor rule
MAP	Maximum a posteriori
ML	Maximum likelihood
NB	Naïve Bayes classifier
SPC	Single-prototype classifier
ssNB	Semi-supervised naïve Bayes classifier
ssSPC	Semi-supervised single-prototype classifier
ssSVM	Semi-supervised support vector machine
SVM	Support vector machine

Symbols

$\lvert \cdot \rvert$	Cardinality of a set
$\lVert \cdot \rVert$	Euclidean length of a vector
$\mathbf{a}^T$	The transpose of vector $\mathbf{a}$
$\arg\max_{x \in X} f(x)$	Returns the value of $x \in X$ that maximizes $f(x)$
c	Class label or, in unsupervised learning, number of clusters
c_i	Class label for the i-th class
C_t^+	One-sided upper cumulative sum at time t
C_t^-	One-sided lower cumulative sum at time t
$\mathcal{C} = \{c_1, \ldots, c_k\}$	Set of class labels
$d \in \mathcal{D}$	Document (in naturally occurring textual form)
$\mathbf{d} = \rho(d) \in \mathcal{R}$	Document vector, representation of document d
$\mathbf{d}_{tf} = (tf_d(t_j))^T$	Term frequency vector of document d
$\mathbf{d}^l$	In semi-supervised learning, labeled document vector
$\mathbf{d}^u$	In semi-supervised learning, unlabeled document vector

$D = \{\mathbf{d}_1, \ldots, \mathbf{d}_n\}$	Set of training document vectors
$D^\star = \{d_1, \ldots, d_n\}$	Set of training documents
$D_c \subset D$	Subset of training document vectors with class label c
D_{non}, D_{rel}	Subsets of non-relevant and relevant training document vectors as used in the information filtering task
D^l, D^u	In semi-supervised learning, labeled and unlabeled sets of training document vectors
$\mathcal{D}$	Document space (in naturally occurring textual form)
$\mathrm{dist}(\mathbf{d}_1, \mathbf{d}_2)$	Euclidean distance between two document vectors
e_{expected}	In quality control, expected error rate
e_{sample}	In quality control, sample error rate
$E[x]$	The expected value of x
$f_\psi : \mathcal{D} \mapsto \{0, 1\}$	Information filtering with respect to a given user interest ψ
$\mathrm{Gain}(t)$	Information gain measure of term t
$h : \mathcal{D} \mapsto \mathcal{C}$	Text classification: mapping from the document space onto the set of class labels
$H : \mathcal{R} \mapsto \mathcal{C}$	Text classification: mapping from the document representation space onto the set of class labels; typically refers to the hypothesis (classifier) constructed by a learning algorithm
$H(\mathbf{d})$	Class label predicted by hypothesis (classifier) H for document vector $\mathbf{d}$
$H_s : \mathcal{R} \mapsto \mathcal{S}$	Text classification with subclasses (first step)
$H_c : \mathcal{S} \mapsto \mathcal{C}$	Text classification with subclasses (second step)
$\mathcal{H}$	Hypothesis space
$\mathrm{idf}(t)$	Inverse document frequency of term t
k	Number of classes
K	Number of documents considered in K-NN classification rule
$\mathrm{MI}(c, t)$	Mutual information between class c and term t
m	Vocabulary size, i.e. the number of index terms used to represent documents
$\hat{m}$	Size of unpruned vocabulary (prior to dimensionality reduction)
n	Number of documents in a given set, e.g. the training set
n_{c_i}	Number of documents with class label c_i
$n(t)$	Number of documents in which term t appears at least once
$\bar{n}(t)$	Number of documents in which term t does not appear
$n_{c_i}(t)$	Number of documents with class label c_i in which term t appears at least once
$\bar{n}_{c_i}(t)$	Number of documents with class label c_i in which term t does not appear
n_l	In semi-supervised learning, number of labeled documents
n_u	In semi-supervised learning, number of unlabeled documents

$\mathbf{n}_j$	The j-th nearest neighbor to some query document
non	Class label for non-relevant documents
$\mathcal{N}$	User interest space
$\mathbb{N}$	The set of natural numbers (including 0)
$O(\cdot)$	Notation for asymptotic complexity analysis
$\mathbf{p}_{c_i}$	Prototype vector for class c_i
$\mathrm{P}(x)$	The probability of event x
$\mathrm{P}(x\|y)$	The probability of event x given y
$\mathcal{P}$	Profile space
q	Number of homogeneous subclasses or queries
rel	Class label for relevant documents
$\mathcal{R}$	Document representation space
$\mathbb{R}$	The set of real numbers
$\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$	The set of non-negative real numbers
s_i	Subclass label
$\mathcal{S} = \{s_1, \ldots, s_1\}$	Set of subclass labels
$\mathrm{sim}(\mathbf{d}_1, \mathbf{d}_2)$	Similarity measure between two document vectors; typically the cosine similarity is assumed
$\mathrm{sign}(x)$	Returns 1 if $x > 0$ and -1 otherwise
$\mathrm{SNR}(t)$	Signal-to-noise ratio of term t
$t \in \mathbb{N}$	Point in time
$t \in \mathcal{V}$	Index term
tf	Number of occurrences of all terms in all documents within a given text collection
$tf(t)$	Number of occurrences of term t in all documents within a given text collection; term frequency
$tf_d(t)$	Number of occurrences of term t in document d
$\mathrm{tfidf}(t)$	Term frequency/inverse document frequency measure of term t
$T : \mathcal{D} \mapsto \mathcal{C}$	Target function, defined for training documents
$T : \mathcal{R} \mapsto \mathcal{C}$	Target function, defined for training document vectors
u_{ij}	Membership degree of pattern $\mathbf{x}_j$ or document $\mathbf{d}_j$ in cluster i
$\mathbf{U} = (u_{ij})$	Membership matrix for all patterns or document vectors
$\mathbf{U}^l = (u^l_{ij})$	In semi-supervised learning, membership matrix for labeled data
$\mathbf{U}^u = (u^u_{ij})$	In semi-supervised learning, membership matrix for unlabeled data
$\mathbf{v}_j$	Representation for cluster j
$V = \{\mathbf{v}_1, \ldots, \mathbf{v}_c\}$	Set of cluster representations
$\mathcal{V} = \{t_1, \ldots, t_m\}$	Vocabulary, set of index terms used to represent documents
$\hat{\mathcal{V}} = \{\hat{t}_1, \ldots, \hat{t}_{\hat{m}}\}$	Unpruned vocabulary (prior to dimensionality reduction)
$VC(\mathcal{H})$	Vapnik-Chervonenkis dimension of hypothesis space $\mathcal{H}$
w_j	Weight of index term t_j

$w_{i,j}$	Weight of index term t_j in document vector $\mathbf{d}_i$
$w(d,t)$	Weight of index term t in document d
$\mathbf{w}$	Weight vector
$\mathbf{x} \in \mathbb{R}^m$	Pattern (used in a more general context than text classification)
$\mathbf{x}^l$	In semi-supervised learning, labeled pattern
$\mathbf{x}^u$	In semi-supervised learning, unlabeled pattern
$X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$	Set of training patterns
X^l, X^u	In semi-supervised learning, labeled and unlabeled sets of patterns
$y = T(\mathbf{d})$	Target class label of document vector $\mathbf{d}$
y_j^l	In semi-supervised learning, known class label of pattern $\mathbf{x}_j$ or document vector $\mathbf{d}_j$
y_j^u	In semi-supervised learning, predicted class label of pattern $\mathbf{x}_j$ or document vector $\mathbf{d}_j$
$\mathbf{y}$	Label vector
$\alpha, \beta, \gamma, \delta$	Parameters (context-dependant)
$\delta(\cdot,\cdot)$	Identity operator, returns 1 if its two arguments are the same and 0 otherwise
$\kappa_h : \mathcal{N} \times \mathcal{D} \mapsto [0,1]^l$	Human comparison function between user interest and document
$\kappa_s : \mathcal{P} \times \mathcal{R} \mapsto [0,1]^l$	System comparison function between user interest representation (profile) and document vector
$\lambda \in [0,1]$	In semi-supervised learning, weight of unlabeled data
$\pi : \mathcal{N} \mapsto \mathcal{P}$	Profile acquisition function
$\psi \in \mathcal{N}$	Information need, user interest
$\rho : \mathcal{D} \mapsto \mathcal{R}$	Document representation function
$\tau_h : [0,1]^l \mapsto \{0,1\}$	Human decision function, applied to the response of κ_h
$\tau_s : [0,1]^l \mapsto \{0,1\}$	System decision function, applied to the response of κ_s
θ	Threshold parameter
θ_c	Threshold parameter for class c
$\theta_c = \mathrm{P}(c)$	In probabilistic models, prior probability of class c
$\hat{\theta}_c = \mathrm{P}(c)$	Estimate of the prior probability of class c
$\theta_{t\|c} = \mathrm{P}(t\|c)$	Probability of observing term t given class c
$\hat{\theta}_{t\|c} = \mathrm{P}(t\|c)$	Estimate of the probability of observing term t given class c
$\boldsymbol{\theta}$	In probabilistic models, parameter vector comprising all required probabilities
ν	In quality control, characteristic value to be monitored
ν_{reject}	In quality control, reject indicator
$\chi^2(t)$	Chi-squared statistic of term t

Bibliography

Ackoff, R. L. (1967). Management misinformation systems. *Management Science 14*(4), B147–B156.

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning* **6**, 37–66.

Allan, J. (1996). Incremental relevance feedback for information filtering. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* Zürich, Switzerland, pp. 270–278.

Allan, J., Carbonell, J., Doddington, G., Yamron, J., and Yang, Y. (1998). Topic detection and tracking pilot study final report. In *Proceedings of the Broadcast News Transcription and Understanding Workshop (Sponsored by DARPA).*

Allen, R. B. (1990). User models: Theory, method, and practice. *International Journal on Man-Machine Studies* **32**, 511–543.

Apté, C., Damerau, F., and Weiss, S. M. (1994). Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems 12*(3), 233–251.

Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T. (1995). Webwatcher: A learning apprentice for the word wide web. In *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments,* Stanford, pp. 6–12.

Balabanoviĉ, M. (1997). An adaptive web page recommendation service. In *Proceedings of the First International Conference on Autonomous Agents,* Marina del Rey, CA, pp. 378–385.

Bayer, T., Mogg-Schneider, H., Schäfer, H., and Renz, I. (1998). Daimler-Benz Research: System and experiments. routing and filtering. In *Proceedings of the Sixth Text REtrieval Conference (TREC-6),* Gaithersburg, MD, pp. 329–346. National Institute of Standards and Technology.

Belkin, N. J. and Croft, W. B. (1992). Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM 35*(12), 29–38.

Bennett, K. P. and Demiriz, A. (1998). Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems 12,* pp. 368–374.

Bensaid, A. M. and Bezdek, J. C. (1998). Semi-supervised point-prototype clustering. *International Journal of Pattern Recognition and Artifical Intelligence 12*(5), 625–643.

Bensaid, A. M., Hall, L. O., Bezdek, J. C., and Clarke, L. P. (1996). Partially supervised clustering for image segmentation. *Pattern Recognition 29*(5), 859–871.

Berry, M. W., Dumais, S. T., and O'Brien, G. W. (1995). Using linear algebra for intelligent information retrieval. *Society for Industrial and Applied Mathematics Review 37*(4), 573–595.

Bezdek, J. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum Press.

Bezdek, J. (1987). Convergence theory for fuzzy c-means: Counterexamples and repairs. *IEEE Transactions on System, Man, Cybernetics SMC-17*(5), 873–877.

Bezdek, J., Reichherzer, T., Lim, G., and Attikiouzel, Y. (1996). Classification with multiple prototypes. In *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, pp. 626–632.

Bezdek, J., Reichherzer, T., Lim, G., and Attikiouzel, Y. (1998). Multiple-prototype classifier design. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews 28*(1), 67–79.

Billsus, D. and Pazzani, M. J. (1999). A personal news agent that talks, learns and explains. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99),* Seattle, Washington, pp. 268–275.

Blair, D. (1992). Information retrieval and the philosophy of language. *The Computer Journal 35*(3), 200–207.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Annual Conference on Computational Learning Theory (COLT-98),* pp. 92–100.

Blum, A. L. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence 97*(1–2), 245–271.

Board, R. and Pitt, L. (1989). Semi-supervised learning. *Machine Learning* **4**, 41–65.

Boser, B. E., Guyon, I. M., and Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory,* pp. 144–152. ACM.

Box, G. and Luceño, A. (1997). *Statistical Control by Monitoring and Feedback Adjustment*. Wiley-Interscience.

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Madison, WI, pp. 43–52. Morgan Kaufmann Publisher.

Breiman, L. (1996). Bagging predictors. *Machine Learning* **26**, 123–140.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth Inc.

Callan, J. (1997). Characteristics of text. `http://www-ciir.cs.umass.edu/~allan/cs646-f97/char_of_text.html`. (as of February 6, 1997).

Castelli, V. and Cover, T. M. (1995). On the exponential value of labeled samples. *Pattern Recognition Letters 16*(1), 105–111.

Castelli, V. and Cover, T. M. (1996). The relative value of labeled and unlabeled samples in pattern recognition with an unknown mixing parameter. *IEEE Transactions on Information Theory 42*(6), 2101–2117.

Cavnar, W. B. and Trenkle, J. M. (1994). N-gram based text categorization. In *Proceedings of the Symposium on Document Analysis and Information Retrieval*, Las Vegas, pp. 161–175.

Cohen, W. W. (1995a). Learning to classify english text with ILP methods. In *Advances in Inductive Logic Programming*, pp. 124–143. IOS Press.

Cohen, W. W. (1995b). Text categorization and relational learning. In *International Conference on Machine Learning (ICML'95)*, pp. 124–132.

Cohen, W. W. and Singer, Y. (1996). Context-sensitive learning methods for text categorization. In *Proceedings of the Nineteenth Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 307–315.

Cohn, D. A., Ghahramani, Z., and Jordan, M. I. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research* **4**, 129–145.

Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning* **20**, 273–297.

Cover, T. M. and Hart, P. (1967). The nearest neighbor decision rule. *IEEE Transactions on Information Theory* **13**, 21–27.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. Wiley & Sons.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slattery, S. (1998). Learning to extract symbolic knowledge from the world wide web. In *Proceedings of AAAI-98*, pp. 509–516.

Croft, W. B. and Harper, D. J. (1979). Using probabilistic models of document retrieval without relevance information. *Journal of Documentation 35*(4), 285–295.

Dagan, I., Karov, Y., and Roth, D. (1997). Mistake-driven learning in text classification. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pp. 55–63.

Dasarathy, B. V. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. McGraw-Hill Computer Science Series. Las Alamitos, California: IEEE Computer Society Press.

Dash, M. and Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis 1*(3). An online version is avaible at `http://www-east.elsevier.com/ida/browse/0103/ida00013/article.htm`.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*(6), 391–407.

Delgado, J., Ishii, N., and Ura, T. (1998). Content-based collaborative information filtering. In M. Klusch and G. Weiß (Eds.), *Proceedings of Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet, Second International Workshop, CIA'98*, Volume 1435 of *Lecture Notes in Computer Science*, pp. 206–215. Springer-Verlag.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* **39**, 1–38.

Denning, P. J. (1982). Electronic Junk. *Communications of the ACM 25*(3), 163–165.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation 10*(7), 1895–1923.

Dietterich, T. G. (2000a). Ensemble methods in machine learning. In *First International Workshop on Multiple Classifier Systems*, Lecture Notes in Computer Science, New York, pp. 1–15. Springer-Verlag.

Dietterich, T. G. (2000b). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning 40*(2), 139–158.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* **29**, 103–130.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. New York: Wiley-Interscience.

Dudani, S. A. (1976). The distance-weighted k-nearest neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics* **6**, 325–327.

Dumais, S., Platt, J., Heckerman, D., and Sahami, M. (1998). Inductive learning algorithms and representation for text categorization. In *Proceedings of the Seventh International Conference on Information and Knowledge Management*, pp. 148–155.

Dumais, S. T. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, and Computers 23*(2), 229–236.

Dumais, S. T. (1994). Latent Semantic Indexing (LSI): TREC-3 report. In D. Harman (Ed.), *Overview of the Third Text REtrieval Conference*, pp. 219–230. NIST.

Dunning, T. E. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics 19*(1), 61–74.

Emde, W. (1994). Inductive learning of characteristic concept descriptions from small sets of classified examples. In *Proceedings of the 7th European Conference on Machine Learning (EMCL)*.

Fagan, J. (1987). *Experiments in Automatic Phrase Indexing for Document Retrieval: A Comparison of Syntactic and non-Syntactic Methods*. Ph. D. thesis, Department of Computer Science, Cornell University.

Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery: An overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press.

Ferguson, I. A. and Karakoulas, G. J. (1996). Multiagent learning and adaptation in an information filtering market. In *Proceedings AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pp. 28–32.

Foltz, P. W. (1990). Using latent semantic indexing for information filtering. In *Proceedings of the Conference on Office Information Systems*, Cambridge, MA, pp. 40–47.

Foltz, P. W. and Dumais, S. T. (1992). Personalized information delivery: An analysis of information-filtering methods. *Communications of the ACM 35*(12), 51–60.

Fox, C. (1992). Lexical analysis and stoplists. See Frakes and Baeza-Yates (1992), pp. 102–130.

Frakes, W. B. (1992). Stemming algorithms. See Frakes and Baeza-Yates (1992), pp. 131–160.

Frakes, W. B. and Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth Interational Conference on Machine Learning*, pp. 148–156.

Freund, Y., Seung, H. S., Shamir, E., and Tishby, N. (1997). Selective sampling using the query by committee algorithm. *Machine Learning* **28**, 133–168.

Friedman, J. H. (1997). On bias, variance, 0/1–loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery* **1**, 55–77.

Fürnkranz, J. (1998). A study using n-gram features for text categorization. Technical Report OEFAI-TR-98-30, Austrian Research Institute for Artificial Intelligence.

Fürnkranz, J., Mitchell, T., and Riloff, E. (1998). A case study in using linguistic phrases for text categorization on the www. In M. Sahami (Ed.), *Proceedings of the AAAI/ICML'98 Workshop on Learning for Text Categorization*, Madison, WI, pp. 5–12.

Ghahramani, Z. and Jordan, M. I. (1994). Supervised learning from incomplete data via an em approach. In J. D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems 6*, pp. 120–127. Morgan Kaufmann Publishers.

Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM 35*(12), 61–70.

Goldman, S. and Zhou, Y. (2000). Enhancing supervised learning with unlabeled data. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp. 327–334.

Greene, W. H. (1997). *Econometric Analysis*. Prentice Hall.

Greenspan, H., Goodman, R., and Chellappa, R. (1991). Texture analysis via unsupervised and supervised learning. In *International Joint Conference on Neural Networks*, pp. 639–644.

Hamilton, J. D. (1994). *Time Series Analysis*. Princeton University Press.

Harman, D. (1992). Ranking algorithms. See Frakes and Baeza-Yates (1992), pp. 363–392.

de Heer, T. (1982). The application of the concept of homeosemy to natural language information retrieval. *Information Processing Manangement 18*(5), 229–236.

Helmbold, D. P. and Long, P. M. (1994). Tracking drifting concepts by minimizing disagreements. *Machine Learning* **14**, 27–45.

Hill, W. C., Hollan, J. D., Wroblewski, D., and McCandless, T. (1992). Read wear and edit wear. In *Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI'92*, pp. 3–9. ACM Press.

Hofmann, T. (2000). Learning the similarity of documents: An information-geometric approach to document retrieval and categorization. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems 12*, pp. 914–920. MIT Press.

Hogg, R. V. and Ledolter, J. (1992). *Applied Statistics for Engineers and Physical Scientists*. New York: MacMillan.

Hsu, W.-L. and Lang, S.-D. (1999). Classification algorithms for netnews articles. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, pp. 114–121.

Hull, D. A. (1998). The TREC-6 Filtering Track: Description and Analysis. In *Proceedings of the Sixth Text REtrieval Conference (TREC-6)*, Gaithersburg, MD, pp. 45–67. National Institute of Standards and Technology.

Hull, D. A., Pedersen, J. O., and Schütze, H. (1996). Method combination for document filtering. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 279–287.

Ingwersen, P. (1992). *Information Retrieval Interaction*. Taylor Graham.

Ingwersen, P. (2000). Personal communication.

Ittner, D. J., Lewis, D. D., and Ahn, D. D. (1995). Text categorization of low quality images. In *Fourth Annual Symposium on Document Analysis and Information Retrieval*, pp. 301–315.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Englewood Cliffs, New Jersey: Prentice Hall.

Jeon, B. and Landgrebe, D. A. (1999). Partially supervised classification using weighted unsupervised clustering. *IEEE Transactions on Geoscience and Remote Sensing 37*(2), 1073–1079.

Jimenez, L. O. Landgrebe, D. A. (1998). Supervised classification in high-dimensional space: Geometrical, statistical, and asymptotical properties of multivariate data. *IEEE Transactions on Systems, Man, and Cybernetics—Part C 28*(1), 39–54.

Joachims, T. (1997a). A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *International Conference on Machine Learning*, pp. 143–151.

Joachims, T. (1997b). Text categorization with support vector machines: Learning with many relevant features. Technical Report LS–8 Report 23, Fachbereich Informatik, Universität Dortmund, Dortmund.

Joachims, T. (1999a). Making large-scale SVM learning practical. See Schölkopf, Burges and Smola (1999), pp. 169–184.

Joachims, T. (1999b). Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*, pp. 200–209.

John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, pp. 121–129.

Jones, R., McCallum, A., Nigam, K., and Riloff, E. (1999). Bootstrapping for text learning tasks. In *IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications*, pp. 52–63.

Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press.

Kilander, F., Fåhræus, E., and Palme, J. (1997). Intelligent information filtering. Technical Report 97–002, Department of Computer and Systems Sciences, Stockholm University, Sweden.

Kivinen, J. and Warmuth, M. (1995). The perceptron algorithm vs. winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant. In *Conference on Computational Learning Theory*, pp. 289–296.

Klapp, O. (1986). *Overload and Boredom: Essays on the Quality of Life in the Information Society*. Greenwood Press.

Klinkenberg, R. and Renz, I. (1998). Adaptive information filtering: Learning in the presence of concept drifts. In *Learning for Text Categorization*, Menlo Park, California, pp. 33–40. AAAI Press.

Klinkenberg, R. (1999). Learning drifting concepts with partial user feedback. In P. Perner and V. Fink (Eds.), *Beiträge zum Treffen der GI-Fachgruppe 1.1.3 Machinelles Lernen (FGML-99)*, pp. 44–53.

Klinkenberg, R. and Joachims, T. (2000). Detecting concept drift with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp. 487–494.

Klose, A., Nünberger, A., Kruse, R., Hartmann, G. K., and Richards, M. (2000). Interactive text retrieval based on document similarities. *Physics and Chemistry of the Earth 25*(8), 649–654.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence 97*(1–2), 273–324.

Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, pp. 170–178.

Konstan, Miller, Maltz, Herlocker, Gordon, and Riedl (1997). GroupLens: Collaborative Filtering for Usenet News. *Communications of the ACM 40*(3), 77–87.

de Kroon, H., Mitchell, T., and Kerckhoffs, E. (1996). Improving learning accuracy in information filtering. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*.

Kuh, A., Petsche, T., and Rivest, R. (1991). Learning time-varying concepts. In *Advances in Neural Information Processing Systems*, pp. 183–189.

Lam, W., Mukhopadhyay, S., J., M., and Palakal, M. (1996). Detection of shifts in user interests for personalized information filtering. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 317–325.

Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 331–339.

Langley, P. (1994). Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall Symposium on Relevance*, pp. 140–144.

Langley, P. (1996). *Elements of Machine Learning*. Morgan Kaufmann Publishers.

Langley, P., Iba, W., and Thompson, K. (1992). An analysis of Baysian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 223–228.

Lanquillon, C. (1999a). Dynamic Aspects in Neural Classification. *International Journal of Intelligent Systems in Accounting, Finance & Management 8*(4), 281–296.

Lanquillon, C. (1999b). Evaluating performance indicators for adaptive information filtering. In *Proceedings of the International Computer Science Conference: Internet Applications*, Lecture Notes in Computer Science, Hong Kong, pp. 11–20. Springer-Verlag.

Lanquillon, C. (1999c). Information filtering in changing domains. In *Proceedings of the IJCAI'99 Workshop on Machine Learning for Information Filtering*, Stockholm, Sweden, pp. 41–48.

Lanquillon, C. (2000a). Learning from labeled and unlabeled documents: A comparative study on semi-supervised text classification. In *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-2000)*, Lecture Notes in Artificial Intelligence, Lyon, France, pp. 490–497. Springer-Verlag.

Lanquillon, C. (2000b). Partially supervised text classification: Combining labeled and unlabeled documents using an EM-like scheme. In *Proceedings of the Eleventh European Conference on Machine Learning (ECML-2000)*, Lecture Notes in Artificial Intelligence, Barcelona, Spain, pp. 229–237. Springer-Verlag.

Lanquillon, C. and Renz, I. (1999). Adaptive information filtering: Detecting changes in text streams. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, Kansas City, Missouri, pp. 538–544.

Larkey, L. S. and Croft, W. B. (1996). Combining classifiers in text categorization. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 289–297.

Lewis, D. D. (1991). Evaluating text categorization. In *Proceedings of the Speech and Natural Language Workshop*, Asilomar, pp. 312–318. Morgan Kaufmann Publishers.

Lewis, D. D. (1992a). 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*, Copenhagen, pp. 37–50.

Lewis, D. D. (1992b). *Representation and Learning in Information Retrieval.* Ph. D. thesis, Department of Computer and Information Science, University of Massachusetts.

Lewis, D. D. (1995). Evaluating and optimizing autonomous text classification systems. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 246–254.

Lewis, D. D. (1997). The TREC-5 filtering track. In *Proceedings of the Fifth Text REtrieval Conference (TREC-5)*, Gaithersburg, MD, pp. 75–96. National Institute of Standards and Technology.

Lewis, D. D. and Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning (ICML'94)*, pp. 148–156.

Lewis, D. D. and Croft, W. B. (1990). Term clustering of syntactic phrases. In *Proceedings of the Thirteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 385–404.

Lewis, D. D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, London, pp. 3–12. Springer-Verlag.

Lewis, D. D. and Ringuette, M. (1994). A comparison of two learning algorithms for text classification. In *Third Annual Symposium on Document Analysis and Information Retrieval*, pp. 81–93.

Lewis, D. D., Schapire, R. E., Callan, J. P., and Papka, R. (1996). Training algorithms for linear text classifiers. In *Proceedings of 19th Annual International ACM SIGIR on Research and Development in Information Retrieval*, pp. 298–306.

Li, Y. H. and Jain, A. K. (1998). Classification of text documents. *The Computer Journal 41*(8), 537–546.

Lieberman, H. (1995). Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 924–929.

Liere, R. and Tadepalli, P. (1996). The use of active learning in text categorization. In *Working Notes of the AAAI 1996 Spring Symposium on Machine Learning in Information Access*, Stanford, CA.

Lippmann, R. P. (1989). Pattern classification using neural networks. *IEEE Communications Magazine* **27**, 47–64.

Little, R. J. A. (1977). Response to Dempster's original EM paper. See Dempster, Laird and Rubin (1977), pp. 1–38.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning* **2**, 285–318.

Loeb, S. and Terry, D. (1992). Information filtering. *Communications of the ACM 35*(12), 27–28.

Losee, R. M. (1998). *Text Retrieval and Filtering: Analytic Models of Performance*. Kluwer Academic Publishers.

Luhn, H. P. (1958a). A Business Intelligence System. See Schultz (1968), pp. 132–139.

Luhn, H. P. (1958b). The Automatic Creation of Literature Abstracts. See Schultz (1968), pp. 118–125.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman (Eds.), *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Volume Volume I, Statistics, pp. 281–297. University of California Press.

Makoto, I. and Takenobu, T. (1995). Cluster-based text categorization: A comparision of category search strategies. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 273–280.

Malone, T. M., Grand, K. R., Turbak, F. A., Brobst, S. A., and Cohen, M. D. (1987). Intelligent Information-Sharing Systems. *Communications of the ACM 30*(5), 390–402.

Maloof, M. and Michalski, R. (1995). Learning evolving concepts using partial memory approach. In *Working Notes of the 1995 AAAI Fall Symposium on Active Learning*, Boston, MA, pp. 70–73.

Marchionini, G. (1997). *Information Seeking in Electronic Environments*. Number 9 in Cambrige Series in Human-Computer Interaction. Cambridge University Press.

Maron, M. (1961). Automatic indexing: An experimental inquiry. *Journal of the ACM* **8**, 404–417.

Masand, B., Linoff, G., and Waltz, C. (1992). Classifying news stories using memory based reasoning. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 59–65.

McCallum, A. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/∼mccallum/bow`.

McCallum, A. and Nigam, K. (1998a). A comparison of event models for naive Bayes text classification. In *AAAI/ICML'98 Workshop on Learning for Text Categorization*, pp. 41–48. AAAI Press.

McCallum, A. and Nigam, K. (1998b). Employing EM and pool-based active learning for text classification. In *Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98)*, pp. 359–367.

McCallum, A. and Nigam, K. (1999). Text classificaiton by bootstrapping with keywords, em and shrinkage. In *Association for Computational Linguistics (ACL'99) Workshop on Unsupervised Learning in Natural Language Processing*, pp. 52–58.

McCallum, A., Rosenfeld, R., Mitchell, T., and Ng, A. (1998). Improving text classification by shrinkage in a hierarchy of classes. In *Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98)*, pp. 359–367.

McLachlan, G. and Basford, K. (1988). *Mixture Models*. New York: Marcel Dekker.

McLachlan, G. and Krishnan, T. (1997). *The EM Algorithm and Extensions*. Wiley & Sons.

McLachlan, G. J. (1975). Iterative reclassification procedure for constructing an asymptotically optimal rule of allocation in discriminant analysis. *Journal of the American Statistical Association 70*(358), 365–369.

McLachlan, G. J. and Ganesalingam, S. (1982). Updating the discriminant function on the basis of unclassified data. *Communications Statistics—Simulation 11*(6), 753–767.

Merkl, D. (1998). Text classification with self-organizing maps: Some lessons learned. *Neurocomputing* **21**, 61–77.

Miller, D. J. and Uyar, H. S. (1997). A mixture of experts classifier with learning based on both labelled and unlabelled data. In *Advances in Neural Information Processing Systems 9*, pp. 571–577.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence* **18**, 203–226.

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

Mitchell, T. M., Caruana, R., Freitag, D., McDermott, J., and Zabowski, D. (1994). Experience with a learning personal assistant. *Communications of the ACM* **37**, 81–91.

Mladenič, D. (1996). Personal WebWatcher: Design and implementation. Technical Report Technical Report IJS-DP-7472, Department of Intelligent Systems, J. Stefan Institute.

Mladenič, D. (1998). Feature subset selection in text-learning. In *Proceedings of the 10th European Conference on Machine Learning (ECML98)*, pp. 95–100.

Mladeniĉ, D. and Grobelnik, M. (1998). Feature selection for classification based on text hierarchy. In *Working Notes of Learning from Text and the Web, Conference on Automated Learning and Discovery (CONALD98).*

Mock, K. J. and Vemuri, V. R. (1997). Information filtering via hill climbing, wordnet, and index patterns. *Information Processing & Management 33*(5), 633–644.

Montgomery, D. C. (1997). *Introduction to Statistical Quality Control* (3rd ed.). New York: Wiley.

Mostafa, J., Mukhopadhyay, S., Lam, W., and Palakal, M. (1997). A multilevel approach to intelligent information filtering: Model, system, and evaluation. *ACM Transactions on Information Systems 15*(4), 368–399.

Moukas, A. (1996). Amalthaea: Information discovery and filtering using a multiagent evolving ecosystem. In *Proceedings of the Conference on Practical Applications of Agents and Multiagent Technology*, London.

Moulinier, I., Raŝkinis, G., and Ganascia, J.-G. (1996). Text categorization: A symbolic approach. In *Annual Symposium on Document Analysis and Information Retrieval (SDAIR)*, pp. 87–99.

Nakhaeizadeh, G., Taylor, C., and Lanquillon, C. (1998). Evaluating usefulness for dynamic classification. In *Proceedings of The Fourth International Conference on Knowledge Discovery & Data Mining*, New York, pp. 87–93.

Neumann, G. and Schmeier, S. (1999). Combining shallow text processing and machine learning in real world applications. In *Proceedings of the IJCAI'99 Workshop on Machine Learning for Information Filtering*, Stockholm, Sweden, pp. 55–60.

Ng, H. T., Goh, W. B., and Low, K. L. (1997). Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'97)*, pp. 67–73.

Nichols, D. M. (1997). Implicit rating and filtering. In *Proceedings of the 5th DELOS Workshop on Filtering and Collaborative Filtering*, pp. 31–36. ERCIM.

Nigam, K. and Ghani, R. (2000). Understanding the behavior of co-training. In *KDD-2000 Workshop on Text Mining*.

Nigam, K., McCallum, A., Thrun, S., and Mitchell, T. (1998). Learning to classify text from labeled and unlabeled documents. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pp. 792–799.

Nigam, K., McCallum, A., Thrun, S., and Mitchell, T. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning 39*(2/3), 103–134.

Oard, D. and Kim, J. (2000). Information filtering resources. `http://www.enee.umd.edu/medlab/filter/filter.html`. (as of April 10, 2000).

Oard, D. W. (1997). The state of the art in text filtering. *User Modeling and User-Adapted Interaction 7*(3), 141–178.

O'Neill, T. J. (1978). Normal discrimination with unclassified observations. *Journal of the American Statistical Association 73*(364), 821–826.

Pao, Y.-H. and Sobajic, D. J. (1992). Combined use of supervised and unsupervised learning for dynamic security assessment. *IEEE Transactions on Power Systems 7*(2), 878–884.

Pazzani, M. and Billsus, D. (1997). Learning and revising user profiles: The identification of interesting web sites. *Machine Learning* **27**, 313–331.

Pedrycz, W. (1984). Algorithms of fuzzy clustering with partial supervision. *Pattern Recognition Letters* **3**, 13–20.

Platt, J. (1999). Fast training of SVMs using sequential minimal optimization. See Schölkopf, Burges and Smola (1999), pp. 185–208.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program 14*(3), 130–137.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers.

Quinlan, J. R. (1999). Some elements of machine learning (extended abstract). In *Proceedings of the Sixteenth International Conference on Machine Learning*, Bled, Slovenia, pp. 523–525.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 Computer Supported Cooperative Work Conference*, New York, pp. 175–186. ACM.

Reuters-21578 (1997). Distribution 1.0. This text categorization test collection from D. D. Lewis is publicly available at `http://www.research.att.com/~lewis/reuters21578.html`.

van Rijsbergen, C. J. (1977). A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation 33*(2), 106–119.

van Rijsbergen, C. J. (1979). *Information Retrieval* (2 ed.). London: Butterworths. An online version is avaible at `http://www.dcs.gla.ac.uk/Keith`.

Rissland, E. E., Brodley, C. E., and Friedman, M. (1995). Measuring structural change in concepts. Technical Report MA 01003, Department of Computer Science, University of Massachusetts, Amherst, MA.

Ristad, E. S. (1995). A natural law of succession. Research Report CS-TR-495-95, Princeton University.

Rocchio, J. J. J. (1971). Relevance feedback in information retrieval. See Salton (1971), pp. 313–323.

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Sahami, M. (1998). *Using Machine Learning to Improve Information Access*. Ph. D. thesis, Department of Computer Science, Stanford University.

Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. (1998). A Bayesian approach to filtering junk e-mail. In M. Sahami (Ed.), *Proceedings of the AAAI/ICML'98 Workshop on Learning for Text Categorization*, Madison, WI.

Salton, G. (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Englewood Cliffs, New Jersey: Prentice Hall.

Salton, G. (1989). *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley.

Salton, G. and Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management 24*(5), 513–523.

Salton, G. and McGill, M. (1983). *Modern Information Retrieval*. New York: McGraw-Hill.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic text retrieval. *Communications of the ACM 18*(11), 613–620.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning 5*(2), 197–227.

Schapire, R. E. and Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine Learning 39*(2/3), 135–168.

Schapire, R. E., Singer, Y., and Singhal, A. (1998). Boosting and Rocchio applied to text filtering. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pp. 215–223.

Schohn, G. and Cohn, D. (2000). Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp. 839–846.

Schölkopf, B., Burges, C., and Smola, A. (1999). *Advances in Kernel Methods – Support Vector Learning*. MIT Press.

Schultz, C. K. (1968). *H.P. Luhn: Pioneer of Information Science—Selected Works*. London: Macmillan.

Schürmann, J. (1996). *Pattern Classification: A unified view of statistical and neural approaches*. New York: Wiley-Interscience.

Schürmann, J. (2000). Personal communication.

Schütze, H., Hull, D. A., and Pedersen, J. O. (1995). A comparison of classifiers and document representations for the routing problem. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'95),* Seattle, WA, pp. 229–237.

Selim, S. Z. and Ismail, M. A. (1984). K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-6*(1), 81–87.

Seung, H. S., Opper, M., and Sompolinsky, H. (1992). Query by committee. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pp. 287–294.

Shahshahani, B. M. and Landgrebe, D. A. (1992). On the asymptotic improvement of supervised learning by utilizing additional unlabeled samples; normal mixture case. In *SPIE International Conference on Neural and Stochastic Methods in Image and Signal Processing*.

Shahshahani, B. M. and Landgrebe, D. A. (1994). The effect of unlabeled samples in reducing the small sample size problem and mitigating the hughes phenomenon. *IEEE Transactions on Geoscience and Remote Sensing 32*(5), 1087–1095.

Shapira, B., Hanani, U., Raveh, A., and Shoval, P. (1997). Information filtering: A new two-phase model using stereotypic user profiling. *Journal of Intelligent Information Systems* **8**, 155–165.

Shapira, B., Shoval, P., and Hanani, U. (1999). Experimentation with an information filtering system that combines cognitive and sociological filtering integrated with user stereotypes. *Decision Support Systems 27*(1–2), 5–24.

Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating ”word of mouth”. In *Proceedings of CHI'95 – Human Factors in Computing Systems*, Denver, CO, pp. 210–217.

Shenk, D. (1997). *Data Smog: Surviving the Information Glut*. San Francisco: Harper.

Sheridan, T. B. and Ferrell, W. R. (1974). *Man-Machine Systems: Information, Control, and Decision Models of Human Performance*. MIT Press.

Sheth, B. (1994). A Learning Approach to Personalized Information Filtering. Master's thesis, Department of Electrical Engineering and Computer Science, MIT.

Smeaton, A. F. (1997). Information retrieval: Still butting heads with natural language processing. In *Information Extraction. International Summer School SCIE 1997*, pp. 115–139. Springer-Verlag.

Sparck Jones, K. (1971). *Automatic Keyword Classification*. London: Butterworths.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation 28*(1), 11–20.

Spertus, E. (1997). Smokey: Automatic recognition of hostile messages. In *Proceedings of Innovative Applications of Artificial Intelligence (IAAI)*, pp. 1058–1065.

Stevens, C. (1992). Automating the creation of information filters. *Communications of the ACM 35*(12), 48.

Tauritz, D. R., Kok, J. N., and Sprinkhuizen-Kuyper, I. G. (2000). Adaptive information filtering using evolutionary computation. *Information Sciences* **122**, 121–140.

Tauritz, D. R. and Sprinkhuizen-Kuyper, I. G. (1999). Adaptive information filtering algorithms. In *Intelligent Data Analysis*, Lecture Notes in Computer Science, pp. 513–524. Springer-Verlag.

Taylor, C., Nakhaeizadeh, G., and Lanquillon, C. (1997). Structural change and classification. In G. Nakhaeizadeh, I. Bruha, and T. C. (Eds.), *Workshop Notes On Dynamically Changing Domains: Theory Revision and Context Dependence Issues, 9th European Conference on Machine Learning (ECML98)*, Prague, Czech Republic, pp. 67–78.

Teufel, B. and Schmidt, S. (1988). Full text retrieval based on syntactic similarities. *Information Systems 13*(1), 65–70.

Tong, S. and Koller, D. (2000). Support vector machine active learning with applications to text classification. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp. 999–1006.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning* **4**, 161–186.

Utgoff, P. E. (1994). An improved algorithm for incremental induction of decision trees. In *Proceedings of the Eleventh International Machine Learning Conference*, pp. 318–325.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM* **27**, 1134–1142.

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

Vapnik, V. (1995). *The Nature of Statistical Learning*. Springer-Verlag.

Voorhees, E. M. and Harman, D. (1998). Overview of the Sixth Text REtrieval Conference (TREC-6). In *Proceedings of the Sixth Text REtrieval Conference (TREC-6)*, Gaithersburg, MD, pp. 1–24. National Institute of Standards and Technology.

Wang, Z. W., Wong, S. K. M., and Yao, Y. Y. (1992). An analysis of vector space models based on computational geometry. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92),* pp. 152–160.

Widmer, G. and Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning* **23**, 69–101.

Wiener, E., Pedersen, J. O., and Weigend, A. S. (1995). A neural network approach to topic spotting. In *Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95),* pp. 317–332.

Willett, P. (1983). Similarity coefficients and weighting functions for automatic document classification: An empirical comparison. *International Classification* **10**, 138–142.

Willett, P. (1988). Recent trends in hierarchic document clustering: A critical review. *Information Management & Processing 24*(5), 577–597.

Xu, J. and Croft, W. B. (1996). Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* Zürich, Switzerland, pp. 4–11.

Yahoo! (2000). A searchable internet guide. `http://www.yahoo.com`.

Yan, T. W. and Garcia-Molina, H. (1994). Index structures for information filtering under the vector space model. In *Proceedings of the International Conference on Data Engineering,* pp. 337–347.

Yan, T. W. and Garcia-Molina, H. (1995). SIFT – a tool for wide-area information dissemination. In *Proceedings of the 1995 USENIX Technical Conference*.

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Information Retrieval Journal 1*(1/2), 67–88.

Yang, Y., Ault, T., Pierce, T., and W., L. C. (2000). Improving text categorization methods for event tracking. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 65–72.

Yang, Y., Carbonell, J., Brown, R., Pierce, T., Archibald, B. T., and Liu, X. (1999). Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems: Special Issue on Applications of Intelligent Information Retrieval 14*(4), 32–43.

Yang, Y. and Chute, C. G. (1993). An application of least squares fit mapping to text information. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 281–290.

Yang, Y. and Liu, X. (1999). A re-examination of text categorization methods. In *Proceeding of ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 42–49.

Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, pp. 412–420.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pp. 189–196.

Zelikovitz, S. and Hirsh, H. (2000). Improving short-text classification using unlabeled background knowledge to assess document similarity. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp. 1183–1190.

Zhang, T. and Oles, F. J. (2000). A probability analysis on the value of unlabeled data for classification problems. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp. 1191–1198.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Reading, MA: Addison-Wesley.

Lebenslauf

Personalien

Name	Carsten Lanquillon
Adresse	Ludwig-Beck-Straße 25, 38116 Braunschweig
Geburtstag	15. August 1972
Geburtsort	Braunschweig
Nationalität	deutsch

Bildungsweg

11/1997 – 10/2000	DaimlerChrysler AG, Research & Technology, Ulm Doktorand
08/1999 – 12/1999	Carnegie Mellon University, Pittsburgh, Pennsylvania, USA Gastwissenschaftler am Center for Automated Learning and Discovery
10/1993 – 03/1998	Fernuniversität Hagen Studiengangszweithörer Schwerpunkt: Wirtschaftswissenschaften
10/1994 – 10/1997	Technische Universität Braunschweig Hauptstudium Informatik Abschluss: Diplom-Informatiker (Gesamtnote: *sehr gut*) Schwerpunkte: Datenbanken, Soft Computing, VLSI Chip-Entwurf Nebenfach: Informationsmanagement
02/1997 – 07/1997	University of Leeds, U.K. Gastwissenschaftler, Diplomarbeit für die DaimlerChrysler AG Thema: Credit Scoring mit Neuronalen Netzen
10/1992 – 09/1994	Technische Universität Braunschweig Grundstudium Informatik Abschluss: Informatik Vordiplom (Gesamtnote: *sehr gut*)
08/1985 – 06/1992	Hoffmann-von-Fallersleben Gymnasium, Braunschweig Abschluss: Abitur (Gesamtnote: *1.2*) Prüfungsfächer: Mathematik, Physik, Englisch, Geschichte
08/1989 – 05/1990	San Juan High School, Blanding, Utah, USA Gastschüler Abschluss: American High School Diploma (Gesamtnote: *A+*)

Berufstätigkeit

seit 11/2000	DaimlerChrysler AG, Research & Technology, Ulm Wissenschaftlicher Mitarbeiter in der Abteilung Data Mining Solutions
02/1993 – 12/1996	APG Gerüstbau GmbH, Braunschweig Datenbankentwurf und Softwareentwicklung