NTNU

Norwegian University of
Science and Technology

Bayesian Text Categorization

Arild Brandrud Næss

Master of Science in Physics and Mathematics
Submission date: December 2007
Supervisor: Jo Eidsvik, MATH
Co-supervisor: Heri Ramampiaro, IDI

Norwegian University of Science and Technology
Department of Mathematical Sciences

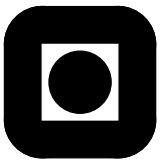
Problem Description

Survey the current methods in text categorization, with particular emphasis on the statistically grounded ones, and assess their potential for use on the BioTracer data. The student is to select one method for further examination. This method is to be tested thoroughly, and the student should attempt to improve upon it. One of the other assessed methods should also be subjected to testing and used as a benchmark.

Assignment given: 30. July 2007
Supervisor: Jo Eidsvik, MATH

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology,
Mathematics and Electrical Engineering



MASTER'S THESIS

FOR

STUD.TECHN. Arild Brandrud Næss

FACULTY OF INFORMATION TECHNOLOGY,
MATHEMATICS AND ELECTRICAL ENGINEERING

NTNU

Discipline: Statistics

Title: "Bayesian Text Categorization"

Purpose of the work: Survey the current methods in text categorization, with particular emphasis on the statistically grounded ones, and assess their potential for use on the BioTracer data. The student is to select one method for further examination. This method is to be tested thoroughly, and the student should attempt to improve upon it. One of the other assessed methods should also be subjected to testing and used as a benchmark.

This Master's Thesis is to be carried out at the Department of Mathematical Sciences under the supervison of associate professor Jo Eidsvik, in collaboration with associate professor Heri Ramampiaro at the Department of Computer Science.

Trondheim, 30th July, 2007.

Abstract

Natural language processing is an interdisciplinary field of research which studies the problems and possibilities of automated generation and understanding of natural human languages. Text categorization is a central subfield of natural language processing. Automatically assigning categories to digital texts has a wide range of applications in today's information society—from filtering spam to creating web hierarchies and digital newspaper archives. It is a discipline that lends itself more naturally to machine learning than to knowledge engineering; statistical approaches to text categorization are therefore a promising field of inquiry.

We provide a survey of the state of the art in text categorization, presenting the most widespread methods in use, and placing particular emphasis on support vector machines—an optimization algorithm that has emerged as the benchmark method in text categorization in the past ten years.

We then turn our attention to Bayesian logistic regression, a fairly new, and largely unstudied method in text categorization. We see how this method has certain similarities to the support vector machine method, but also differs from it in crucial respects. Notably, Bayesian logistic regression provides us with a statistical framework. It can be claimed to be more modular, in the sense that it is more open to modifications and supplementations by other statistical methods; whereas the support vector machine method remains more of a black box.

We present results of thorough testing of the BBR toolkit for Bayesian logistic regression on three separate data sets. We demonstrate which of BBR's parameters are of importance; and we show that its results compare favorably to those of the SVM^{light} toolkit for support vector machines. We also present two extensions to the BBR toolkit. One attempts to incorporate domain knowledge by way of the prior probability distributions of single words; the other tries to make use of uncategorized documents to boost learning accuracy.

Preface

This thesis forms part of a Master of Technology degree in Applied Mathematics at The Norwegian University of Science and Technology (NTNU), and was be completed in a total of 20 weeks in the final term.

Although mathematical models generally, and statistical ones especially, are growing increasingly widespread within many fields of natural language processing, it has been a challenge to find the mathematically relevant material for this thesis. One often has to plough through a large amount of survey articles, before finding good descriptions of the theoretical foundations for the models. We have done our best, however, to provide some theoretical framework for the methods presented.

At the NTNU, the Master of Technology degree includes two theses, a preliminary thesis commonly referred to as the "Prosjektet", which is written in the second to last term, and the master's thesis of the final term, "Diplomen". These can be concerned with the same or with distinct subject matters.

The project description for this thesis would probably have been more suitable as a project for a full academic year—i.e. as a subject for both the preliminary thesis and the master's thesis. My preliminary thesis (Næss, 2007), however, was written on an entirely different subject matter. The work behind this thesis has thus been done in its entirety in the allotted 20 weeks.

Acknowledgements

I would like to thank my supervisors Jo Eidsvik and Heri Ramampiaro for their support. The layout of this thesis is indebted to Ph.D. student Marius Thaule's proficiency in LaTeX. My thanks also to Arvid Næss for proofreading the final versions of the manuscript.

Contents

1

Introduction

The point of departure for the work behind this thesis, was trying to find a way to improve the BioTracer search engine. BioTracer is developed for use in biomedical research, and uses the Medline database of medical abstracts as its search space.[1] Medline contains about 15 million entries from all fields of medical research. Thus, one of the greatest potentials for improvement is to try to reduce the search space—i.e. to separate the data into several groups, so that for a given search we would only have to search through a few of them. One can envision several different ways of doing this, either by automatically choosing the groups to search based on the user's search query, or letting the user choose manually which groups he considers relevant. We have not been concerned with these choices; the scope of the work in this thesis has been restricted to examining the possibilities for categorising medical texts automatically.

The field of text categorization (TC) has undergone a major development since the first efforts in the 1960s, particularly in the last 15 years, and today's TC methods show fairly good results. Compared to the performance of information retrieval systems, however, the results of current TC algorithms remain inferior. TC may thus be said to not yet have fully matured as a research field, and Sebastiani (2002) notes that there are neither textbooks nor journals entirely devoted to TC yet.

This thesis will give an extensive account of the research field of TC. Although it does not aspire to be fully comprehensive, it should provide a thorough introduction.

Looking over the current research on TC, one notices that support vector machines (SVMs) are growing increasingly dominant. The SVM method, which is a form of optimization algorithm, is showing comparatively very good results. However, we also found that Bayesian logistic regression (BLR) is maturing as an alternative, and it might now be said to be the main competitor to SVMs in TC.

We have chosen to place particular emphasis on BLR—partly because it lends itself more naturally to a statistical discussion than is the case for SVMs.

Overview

We begin by presenting the different facets of the text categorization task in chapter 2. After presenting a selection of the most well-known TC methods in section 2.7.4, we consider support vector machines in some detail. Chapter 3 goes on to present the theory behind Bayesian logistic regression, as well as its implementation. At the end of chapter 3 we present our proposed extensions to BLR. Chapter 4 concerns our data sets, and chapter 5 presents the results we obtained on them. These are discussed in chapter 6, before chapter 7 provides a summary and suggestions for future work.

[1] The work on BioTracer is as yet unpublished, but further information can be found in an article from *forskning.no*, available at http://www.forskning.no/Artikler/2007/mai/1179818465.89

2

Text categorization

Text categorization is a form of text mining, which in its turn is a form of data mining. "Where is the knowledge we have lost in information?" lamented the poet T.S. Eliot in his pageant play *The Rock*. In today's world, where internet-enabled computers have become ubiquitous, unlimited amounts of information seem to be universally accessible—at least in the Western hemisphere. However, most of this information is in a raw unstructured form, that might be better described as *data*. So one could be tempted to add to Eliot's lament: "Where is the information we have lost in data?"

Feldman and Sanger (2007) define *information* as the set of patterns or expectations that underlie the data. *Data mining*, then, is the extraction of implicit, previously unknown, and potentially useful information from data. In the case of *text mining*, the data is unstructured text in digital form, and the task is to automatically extract some form of non-trivial information, or knowledge, from these text files. Text mining encompasses such diverse fields as information retrieval, entity relation, event extraction, clustering and text categorization.

We can define text categorization as the task of dividing natural language documents into a predefined set of semantic categories. By a "semantic" category, we mean that the category has to be meaningful in some sense—all documents pertaining to prostate cancer is a semantic category, all documents not containing the letter *e* is not. And the set of such categories is predefined, we know in advance which and how many they are. This separates text categorization from *clustering* of texts. A clustering algorithm also attempts to group documents that are similar in meaning, but the grouping is *ad hoc*, we do not know exactly what characterizes each group or cluster. Thus, whereas clustering is an example of *unsupervised learning*, text categorization is an instance of *supervised learning*.

Text categorization is also referred to as *text classification*. At the time of writing, Google Scholar gives 14400 search results for the phrase "text categorization", versus 12000 results for "text classification"; i.e., one is not substantially more common than the other. These terms are usually used synonymously and they are both commonly abbreviated as TC. Genkin et al. (2007), however, distinguishes between the terms by defining text categorization as the special case of text classification where the classes are "of general interest" (such as e.g. genres of text or Library of Congress headings). In this thesis, we do not make such a distinction; we will use the two terms interchangeably, and will often refer to a text categorization system as a *classifier*.

The history of text classification dates back to the early 1960s. To begin with, the motivation was mainly to try to annotate scientific articles from a controlled vocabulary. As has happened within many fields of artificial intelligence (machine translation is another example that springs to mind) these early pioneering efforts was followed by a long hiatus, until the research field saw a revival with the surge in information technology and processing power of the early 1990s.

Today the applications of text categorization abound. The most omnipresent of these, are the spam filters we all use. A spam filter is a form of text classifier, where the e-mail is predicted to belong to one of two categories: spam or not-spam. Thus, this is a *binary* classifier, and as we will see, binary classifiers might be said to form the backbone of many TC systems. An example of a more advanced TC task, would be a corporation who wishes to route incoming e-mails to the correct employee, e.g. all support requests go to Mr. Smith and all job applications go to Mr. Jones. Another common example are newspapers who want to automatically archive all their stories, sorted in thematic categories. Web hierarchies such as those developed by Yahoo! is another application of text classification.

2.1 Knowledge engineering v machine learning

Artificial intelligence (AI) can be said to be divided in symbolic and sub-symbolic approaches. Symbolic AI tends to rely on rules formulated in formal logic, it is also called *knowledge engineering*. It involves an expert's knowledge which is encoded in a system in the form of handwritten rules. The obvious drawback is what Feldman and Sanger (2007) call the *knowledge acquisition bottleneck*: The construction of such a system requires not only a huge amount of highly skilled labor, but also very detailed expert knowledge of the task at hand.

The most famous example of a knowledge engineering approach to TC is the CONSTRUE system developed by the Carnegie group. A typical rule in this system would have the form:
if DNF formula[1]
then category
else not category.

The CONSTRUE system was developed for the Reuters corpus, which consists of texts from Newswire so a rule might look like the following example from Feldman and Sanger (2007):
if ((wheat **and** form) **or** (wheat **and** commodity)
or (bushels **and** export) **or** (wheat **and** tonnes)
or (wheat **and** winter **and** not soft))
then Wheat
else not Wheat

The CONSTRUE system managed to achieve very good results, but it was used only on a small subset of the Reuters corpus and took several man-years to develop and fine-tune for these texts.

Therefore the machine learning approach has an obvious appeal, and is today without question the dominant paradigm within TC. The principle behind machine learning algorithms for TC might be said to be to induce rules such as those outlined above automatically. In a lecture, the late Karen Sparck Jones summed this up as follows: "The take-home message is: statistical methods work through redundancy; all use of language has redundancy; so: statistical strategies are sound basic tools for information management."

One of the main challenges of TC is the large input space. When each distinct word is considered as one dimension, a document collection of some size will span about 30 000 dimensions.

[1] Disjunctive Normal Form. This is a formula in predicate logic which consists of a disjunction of conjunctive clauses.

2.2 The representation of text

Machine learning algorithms are not able to process text directly, it has to be represented in the form of numerical vectors. There are several possible design choices in how to thus represent the text. We will here review the main options.

We want to represent a document d as a vector $\mathbf{d}$, which will consist of numerical *term weights*: $\mathbf{d} = (w_1, w_2, ..., w_{|\mathcal{T}|})$ where $\mathcal{T}$ is the set of *feature terms*—ie, our basic level of analysis. As Sebastiani (2002) points out, the differences among text representation approaches are accounted for by

- different ways to understand what a term is; and by
- different ways to compute term weights.

In other words, we need to decide on both which features of a text we want our TC algorithm to keep track of, and how to assign numerical values to each of these features for a certain document.

2.2.1 Levels of feature representation

Joachims (2002) separates the different ways to understand terms into five levels

1. Sub-word level: decomposition of words and their morphology
2. Word level: words and lexical information
3. Multi-word level: phrases and syntactic information
4. Semantic level: the meaning of text
5. Pragmatic level: the meaning of text with respect to context and situation

These levels are not separate—our understanding of a text incorporates all of them. We can say that on each level, there exists ambiguities that can only be solved by using the level above. Viewed as a single word, we cannot say whether "book" is a noun or a verb; but once we see it in the context of a phrase, e.g. "Please book that flight!", it is usually evident.

If we represent a text on the *word level*, we are disregarding the order in which the words occur, or their wider context. We are representing the document solely based on which words occur in it, and how many times. This is what is commonly called a *bag-of-words* approach.

There is a case to be made for going one level further down, to the *sub-word* level. A word-level approach would treat different inflections of a verb, e.g. "discuss" and "discussed", as entirely unrelated. It will also ignore that similar words, such as "compute", "computability" and "computer", tend to have very similar meanings.

In Norwegian we also have the case of compound words. Entities which in English would be denoted by two or more words, such as "The European Parliament", will in Norwegian, and the other Scandinavian languages, be a single word, "Europaparlamentet". The Scandinavian languages also have a richer word inflection than English; in addition to verbs, adjectives and nouns are inflected to a greater extent. There are also more extreme examples, such as Turkish or Finnish, which are very morphologically complex languages. Jurafsky and Martin (2000) give the following example of a Turkish word:

uygarlaştıramadıklarımızdanmışsınızcasına,

the approximate translation of which is: "behaving as if you are among those whom we could not civilize". This is obviously no common Turkish word, but it is a grammatically coherent one. The mere possibility of constructing such words, highlights the problem that processing text on the word level entails—also for morphologically simpler languages than Turkish.

There are two standard representations on the sub-word level: n-grams and stemming. In a n-gram representation, we use strings of n characters as our basic terms. Thus, if we let $n = 3$, i.e. we choose a a *trigram* representation, the word "book" is represented as the sequence ("_bo", "boo", "ook", "ok_"). This has the advantage of being straightforward, but it can be misleading, e.g. in detecting a likeness between "computer" and "commuter" (cf. e.g. Neumann and Schmeier (1999)).

Stemming is a more advanced approach, in that it conducts a simple morphological analysis of the words. It assumes that the different word forms of the same stem are equivalent with respect to the classification task—ie, "compute", "computability" and "computer" would all be projected to the same stem term "comput". The Porter stemming algorithm for English is well developed and achieves good results (Porter, 1980), but it is not entirely straightforward to develop a stemming algorithm for a new language, especially not for a morphologically rich one, like Turkish or Finnish. And in the case of medical articles, one tends to avoid preprocessing like stemming, since it can often lead to the system confounding different genes or pathological conditions with similar names.

There are also many situations where we would like to go a level up from the separate words, e.g. to resolve ambiguous words like "book" or "bank". On the *multi-word* level, there is also two standard forms representation, and again n-grams is the simpler of the two. On this level n denotes the number of words rather than the number of characters, so that a 5-gram model will consider five words in a row. This demands large storage capacity—the number of possible sequences of five words is far larger than the possible sequences of five characters—but is straightforward to implement.

It is also possible, however, to analyze text with respect to its syntactic structure. The most commonly used syntactic structures are noun phrases (cf. e.g. Lewis (1992)). This captures more meaningful contexts of words.

The satisfactory representation on the *semantic level* would be close to ideal, since we would then capture the meaning of each sentence or even the document as a whole. However, the semantic representations of today are either too demanding to translate into, like predicate logic or semantic nets, or too simple to capture much of the actual semantics of a document, e.g. web hierarchies. And as yet, there has been no systematic efforts in representing texts at the *pragmatic level* (Joachims, 2002). So, in practice, there have been no levels 4 and 5 are out of the question. As for efforts at representing texts at levels 1 and 3, there have been a few, but they have generally not shown improved results compared to indexing at the word level. Thus, the standard choice of feature representation level in TC research, has become level 2—indexing the words that occur in a text.

2.2.2 Computation of feature weights

After having chosen what the feature terms should be, one has to decide how to *weight* them, i.e. how to compute the numeric values of the document representation vectors.

The simplest form of feature weighting is the *binary one*, letting the weight w_{ij} of the feature term t_j in the document d_i equal one if t_j occurs in d_i and zero otherwise. Another weighting is using the *raw frequency*, letting w_{ij} equal the number of occurences of t_j in d_i.

Both of the above-mentioned feature weightings have been frequently used. However, the TF-IDF weighting scheme introduced by Salton et al. (1975) has become something of a default choice. The TF-IDF is given by

$$w_{ij} = f_{ij} \log \frac{|\mathcal{D}|}{D(t_i)} \tag{2.1}$$

where f_{ij} is the frequency of feature term t_j in document d_i, $|\mathcal{D}|$ is the total number of documents and $D(t_i)$ is the number of documents containing the feature t_i. The first factor of (2.1) is often referred to as the term frequency (TF), and the second term as the inverse document frequency (IDF). The IDF factor ensures that terms which occur in a large portion of the documents are given less weight than those which occur in just a few.

2.3 Dimension reduction

With such a large dimensionality, there are many reasons to try to reduce the number of terms. There are two ways to approach this: One can either try to *select* some of the features and ignore the rest, or one can try to *construct* a smaller set of new features that tries to capture as much of the information in the existing features as possible.

2.3.1 Feature selection

Zipf's law might be said to provide the rationale behind the simplest form of dimension reduction: *stop-word removal*.

Stop-words are the grammatical words which are not likely to be useful for searching, such as "and", "but", "the" and "could". A standard list of stop-words will usually cover all of the most frequent words in a text corpus, and in keeping with Zipf's law, this can reduce the amount of text that has to be processed substantially.

Stop-word removal has certain drawbacks. Jurafsky and Martin (2000) note that a standard list of stop-words would reduce the phrase "to be or not to be" to the single word "not". This tells us, however, that stop-word removal is more of a problem in systems which indexes on the multi-word level; in a bag-of-words indexing setting, this Shakespearian phrase is unlikely to yield important information with or without stop-word removal.

Zipf's law tells us that stop-word removal drastically reduces the number of word tokens, but the reduction of word types, and thus of the dimensionality, is modest. *Document frequency thresholding* is a similar approach, but it starts from the other end of the spectrum—by ignoring all rare words. All words that occur in less than m documents of the training corpus, where m is a small number,are not considered as features. This is based on the assumption that infrequent words are not influential on the classification algorithm due to their rare occurences. Apte et al. (1994) justify this selection method based on the statistical properties of low-frequency words. They conjecture that parameter estimates for low-frequency terms are not reliable enough to contribute useful information.

These two methods combined would reduce both the number of word types and word tokens substantially, but it is unclear whether they help the model to find the differences between documents that are consequential for their correct classification. The following methods analyze the class labels in the training data to remove irrelevant features. This is hoped to reduce the noise in the data, and thus make the classification task easier, we can thus call them *feature quality measures*.

The χ^2 test (Yang and Pedersen, 1997) chooses the features that are the least independent from the class label. It is based on a 2×2 contingency table between a predictor term j and a predicted category label. For a given term j and a given category k, let a be the number of documents in k that contain j ("true positives"), b the number of documents in k that do not contain j ("false negatives"), c the number of documents containing j but not belonging to k ("false positives"), and d the number of documents neither in k nor containing j ("true negatives"). Then let $n = a+b+c+d$, the total number of documents in the training set. The χ^2 measure is then:

$$\chi^2 = \frac{n(ad-bc)^2}{(a+b)(c+d)(a+c)(b+d)} \tag{2.2}$$

Forman (2003) defines a new measure called *bi-normal separation* (BNS), which in his study outperforms the traditional χ^2 test. It is defined as:

$$B(j) = |\Phi^{-1}(\frac{a}{a+b}) - \Phi^{-1}(\frac{c}{c+d})| \tag{2.3}$$

where Φ is the standard normal cumulative distribution function. By way of explanation, suppose the occurrence of a given feature in each document is modeled by the event of a random Normal variable exceeding a hypothetical threshold. The prevalence rate of the feature corresponds to the area under the curve past the threshold. If the feature is more prevalent in the positive class, then its threshold is further from the tail of the distribution than that of the negative class . The BNS metric measures the separation between these two thresholds. Forman (2003) also provides a more formal justification by way of receiver operating characteristic (ROC) analysis.

A third feature selection measure is the Pearson product-moment correlation. The two other statistically founded feature measures discussed above, take into account only the presence or absence of terms in documents. The Pearson measure also makes use of term weights in choosing features. Genkin et al. (2007) define it by:

$$r_j = \frac{\sum_{i=1}^{n}(x_{ij} - \bar{x}^{(j)})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_{ij} - \bar{x}^{(j)})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{2.4}$$

where x_{ij} is the frequency of term j in document i, $\bar{x}^{(j)}$ is the mean of x_{ij} across all the documents, $y_i \in \{-1, 1\}$ is the class label for document i, and $\overline{y}$ is the mean of y_i across all the documents. r_j can be positive or negative, but since it is its size that determines the strength of the correlation, $|r_j|$ is used as the feature quality measure. $|r_j|$ measures the degree to which there is a linear relationship between the x_{ij}'s for some term j and the category labels y_i.

The effects of the χ^2, BNS and the Pearson feature selection methods are examined in section 5.2.1.

2.3.2 Feature construction

The alternative approach to reducing the dimensionality by selecting a subset of the features, is to construct a set of new features that tries to incorporate as much as possible of the information value in the existing features, and preferably also tries to generalize them. Two of the main methods for feature construction are *term clustering* (Velldal, 2003) and *Latent semantic indexing* (Deerwester et al., 1990).

For the two main methods we shall be considering, support vector machines and lasso regression, there is however, doubt as to whether dimension reduction is necessary, since both of these in some sense incorporate a form of dimension reduction, as we shall return to.

2.4 Polytomous classification

Some classification tasks are binary, e.g. separating relevant e-mail from spam, but many others are polytomous, i.e. there are more than two classes to choose from. The categorization of Medline abstracts, a problem we shall return to, is one example of a setting where we would like to separate a set of documents into smaller categories, such as "cancer", "cardiovascular disease", "psychiatric disorders" etc.

As Joachims (2002) points out, any polytomous classification task with n classes, can be split into n binary classification tasks. This can be achieved by simply generating n binary classifiers, one for each class, running each document through all of them, and choosing the class which achieves the highest probability score.

There is also an ongoing development of a Bayesian multinomial regression model that can handle polytomous classification directly (cf. Madigan et al. (2004)).

2.5 Evaluation

Evaluation is of course of paramount importance in the development of systems for text categorization—to be able to develop a good classifier, one has to have a precise measure of what "good" is.

There are several ways of evaluating the performance of TC systems, we shall review the most widespread of them in this section. We let $\mathbf{d}$ denote the representation vector of a document. c denotes the document's true category, h denotes the classifier function and $h(\mathbf{d})$ denotes the predicted category of the document. When the classification task is binary, we have that $c \in \{1, -1\}$.

Most of these evaluation measures were originally designed for information retrieval or other fields of machine learning, and they apply only to binary classification. To apply them to the polytomous case, one takes the average over the binary classifications.

2.5.1 The error rate

According to Joachims (2002) the *error rate* is the most common performance measure in machine learning. It can be defined as the probability that the classifier predicts the wrong category:

$$Err(h) = P(h(\mathbf{d}) \neq c|h) \tag{2.5}$$

This is a very intuitive measure, but for the case of TC it is not always very helpful. An error rate of 5% can be very good. Say, for instance, we have a data set of 100 documents, 50 of which belong to category c. If a classifier predicts 55 documents to belong to c, 5 of which do not, this classifier would have an error rate of 5%. If only 5 documents in the data set belong to c and a classifier returns none of them, this classifier could also have an error rate of 5%. Obviously, the latter is a very much less useful classifier than the former.

To get a complete picture of the performance of a classifier, one needs to consider not only the number of erroneous (and correct) classifications, but what kind of errors (and successes)

Table 2.1: A contingency table. T^+, T^-, F^+ and F^- denote the number of true positives, true negatives, false positives and false negatives, respectively.

		Predicted value $c = 1$	Predicted value $c = -1$
True value	$c = 1$	T^+	F^-
	$c = -1$	F^+	T^-

they are.

2.5.2 Precision and recall

Precision and recall have come to be the standard measures of quality in information retrieval, and are also widely used in TC. We define precision of a classifier h, $\pi(h)$, as the probability that a document $\mathbf{d}$ classified as belonging to c, is classified correctly:

$$\pi(h) = P(y = 1|h(\mathbf{d}) = 1, h). \tag{2.6}$$

While precision is a measure of the exactness of the classifier h, recall is a measure of its completeness. It is defined as the probability that a document belonging to c, is classified correctly:

$$\rho(h) = P(h(\mathbf{d}) = 1|c = 1, h). \tag{2.7}$$

Now that we have defined the measures in a probabilistic framework, we will proceed to present their empirical estimators.

2.5.3 The contingency table

Given $\mathbf{d} \in c$ and a classifier $h(\cdot)$, the value of $h(\mathbf{d})$ falls into one of four classes: If the classification is correct, it is either a true positive or a true negative; if it is erroneus, a false positive or a false negative. The counts of these four types of classifications are commonly represented in a *contingency table* as shown in table 2.1. We let T^+ denote the number of true positives and T^- the number of true negatives; and similarly F^+ and F^- denote the number of false positives and negatives, respectively.

These four numbers are used to form empirical estimates of the evaluation measures defined above:

$$\hat{Err} = \frac{F^+ + F^-}{T^+ + F^+ + F^- + T^-} \tag{2.8}$$

$$\hat{\pi} = \frac{T^+}{T^+ + F^+} \tag{2.9}$$

$$\hat{\rho} = \frac{T^+}{T^+ + F^-} \tag{2.10}$$

We see that equations (2.8) through (2.10) are all variants of dividing the number of favorable outcomes by the number of possible outcomes.

A contingency table is not only an aid for computing empirical estimates. It might be claimed to provide the best overview of the performance of a classifier for a given categorization task. However, contingency tables are of little use for comparing results across different data

sets, since its elements are not normalized for the total number of documents. Precision and recall are an option, but for comparing different classifiers, it is inconvenient to have two measures.

2.5.4 Combining precision and recall

We would like a single measure that evaluates the quality of a classifer. We have seen that the error rate has its shortcomings in this respect. To avoid these,the following methods attempt to incorporate both the dimension of recall and and that of precision in a single measure, although using one rather than two measures will necessarily lead to a loss of information.

Precision/recall breakeven point

Many TC algorithms will return a ranking of the documents in the test set according to how likely they are to belong to category c. The principle of the precision/recall breakeven point (PRBEP) is to classify the k top ranked documents as belonging to c, choosing k such that the estimated values for precision and recall are equal. It is easily verified that $\hat{\pi} = \hat{\rho}$ when the number of predicted positives equals the number of actual positives, i.e. when $T^+ + F^+ = T^+ + F^-$.

F_β measures

The weighted harmonic mean of precision and recall is commonly referred to as an F_β measure, and is defined as

$$F_\beta(h) = \frac{(1+\beta^2)\pi(h)\rho(h)}{\beta^2\pi(h) + \rho(h)} \tag{2.11}$$

where β is the weighting parameter. The F_1 measure, giving equal weight to precision and recall, is by far the most used of these.

The empirical estimate of an F_β measure is given by

$$F_\beta(h) = \frac{(1+\beta^2)T^+}{(1+\beta^2)T^+ + \beta^2 F^- + F^+} \tag{2.12}$$

The F_β measures are often credited to C. J. van Rijsbergen (cf. e.g. Genkin et al. (2007)). In his classic book on information retrieval (van Rijsbergen, 1979), however, he defines a similar but different measure, the E measure:

$$E(h) = \frac{1}{\beta\frac{1}{\pi(h)} + (1-\beta)\frac{1}{\rho(h)}} \tag{2.13}$$

The E measure is still in use (cf. Baeza-Yates and Ribeiro-Neto (1999)), although it is far less common than the F_β measures.

TREC measures

The measures T11U and T13U both originate from the Text REtrieval Conference (TREC), and both give high weights to true positives. T11U is given by

$$\text{T11U} = 2T^+ - F^+, \tag{2.14}$$

and T13U is given by

$$\text{T13U} = 20T^+ - F^+. \tag{2.15}$$

2.6 Standard methods for text categorization

In this section we will present the most common TC methods, before we go on to introduce support vector machines in section 2.7.

2.6.1 Decision Tree Classifiers

One drawback of many common approaches to TC such as SVM is that they are not easily interpretable by humans. Decision tree classifiers are a *symbolic* type of algorithm that lends itself more easily to interpretation.

The key idea behind DTCs is a divide and conquer strategy: The text corpus is recursively divided in two parts, until all the documents in both parts belong to the same category. Thus, it is a quite similar method to the CONSTRUE system outlined in section 2.1, actually the example we gave of a rule from CONSTRUE, can be expressed as a DTC, as seen in figure 2.1. However, whereas the CONSTRUE system had to be developed by experts, DTCs are learned automatically. The recursive algorithm is as follows:

1. Check whether all the documents belong to the same category.
2. If not, select a term t and partition the documents into classes which have the same value for t, and place each such class in a subtree.
3. Repeat for all the subtrees.

The key element in a DTC algorithm is the choice of term t on which to base the partition. The two most popular of the standard packages for DT learning, ID3 and C4.5, use slightly different criteria for choosing t, but both are based on Shannon's concept of *information entropy* (Shannon, 1948).

A tree generated in this way, is prone to overfitting, thus most methods also include a "pruning" step, i.e. they remove the overly specific branches of the tree.

In figure 2.1 all the partitions are binary; they need not be. However, most such classifiers use binary document representations, and thus consist of binary trees (Sebastiani, 2002).

2.6.2 Decision Rule Classifiers

Decision rule classifiers are similar to DTCs, in that they are symbolic and are learned inductively. But whereas DTCs are learned in a top-down manner, DRCs are usually built bottom up. A trained DRC consists of a set of DNF rules, it is thus even more similar to the CONSTRUE system than a DTC. Recall that a DNF is a disjunction of conjunctive clauses, i.e. a statement of the form

$$\bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} Q_{ij}$$

where each Q_{ij} is either a statement variable or the negation of a statement variable. It is a well-known fact that any statement of logic that is not a contradiction is equivalent to a DNF (see e.g. Hamilton (1988)).

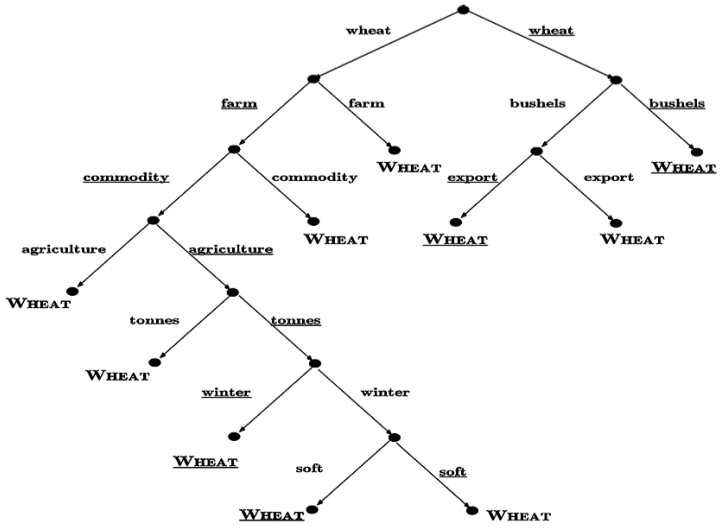


Figure 2.1: An example of a Decision Tree Classifier, amended from the CONSTRUE system. Underlining denotes negation. (Figure from Sebastiani (2002).)

A common way to train a DRC is to initially view every document d_i in the training set as a DNF rule of the form

$$(t_1 \wedge t_2 \wedge ... \wedge t_n) \vee \neg c_i, \tag{2.16}$$

where $t_1 ... t_n$ are the terms contained in document d_i, and c_i is the category of d_i.[2] (2.16) is equivalent to

$$(t_1 \wedge t_2 \wedge ... \wedge t_n) \rightarrow c_i,$$

i.e. if a document contains the same terms as d_i, then it belongs to the same category as d_i. The conjunction of all these DNF rules (one for each document) is already a complete classifier, but it is obviously a very overfitted one. The generalization process of the DRC has two steps: First the logical formula is simplified through a series of modifications, to a more compact, but logically equivalent form; then a "pruning" phase similar in spirit to that employed in a DTC is applied, where the ability to classify *all* the documents correctly is traded for more generality.

An advantage of DRCs over DTCs is that they tend to produce more compact classifiers (Sebastiani, 2002). The appeal of both these methods might be said to lie mainly in their being easily interpretable, however.

2.6.3 Regression methods

Regression models have been common in TC (cf. Lewis and Gale (1994), Schütze et al. (1995) and Ittner et al. (1995)). In a machine learning setting, *regression* is commonly defined as the approximation of a real-valued function f, by means of a function $\hat{f}$ that fits the training data. This might be said to be a slightly more general definition than what is common in statistics. In the *Linear least-squares fit* (LLSF) model, used by Yang and Chute (1994), each

[2]It is worth noting here, that the document representation here is binary; if a standard TF-IDF scheme were used, the pruning step of the DRC would be rendered impossible.

document d_j is associated with two vectors, one "input vector" $\boldsymbol{I}^{(d_j)}$ of $|\mathcal{T}|$ term weights, and one "output vector" $\boldsymbol{O}^{(d_j)}$ of dimension $|\mathcal{C}|$, i.e. with one dimension for each category c_i. For a document d_j in the training set, the element $O_i^{(d_j)}$ equals 1 if $d_j \in c_i$ and zero otherwise; for a document d_k in the test set, the elements of $\boldsymbol{O}^{(d_k)}$ will be probabilities.

Thus, the LLSF method amounts to attempting to find a $|\mathcal{C}| \times |\mathcal{T}|$ matrix $\hat{M}$ such that

$$\hat{M}\boldsymbol{I}^{(d_j)} = \boldsymbol{O}^{(d_j)}.$$

Let n be the number of documents in the training set, and let $I \in |\mathcal{T}| \times n$, be a matrix with the vectors $\boldsymbol{I}^{(d_j)}$ as columns, and $O \in |\mathcal{C}| \times n$, a matrix with the vectors $\boldsymbol{O}^{(d_j)}$ as columns. We can then make an estimate of M by

$$\hat{M} = \operatorname*{argmin}_M \|MI - O\|_F$$

where $\|\cdot\|_F$ denotes the Frobenius norm, given by $\|A\|_F = \sqrt{\sum_{i=1}^{|\mathcal{C}|} \sum_{j=1}^{|\mathcal{T}|} a_{ij}^2}$.

$\hat{M}$ can be computed by performing a singular value decomposition (SVD) on the training set. Its elements $\hat{m}_{ik}$ represent the degree of association between the term t_k an the category c_i.

Yang and Chute (1994) and Yang and Liu (1999) report good results for the LLSF method. However, since SVD is a computationally very costly procedure, LLSF is inferior in effectiveness compared to most other methods.

2.6.4 Naïve Bayes classifiers

The Naïve Bayes classifiers use a probabilistic model of text to estimate $P(c|\mathbf{d})$, the probability that the document represented by the term vector $\mathbf{d} = (t_1, t_2, ..., t_{\mathcal{T}})$ belongs to category c. This probability is computed by an application of the Bayes' theorem:

$$P(c|\mathbf{d}) = \frac{P(c)P(\mathbf{d}|c)}{P(\mathbf{d})} \propto P(c)P(\mathbf{d}|c). \tag{2.17}$$

Estimating the marginal probability $P(\mathbf{d})$ would be problematic, since the number of possible vectors $\mathbf{d}$ is too high. Fortunately, $P(\mathbf{d})$ is constant for all categories c and can therefore be omitted. We do however need an estimate for $P(\mathbf{d}|c)$, i.e. for the probability of the term vector $\mathbf{d}$, given the category c. To alleviate this problem, it is common to make the assumption that any two coordinates of the document vector are, when viewed as random variables, independent of each other. Thus, we have that

$$P(\mathbf{d}|c) = \prod_{i=1}^{(|\mathcal{T}|)} P(t_i|c).$$

It is because of this assumption of independence of the terms, which is in many cases obviously violated, that this kind of classifier is characterized as "naïve". Many efforts have been made to relax the independence assumption for this classifier, but these efforts have not shown the performance improvements that were hoped for (cf. e.g. van Rijsbergen (1977) and Koller and Sahami (1997)).

However, empirical results show that it performs surprisingly well in many domains containing clear attribute dependencies. Domingos and Pazzani (1997) has provided some theoretical justification for the fact that this assumption seldom harms the effectiveness of the classifier. They point out that while it is known that the Naïve Bayes classifier is optimal when attributes are independent given the class, there might also be sufficient conditions for its optimality. Although the Bayesian classifier's probability estimates are only optimal under quadratic loss if the independence assumption holds, the classifier itself can be optimal under zero-one loss (misclassification rate) even when this assumption is violated by a wide margin. The region of quadratic-loss optimality of the Bayesian classifier is in fact a second-order infinitesimal fraction of the region of zero-one optimality. This indicates that the Bayesian classifier may have a much greater range of applicability than its apparent "naïvety" would suggest. However, Bayesian logistic regression, which we shall introduce in chapter 3, provides more flexible framework than Naïve Bayes can give us.

2.6.5 The Rocchio method

The Rocchio method is originally created for an application in information retrieval, specifically for relevance feedback in the vector space model (Rocchio, 1971). An adaption for TC was proposed by Hull (1994), and Rocchio classifiers have been used extensively in TC research since then.

The basic principle of the Rocchio method is to find a "prototypical example" of each category, and then classify a given document to the category whose prototypical example it lies closest to, according to the metric chosen. The prototypical examples are represented as document vectors with weights w_i. For a category c, these weights are given by:

$$w_i = \frac{a}{|c|} \sum_{d_j \in c} w_{ij} - \frac{b}{|\mathcal{D}| - |c|} \sum_{d_j \notin c} w_{ij} \tag{2.18}$$

where w_{ij} is the weight of feature i in document j, $|\mathcal{D}|$ is the number of documents in the training set, $|c|$ is the number of documents belonging to category c, and a and b are constants that determine the relative weight of positive compared to negative examples. If b is set to be zero, the prototypical example of a category will simply be its centroid in the feature space. Traditionally b is non-zero, but substantially smaller than a, so that positive examples are given the most importance. Ittner et al. (1995) and Joachims (1997) use $a = 16$ and $b = 4$.

The main appeal of the Rocchio classifier is that it is easy to implement and computationally cheap. However, it often gives mediocre results, and has generally been considered an underperformer. Especially for TC tasks where the categories form disjoint clusters, the Rocchio method proves problematic, since the centroid can fall outside all of them.

Schapire et al. (1998) has proposed an enhancement to the basic Rocchio framework, based on the idea of using *near positives* instead of negatives in the second term of (2.18). Rather than summing over the entire set $\{d_j | d_j \notin c\}$, Schapire et. al. use a well-chosen subset of documents that are similar to those of category c, but do not belong to it. These are thought to be more helpful for discriminating positive from negative examples. By combining this method with dimension reduction and a method called *dynamic feedback optimization*, Schapire et. al. achieved good results while maintaining the very short training time. This has spawned a certain degree of renewed interest in the Rocchio classifier (cf. Cohen and Singer (1996) and Yang (1999)).

2.6.6 The k nearest neighbors classifier

The k nearest neighbors (kNN) classifier is one of the simplest, but also one of the best performing TC methods. In principle it does exactly what its name would suggest: categorize a document d based on the categories of its k nearest neighbors—ie, the k documents whose vector representations lie closest to $\mathbf{d}$ in the feature space, according to the chosen metric.

The choice of k should ideally be done by cross-validation, but is often chosen a priori. Occasionally, kNN is used with $k = 1$, but this is not common in TC applications. Larkey and Croft (1996) used $k = 20$, while Yang (1999) found $30 \leq k \leq 45$ to give the best effectiveness. It is natural to assume that the ideal value of k will depend both on the size of the data set, and on the document representation and metric chosen.

Some implementations of kNN use a threshold for how large a proportion of k that has to belong to a category before d is classified. For instance, for a kNN with $k = 40$, a document d might not be categorized, if no more than 5 of its 40 nearest neighbors belong to any single category. Yang (1994) introduces a distance-weighted version of kNN, where the influence of the k nearest neighbors, depend on their similarity to the document to be classified. This makes it easier to resolve ties in the single-label situation.

kNN is a robust classifier in the sense that it does not require the categories to be linearly separable. Since it does not build any formal representation of the categories, but only considers a document's "neighbors", kNN is able to accomodate complex category structures that a Rocchio classifier can not, e.g. a case where a category forms disjoint clusters.

Classifiers that do not attempt to create a generalized model for each category c, but categorizes a document simply based on the categories of similar documents in the training set, are called *example-based classifiers*. They have also been referred to as *lazy learners* since "they defer the decision on how to generalize beyond the training data until each new query instance is encountered" (Mitchell, 1996). This points us to the main problem of the kNN classifier, namely its computational inefficiency at classification time. For each test document, its similarity to all of the documents in the training set has to be computed. For large data sets this can be computationally very expensive. One could claim that kNNs do not have a true training phase and thus defer all the computation to the test phase.

2.7 Support Vector Machines

The support vector machine (SVM) method is a very popular approach to many machine learning tasks, and after it was introduced in TC by Joachims (1998), it has become one of the dominant methods in current research.

The name "support vector machine" has perplexed many. We shall return to the meaning of the phrase *support vector* in section 2.7.4. As for the last part of the name, the term *machine*, it probably derives from the expression *learning machine*. In some of the literature on Artificial intelligence, a learning machine is taken to mean any *function estimation algorithm*. As we shall see, SVMs are exacly this—they optimize a function. In this chapter we use the term learning machine in this sense.

In this thesis, SVMs are used mainly as a benchmark for the Bayesian techniques (which we will present in the next chapter), so we will not give any detailed or exhaustive presentation of SVMs. As it is such a central method in TC today, however, we have found it important to give some background to this field; and we have attempted not to limit ourselves to the practical algorithms, but to provide a glimpse of the underlying theory as well.

2.7.1 The VC dimension

The theory of SVMs is based on Vapnik's principle of structural risk minimization (SRM) (see Vapnik (1982)).

This principle might be said to be a formalization of the trade-off between a model's complexity and its success at fitting the training data—where a too complex model will be prone to overfitting the data. The SRM principle applies not only to SVMs, but to all *learning machines*, i.e. all function estimation algorithms.

Burges (1998) gives a simple illustration of this principle:

> Roughly speaking, for a given learning task, with a given finite amount of training data, the best generalization performance will be achieved if the right balance is struck between the accuracy attained on that particular training set, and the "capacity" of the machine, that is, the ability of the machine to learn any training set without error. A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from anything she has seen before; a machine with too little capacity is like the botanist's lazy brother, who declares that if it's green, it's a tree. Neither can generalize well. The exploration and formalization of these concepts has resulted in [the SRM principle].

This notion of "capacity" is measured by the Vapnik Chervonenski (VC) dimension. We shall now give a brief introduction to this concept.

For a binary text categorization task, suppose we have l documents represented by n-dimensional vectors $\mathbf{d}_1$, ..., $\mathbf{d}_l$ and their associated classifications c_1, ..., c_l, where $c_i \in \{-1, 1\}$. Assume further that these data are independently and identically distributed with an unknown distribution $P(\mathbf{d}, c)$.

Now suppose we want to learn the mapping $\mathbf{d}_i \mapsto c_i$. This learning machine is defined by a set of possible mappings $\mathbf{d} \mapsto f(\mathbf{d}, \alpha)$, where $f(\mathbf{d}, \alpha)$ is a function with domain $\{-1, 1\}$, and α is a set of adjustable parameters. A particular choice of α gives us a *trained machine*.

The expectation of the test error, commonly referred to as the *risk*, is given by

$$R(\alpha) = \int \frac{1}{2}|y - f(\mathbf{d}, \alpha)| \mathrm{d}P(\mathbf{d}, y)$$

When we do not have an estimate for the probability $P(\mathbf{d}, y)$, however, this formula is of little help. The empirical risk on the other hand, is calculated simply as the measured mean error rate on the training set:

$$R_{\text{emp}}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(\mathbf{d}_i, \alpha)|$$

Vapnik (1995) shows that a bound for $R(\alpha)$ can be expressed in terms of $R_{\text{emp}}(\alpha)$. Choose a number η such that $0 \leq \eta \leq 1$. Then, with probability $1 - \eta$, the following bound holds:

$$R(\alpha) \leq R_{\text{emp}}(\alpha) + \sqrt{\frac{h(log\frac{2l}{h} + 1) - log\frac{\eta}{4}}{l}} \tag{2.19}$$

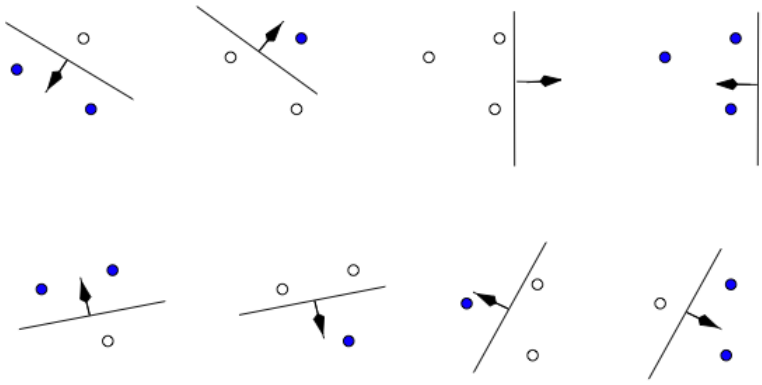


Figure 2.2: shatterlines

where l is still the number of documents, and h is the VC dimension. We see that when we know the value of h, the right-hand side of (2.19) is readily computed. So, given a set of candidate learning machines (i.e., families of functions $f(\mathbf{d}, \alpha)$, and choosing a fixed and preferrably small value for η, the machine which minimizes the the right-hand side of (2.19), is the machine with the lowest risk bound. This gives a principled method for choosing a learning machine for a given task, and is the essential idea of Vapnik's structural risk minimization (Burges, 1998).

To clarify how the VC dimension quantifies the notion of "capacity", we shall provide a definition and a couple of simple results.

The VC dimension has slightly different definitions for different classes of function f. For the binary classification task, our primary concern in this thesis, we need only consider functions with $\{-1, 1\}$ as their domain. We say that a set of data points is *shattered* by a set of functions $\{f(\cdot, \alpha)\}$, if for every possible labeling of the data points, we can find an $f \in \{f(\cdot, \alpha)\}$ which correctly assigns those labels. The VC dimension for the set of functions $\{f(\cdot, \alpha)\}$ is defined as *the maximum number of data points that can be shattered by* $\{f(\cdot, \alpha)\}$.

The VC dimension of a line in $\mathbb{R}^2$ is 3. Figure 2.2 shows how 3 points in $\mathbb{R}^2$ are shattered by oriented lines. We note that it is easy to find a set of 3 points in $\mathbb{R}^2$ that cannot be shattered by a line: Simply place a data point of class c_1 somewhere on the line between two data points of class c_2. This illustrates the fact that the VC dimension denotes the *maximum* number of points that can be shattered. This example is generalized[3] in the following theorem from Burges (1998):

Theorem 2.7.2. *Consider some set of m points in $\mathbb{R}^n$. Choose any one of the points as origin. Then the m points can be shattered by hyperplanes if and only if the position vectors of the remaining points are linearly independent.*

The proof involves a certain level of technicality; we omit it here, and refer the interested reader to the appendix in Burges (1998). We can, however deduce a simple corollary from this theorem.

Corollary 2.7.3. *The VC dimension of the set of oriented hyperplanes in $\mathbb{R}^n$ is $n + 1$.*

[3]Recall that a hyperplane in $\mathbb{R}^2$ is a line.

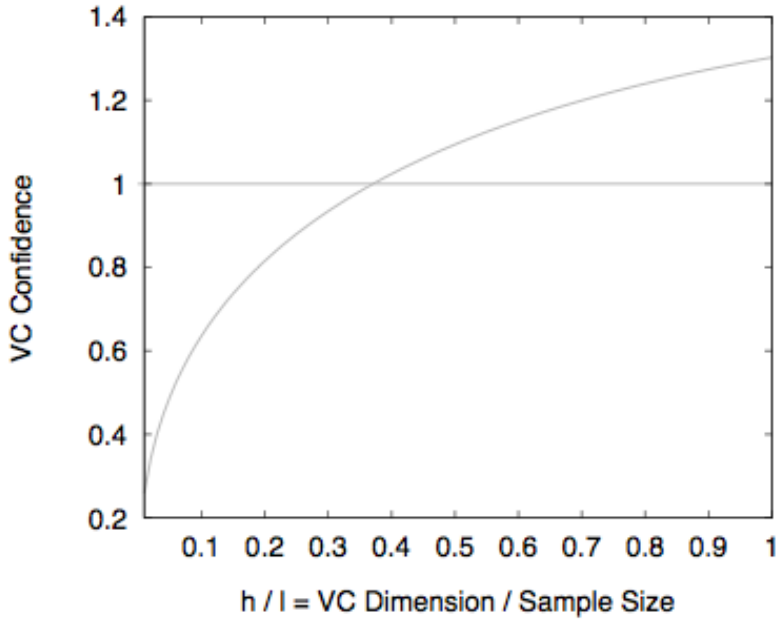


Figure 2.3: vc-mono

This is easily seen: We can always choose $n + 1$ points and then choose one of them as origin, such that the position vectors of the remaining n points are linearly independent; but we can never choose $n+2$ such points, since no $n+1$ vectors in $\mathbb{R}^n$ can be linearly independent.

From the definition of VC dimension we see that a more complex learning machine (i.e. one with higher VC dimension) will generally be able to classify better, thus reducing the first term of (2.19), $R_{\text{emp}}(\alpha)$. On the other hand, the second term of the right-hand side of (2.19) is often referred to as the *VC confidence*, and as can be seen from figure 2.3, it is a monotonic increasing function of h. Thus, to minimize the bound on the risk $R(\alpha)$, there is a balance to be struck between keeping h and $R_{\text{emp}}(\alpha)$ low, at the expense of one another.

2.7.4 Linear and non-linear classification with SVMs

In geometrical terms, an SVM classifier attempts to find the hyperplane in the feature space that separates the positive from the negative instances with the maximum possible margin. Figure 2.4 shows a simple example in $\mathbb{R}^2$, where positive instances (eg, documents belonging to the class in question) are marked off with solid dots, and negative instances are marked off with circled dots. The margin is delimited by the hyperplanes H_1 and H_2, and the optimal separating hyperplane lies in the middle between these. Those training points whose removal would change the solution found, i.e. those points that lie on H_1 or H_2, are called *support vectors*. This will in almost all cases be a very small subset of the vectors in the training set. Since the hyperplane in an SVM is thus fully determined by the support vectors, this entails that a large portion of the training data have no influence on the solution.

In the simplest case, the classes are linearly separable. Requiring linear separability, however, is a big restriction on the applicability of SVMs. For one, they become very vulnerable to mislabelled data. In a TC setting you must expect some degree of error in your training set, a single mislabelled example could ruin the linear separability of a data set. Then there

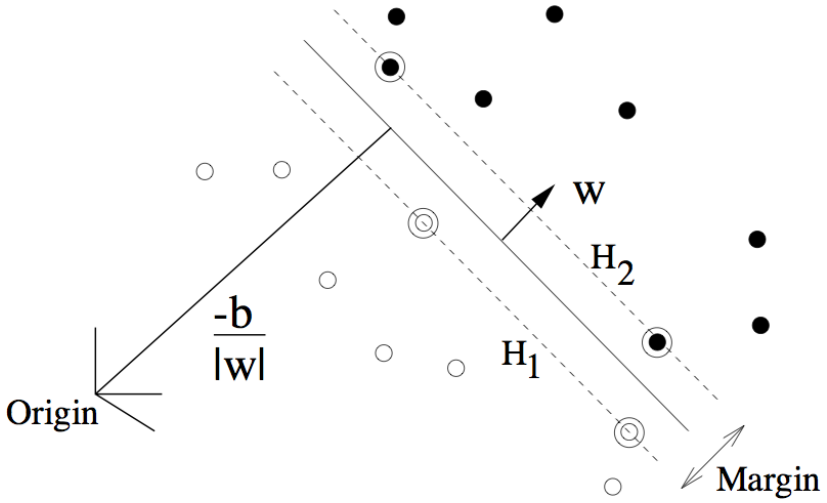


Figure 2.4: Hyperplane. Illustration from Burges (1998).

are the datasets that are not linearly separable to begin with. A simple example would be that the data points of category c_1 were gathered in a clustered formation, with data points of category c_2 scattered around them.

SVMs first gained momentum when Vapnik (1982) found a way around this, by mapping the data to some other Euclidean space (possibly of infinite dimension).

It is customary to give a Lagrangian formulation of the SVM problem. In such a formulation, the data points only appear in the form of dot products between vectors—we shall shortly see why this is a major advantage. The primary Lagrangian formulation is (cf. Burges (1998) for the details):

$$L_P = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{l} \alpha_i c_i (\mathbf{d}_i \cdot \boldsymbol{w} + b) + \sum_{i=1}^{l} \alpha_i \tag{2.20}$$

where the α_is are the Lagrange multipliers, $\boldsymbol{w}$ is normal to the separating hyperplane, and the points which lie on the hyperplane satisfy $\boldsymbol{w} \cdot \mathbf{x} + b = 0$. We wish to minimize this expression, subject to the constraints $\alpha_i \geq 0$. This is a convex quadratic programming problem, thus we can equivalently solve the following dual problem:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j c_i c_j \mathbf{d}_i \cdot \mathbf{d}_j \tag{2.21}$$

The fact that the SVM problem is convex also entails that the Karush-Kuhn-Tucker (KKT) conditions are not only necessary, but also sufficient for $\boldsymbol{w}$, b and α to be a solution. And this entails that the SVM problem is also equivalent to finding a solution to the KKT conditions (Fletcher, 1987). Thus we have several different approaches for solving our SVM problem.

We see that the only way in which the data appears in the training problem (2.21), is in the form of dot products $\mathbf{d}_i \cdot \mathbf{d}_j$. This enables us to approach the problem of nonlinear SVM problems from a new angle. Let ψ be a mapping of the data to some other (possibly infinite dimensional) Euclidean space $\mathcal{H}$:

$$\psi : R^{|\mathcal{T}|} \mapsto \mathcal{H}$$

where $|\mathcal{T}|$ as before is the number of feature terms, giving the dimensionality of the document vectors. Now, since the document vectors only appear in (2.21) as dot products, the training algorithm would only depend on them through dot products in $\mathcal{H}$, i.e. on functions of the form $\psi(\mathbf{d}_i) \cdot \psi(\mathbf{d}_j)$. This entails that if there were a "kernel function" K such that $K(\mathbf{d}_i, \mathbf{d}_j) = \psi(\mathbf{d}_i) \cdot \psi(\mathbf{d}_j)$, we would only need to use K in the training algorithm, and would never need to explicitly even know what ψ is. One example of a kernel function is the Gaussian Radial Basis function:

$$K(\mathbf{d}_i, \mathbf{d}_j) = \mathrm{e}^{-\|\mathbf{d}_i - \mathbf{d}_j\|^2/2\sigma^2} \tag{2.22}$$

In this particular example, $\mathcal{H}$ is a Hilbert space of infinite dimension, so it would not be very easy to work with ψ explicitly. However, we do not need to. If one replaces $\mathbf{d}_i \cdot \mathbf{d}_j$ by $K(\mathbf{d}_i, \mathbf{d}_j)$ everywhere in the training algorithm, the algorithm will produce a support vector machine in the infinite dimensional space. Note that we are still doing a linear separation, but we are doing it in a different space.

This completes our presentation of SVM. We consider SVM's potential as superior to the other methods surveyed in section . However, we have chosen another method as our main focus in this thesis, namely Bayesian logistic regression, to which we shall now turn our attention.

3

Bayesian logistic regression

Like the Naive Bayes classifier of section 2.6.4, Bayesian logistic regression (BLR) employs a statistical framework, and tries to estimate the conditional probability of a document represented by a vector $\mathbf{d}$ belonging to a category c, $P(c|\mathbf{d})$. Rather than employing Bayes rule to establish the relation $P(c|\mathbf{d}) \propto P(c)P(\mathbf{d}|c)$, BLR introduces a vector of weighting parameters $\boldsymbol{\beta}$, and conditions the probability $P(c|\mathbf{d})$ on this $\boldsymbol{\beta}$:

$$P(c|\boldsymbol{\beta}, \mathbf{d}) \propto P(\boldsymbol{\beta})P(\boldsymbol{\beta}|c, \mathbf{d}).$$

In this chapter we shall present BLR in overview. We shall see how both linear and logistic regression are variants of *generalized linear models*. We shall discuss the properties of the logistic component of BLR, and examine the Bayesian strategies such as choice of prior distribution. We shall then look briefly into how the theory of BLR compares to that of SVM; before we go into some detail on the toolkit we have employed for our experiments, and what extensions we have attempted to add to it.

3.1 Linear discriminants by least squares

Fisher's linear discriminant (Fisher, 1936) provides a natural starting point. It is an empirical method for classification that is analogous to linear SVMs in so far as it tries to find a linear separation of the data points by way of a hyperplane. For the binary case of two categories, where a document is represented by a vector $\mathbf{d} \in \mathbb{R}^p$, we have a linear combination of feature values

$$q(\mathbf{d}) = \boldsymbol{\beta}^{\mathrm{T}}\mathbf{d} = \sum_i \beta_i d_i.$$

We call $q(\cdot)$ the *discriminant*, and we wish the discriminants for the two classes to vary as much as possible. We can define a simple measure for this: Let $\overline{\mathbf{d}}_1$ and $\overline{\mathbf{d}}_2$ be the means of the feature vectors of the two classes, respectively,and let s_g be the standard deviation of the discriminants $g(\mathbf{d})$, calculated over all the documents $\mathbf{d}$. Then consider the difference between the mean discriminants for the two classes divided by the standard deviation of the discriminants, s_g:

$$\frac{q(\overline{\mathbf{d}}_1) - q(\overline{\mathbf{d}}_2)}{s_g}. \tag{3.1}$$

As Mitchell (1994) points out, this measure of discrimination is related to an estimate of misclassification error based on the assumption of a normal distribution for $q(\mathbf{d})$. We note that this is a weaker assumption than saying that $\mathbf{d}$ has a (multivariate) normal distribution.

If we let the dividing line go through the midpoint between the two class means, we may estimate the probability of misclassification for one class as the probability that the normal random variable $q(\mathbf{d})$ for that class is on the wrong side of the dividing line given by

$$\frac{q(\overline{\mathbf{d}}_1) - q(\overline{\mathbf{d}}_2)}{2}.$$

This probability is easily seen to be

$$\Phi(\frac{q(\overline{\mathbf{d}}_1) - q(\overline{\mathbf{d}}_2)}{2s_g}).$$

Algebraically it is more convenient to use another measure defined in terms of sums of squared deviations, different from, but equivalent to (3.1). The *sum of squares of* $q(\mathbf{d})$ *within class* c is

$$\text{ssq}(c) = \sum_{\mathbf{d}\in c}(q(\mathbf{d}) - q(\overline{\mathbf{d}}_c))^2,$$

where $\overline{\mathbf{d}}_c$ is the mean of the $\mathbf{d} \in c$. Further, let v be the *the pooled sum of squares within classes*:

$$v = \text{ssq}(c_1) + \text{ssq}(c_2),$$

and let t be the *total sum of squares*

$$t = \text{ssq}(c_1 \cup c_2)$$

i.e., the sum of squares over all the documents, in the binary case. The *pooled sum of squares between classes* is $t - v$, and is proportional to $(q(\overline{\mathbf{d}}_1) - q(\overline{\mathbf{d}}_2))^2$.

To perform an F-test for the significance of the difference $q(\overline{\mathbf{d}}_1) - q(\overline{\mathbf{d}}_2)$, we would calculate the F-statistic:

$$F = \frac{(t - v)}{v/(n - 2)}$$

where n is the total number of documents $\mathbf{d}$. To make discriminants differ as much as possible, we can maximize the F-statistic. Since this is equivalent to maximizing the ratio $\frac{t}{v}$, we can choose the coefficients β_i to maximize this simpler relation instead. This maximization problem may be solved analytically, giving an explicit solution for the β_is.

Fisher's linear discriminant is obviously hampered by the same limitations as any other linear classifier. To be able to handle data sets that are not linearly separable, we need to use a logistic discriminant. This would be a form of what is called generalized linear models, and we shall now give a short expositions of the theory behind these.

3.2 Generalized linear models

Generalized linear models (GLMs) are an extension of classical linear models, of which we have just seen one concrete example. McCullagh and Nelder (1989) describes a GLM to have three components: a random component, a systematic component and a link. We will begin by describing classical models in general terms, and go on to show how they can be extended to deal with data sets that are not linearly separable.

For the sake of generality, we shall here depart from our usual notation of documents $\mathbf{d}$ and categories c, and define the GLM in terms of an observed vector $\mathbf{y}$ of dimension n. We will then go on to show how GLMs are used for TC in section 3.2.1.

$\mathbf{y}$ is assumed to be a realization of the random vector $\mathbf{Y}$. The n components of $\mathbf{Y}$ are independently distributed with means given by the n-dimensional vector $\boldsymbol{\mu}$. In a classical linear model, we assume a Normal distribution with constant variance σ_i for the components Y_i. These are strong assumptions; there are many data sets for which independence and constant variance cannot be assumed for the Y_is. We have now specified the random component of the model.

Now, for each observation y_i in $\mathbf{y}$, we have p observations $x_{i1}, x_{i2}, ..., x_{ip}$; we call these the *covariates* of y_i. Thus, we have n p-dimensional covariate vectors, $\mathbf{x}_1, ..., \mathbf{x}_i, ..., \mathbf{x}_n$. We then let $\boldsymbol{\beta}$ be a p-dimensional vector of unknown parameters, such that for $i = 1, ..., n$

$$\mu_i = \boldsymbol{\beta}^{\mathrm{T}} \mathbf{x}_i.$$

We have now specified, for each observation y_i, the mean μ_i of its underlying distribution, in terms of its covariates $\mathbf{x}_i$. This can be expressed in matrix notation as

$$\boldsymbol{\mu} = X\boldsymbol{\beta}$$

where the model matrix X is of dimension $n \times p$. Its rows are the covariate vectors $\mathbf{x}_i$, so x_{ij} is the value of the jth covariate for observation i. This completes the specification of the systematic part of the model.

Thus far the classical linear model. Now, for the generalization, let the βs be such that they produce not the means themselves ($\boldsymbol{\mu}$), but rather a *linear predictor* $\boldsymbol{\eta}$; i.e.

$$\boldsymbol{\eta} = X\boldsymbol{\beta}.$$

$\boldsymbol{\eta}$ and $\boldsymbol{\mu}$ are connected by a mapping $\boldsymbol{\mu} \mapsto \boldsymbol{\eta}$, specified by a *link function* $\psi(\cdot)$, such that

$$\eta_i = \psi(\mu_i)$$

for $i = 1, ..., n$. This constitutes the link between the random and the systematic component of the GLM.

We see then how the classical linear model is a special case of a generalized linear model, with the identity function as the link. In GLMs, any monotonic differentiable function is an acceptable link function. Three principal link functions are: the *logit*:

$$\eta = \log(\frac{\mu}{1-\mu}); \tag{3.2}$$

the *probit*:

$$\eta = \Phi^{-1}(\mu), \tag{3.3}$$

where $\Phi(\cdot)$ is the Normal cumulative distribution function; and the *complementary log-log*:

$$\eta = \log(-\log(1-\mu)). \tag{3.4}$$

In addition to accomodating non-linear correlations in the data sets by the use of the link function, GLMs allow for the distribution of the random component to come from an exponential family other than the Normal, giving further flexibility.

3.2.1 Applying a GLM to TC

In our case, i.e. for text categorization, the observed vector $\mathbf{y}$ would be a vector of categories c_i, one for each of the n documents d_i in our data set. For binary classification, we have that $c_i \in \{-1, 1\}$. The covariate vectors $\mathbf{x}_i$ correspond to the document representations $\mathbf{d}_i$. Thus, for an observed category c_i, the covariate observations are the weighted frequencies of the indexed features $t_1, ..., t_p$ in the document d_i.

In a linear regression model for TC, like the one described in section 2.6.3, we let $\boldsymbol{\beta}^{\mathrm{T}}\mathbf{d}_i$ model the mean μ_i of c_i's distribution directly.[1] The central idea in BLR, however, is using the logit link (3.2) to enable us to capture more complex divisions than the linear ones. The general binary regression model can then be expressed thus:

$$P(c = 1|\boldsymbol{\beta}, \mathbf{d}) = \psi(\boldsymbol{\beta}^{\mathrm{T}}\mathbf{d}) = \frac{1}{1 + e^{-\boldsymbol{\beta}^{\mathrm{T}}\mathbf{d}}} \tag{3.5}$$

We have now introduced logistic regression, and seen how it can be applied to TC. It remains to look into the Bayesian component of BLR.

3.3 Maximum a posteriori estimation

Maximum likelihood (ML) estimation is the standard method used with GLMs. If we wish to estimate an unknown parameter β based on a series of observations $\mathbf{x}$, and let f be the sampling distribution of $\mathbf{x}$, then the maximum likelihood estimate is

$$\hat{\beta}_{ML}(\mathbf{x}) = \underset{\beta}{\operatorname{argmax}}\, f(\mathbf{x}|\beta) \tag{3.6}$$

The standard method for solving this equation is the Newton-Raphson method, which can be regarded as an iterative least squares of sorts. Newton-Raphson employs matrix inversion; for an application in TC, this presents us with two problems. The first is that since ML estimation places no constraints on β, we risk ending up with matrices that do not have full rank, and thus are not invertible. Secondly, the run time of matrix inversion is $O(p^3)$, where p is the dimension of beta. This is tractable for values of p up to about 1000, when it starts to become prohibitively slow. In our case $p = |\mathcal{T}| \approx 30000$, the number of indexed features.

Maximum a posteriori (MAP) estimation address these problems to an extent by treating $\boldsymbol{\beta}$ as a stochastic variable with a prior distribution $g(\boldsymbol{\beta})$. Then the posterior distribution of $\boldsymbol{\beta}$ is

$$\beta \sim \frac{f(\mathbf{x}|\beta)g(\beta)}{\int_B f(\mathbf{x}|\beta')g(\beta')\mathrm{d}\beta'} \tag{3.7}$$

where B is the domain of g and β' is the integration variable. The MAP estimate of β is the mode of this distribution

$$\hat{\beta}_{\mathrm{MAP}}(\mathbf{x}) = \underset{\beta}{\operatorname{argmax}}\, \frac{f(\mathbf{x}|\beta)g(\beta)}{\int_B f(\mathbf{x}|\beta')g(\beta')\mathrm{d}\beta'} = \underset{\beta}{\operatorname{argmax}}\, f(\mathbf{x}|\beta)g(\beta). \tag{3.8}$$

We return to how this estimate can be done in a computationally feasible way in section 3.6.

[1] In the LLSF model of section 2.6.3, the c_is take values 0 or 1, rather than -1 and 1. This is convenient, since it enables us to interpret the mean μ_i the probability of document d_i belonging to the category. Since BLR is not a linear model, this has not been a concern for us.

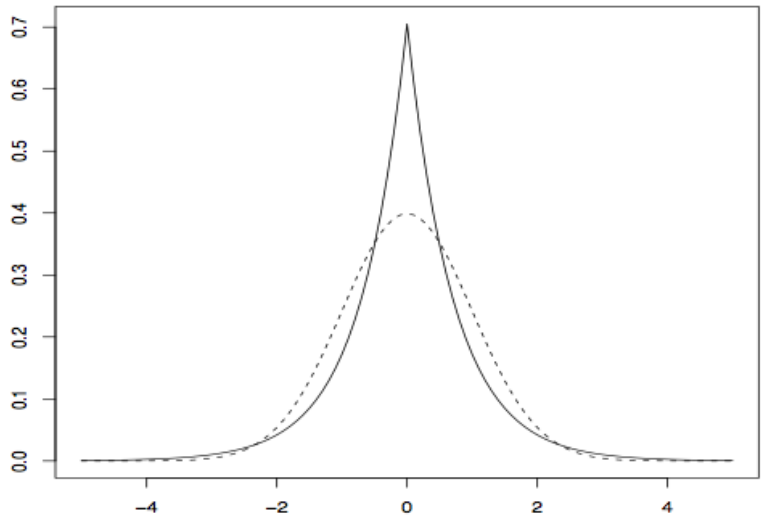


Figure 3.1: Normal and Laplace distributions. Due to its being denser around zero, the Laplace distribution favors sparse solutions when used as a prior for the regression parameters β_i. [Figure from Genkin et al. (2007).]

So whereas ML will find the $\boldsymbol{\beta}$ that maximizes $f(\mathbf{x}|\boldsymbol{\beta})$, a MAP estimate maximizes the product $f(\mathbf{x}|\boldsymbol{\beta})g(\boldsymbol{\beta})$. Since we can choose the prior $g(\beta)$ freely, this gives us a new degree of flexibility. We shall now see how we can use this to address the problem of high dimensionality and overfitting.

3.4 Priors and overfitting

We have seen in section 2.3, how the high dimensionality of the feature space makes TC models prone to overfitting. This makes it advantageous to have prior distribution that favors feature weights around zero. The Normal distribution with mean zero is one possibility.

Regression with a Normal prior is commonly referred to as *ridge regression*, and is widely used in TC. However, while it creates feature weights scattered around zero, and as such makes the model less prone to overfitting, it does not lead to a sparse model. Tibshirani (1996) showed how the Laplace prior was in some respects a better choice of prior distribution for logistic regression. This combination is what is referred to as *lasso regression*. The normal distribution will tend to generate small, but non-zero β_is. A Laplace distribution on the other hand, will generate more zero elements as well as some larger elements as it is denser around zero and has heavier tails (cf. figure 3.1). This might be seen as advantageous, as it can function as a form of feature selection.

Figure 3.2 illustrates this even more clearly, by plotting the estimated $\hat{\beta}$ values influenced by the prior, against what might be called the "true" β values, which would be found by an ordinary least squares regression. We notice how the lasso, unlike ridge regression, is zero for small values of β.

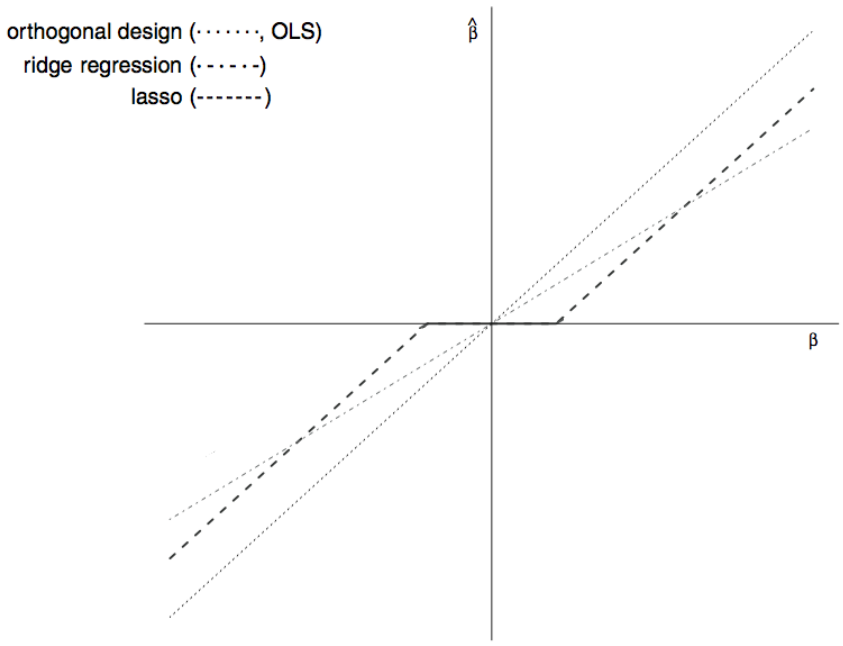


Figure 3.2: The estimated parameters $\hat{\beta}$, plotted against the actual parameters β for ordinary least squares regression, ridge regression and lasso regression. [Figure adapted from Zou and Hastie (2005).]

3.5 Comparison to SVM

As we have seen, BLR and SVMs are based on the same principles, but go about it in slightly different ways. Both are generalized linear classifiers, and make use of mapping the data into a different space where they are easier to separate. As demonstrated by Pelckmans et al. (2004), SVMs can also be seen as a form of ridge regression.

After BLR was introduced in TC by Genkin et al. (2007), it has achieved good results and has been gaining in popularity, but is still behind SVM, which has been around for a while longer.

BLR does however have the advantage of being more easily interpretable in a Bayesian framework, which is part of the reason why we have chosen to pursue it further here. Manipulating the prior distribution of BLR enables us not only to regulate the feature selection in a comprehensible way, but also to incorporate domain knowledge, as we shall see in section 3.7.1.

3.6 The BBR toolkit

In our experiments, we have employed the BBR toolkit for Bayesian binary regression (cf. Genkin et al. (2007) and the BBR homepage[2]). BBR is implemented in C++ and supports both Normal and Laplace priors. It is assumed that the priors for the components of $\boldsymbol{\beta}$ are

[2]URL: http://www.stat.rutgers.edu/ madigan/BBR/

independent, so that the prior for $\boldsymbol{\beta}$ is the product of the priors for its components β_i.

The MAP estimation in BBR is based on the CLG algorithm of Zhang and Oles (2001). This is a *cyclic coordinate descent* optimization algorithm (cf. Luenberger (1984)).

Cyclic coordinate descent algorithms solve multidimensional optimization problems by treating them as a series of one-dimensional problems. It begins by setting all the variables to a given initial value. It then sets the first variable to the value that minimizes the objective function, holding all other variables constant. It then proceeds to find the minimizing value of the second variable, holding all other variables constant, including the new value of the first variable. It proceeds in this manner until all the variables have been traversed, when it returns to the first variable and starts again. Multiple passes are made over all the variables until some convergence criterion is met.

For a ridge regression model, where the Normal prior distributions have mean 0 and variance τ, let y_i be an observation and $\mathbf{x}_i$ its covariate vector. Then, the one-dimensional optimization problems amount to finding the $\beta_i^{(\text{new})}$ that gives the minimum value to the negative log posterior, given by

$$-l(\boldsymbol{\beta}) = \sum_{i=1}^{n} \ln(1 + \mathrm{e}^{-\boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i y_i}) + \sum_{j=1}^{d} (\ln \sqrt{\tau} + \frac{ln2\pi}{2} + \frac{\beta_j^2}{2\tau}),$$

assuming that all the other β_is are held at their current values. This is equivalent to finding the z that minimizes:

$$h(z) = (\sum_{i=1}^{n} f(r_i + (z - \beta_j)x_{ij}y_i)) + \frac{z^2}{2\tau_j}$$

where $f(r) = \ln(1 + \mathrm{e}^{-r})$ and $r_i = \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}_i y_i$ are computed using the current value of $\boldsymbol{\beta}$ and so are treated as constants. The Gaussian penalty terms not involving z are constant and thus omitted.

To solve these optimization problems, BBR follows the procedure of Zhang and Oles (2001) in using a variant of the one-dimensional Newton's method.

The BBR classification module outputs a probability for each document belonging to the class. The document will be assigned to the class if this probability is above a certain threshold. To perform the classification, one can choose either to set the threshold to a default value of 0.5, or tune the threshold to maximize an effectiveness measures on the training set. (See section 2.5 for a discussion of the different effectiveness measures.)

3.7 Extensions to the BBR toolkit

In addition to doing extensive testing of the BBR toolkit with different parameters on our corpora, we have also attempted to make some minor improvements upon it. This section presents in outline these extensions to BBR.

3.7.1 Constructing informative priors

As we mentioned in section 2.1, there is a polarity in Artificial intelligence research, between approaches based on knowledge engineering and approaches based on machine learning. In the machine learning paradigm there might be said to be a tendency towards purism—there is a certain bias towards techniques that do not require *any* human intervention. Machine

translation (MT) can serve as an example. In MT research, the knowledge engineering approach is to build logical rules for translating sentences based on the foundation of a bilingual dictionary of word translations. In the machine learning approach, *statistical machine translation* one traditionally abstains from using dictionaries altogether, and instead tries to build the translation model exclusively from so-called paralell text, i.e. a text corpora consisting of a text in one language and its translation in another language (cf. Næss (2007)).

From a research perspective, it is very interesting to examine how well an algorithm can perform without incorporating any prior knowledge. In practical applications, however, we are more concerned with the final results.

In text categorization, we do not start with a blank slate; we usually do have an idea which words are most relevant for each of the categories. This knowledge alone can bring us a long way; we have seen in section 2.1 how entire TC systems can be built around this. We have touched upon why such knowledge engineering approaches are not ideal: they are labor-intensive in development and tend to give low coverage. However, there are also cases where simple systems based on very modest domain knowledge can outperform advanced sub-symbolic approaches. For a task in the TREC-04 genomics track involving separating the documents that were relevant to the Mouse Genome Informatics system from the irrelevant ones, Hersh et al. (2004) note that using the occurrence of the term "mice" was a better predictor of relevance than anything else, including the complex feature extraction and machine learning algorithms of many participating groups. Thus, it is an enticing idea to try to combine machine learning algorithms with some degree of knowledge engineering, trying to capture the best of both worlds.

There have been some efforts along this direction; most of them are based on a form of *feeature selection* We have discussed general statistical methods for feature selection in section 5.2.1. When trying to exploit domain knowledge by way of feature selection, one can choose to index only the words which occur in certain reference texts, such as a list of keywords that are known to be central to a category. This is a somewhat crude approach as it is a binary one: words are included or excluded from the document representation, and there is no way to distinguish between more and less important words.

The approach we have chosen is inspired by the work of Dayanik et al. (2006), and is based on finding a set of words that are deemed relevant to the categorization task in question, and assigning prior distributions to these words that are different from those of the rest of the indexed words. This amends the mentioned flaw of a crude feature selection: we do not disregard the other words completely, but give higher weight to the words we consider important through their priors. We shall call these words *"knowledge words"*.

How can we adjust the prior distributions of the knowledge words to increase their influence in the classifier? Giving them a prior distribution from a different class of probability functions than the other words, would present substantial implementational problems, so we confine ourselves to using the same probability function for all the words and adjusting its parameters for the knowledge words.

We can choose to adjust either the mode or the variance of the prior distributions. Assigning a higher mode to the priors of the knowledge words than the other words, would make the β values for the knowledge words more likely to have a higher value, and thus more likely to be positive predictors for the category.

A problem of increasing the modes of the knowledge word priors, however, is that for a polytomous classification task it requires us to adjust the priors separately for each category—a good predictor for one category is unlikely to be a good predictor for the others, rather to the

contrary. Increasing the variances of the knowledge word priors is a less vulnerable approach. By giving the prior distribution of a word a high variance, we make it more likely to have a β that is large in absolute value—positive or negative. By increasing the variance of the prior for a knowledge word we also assume that it will tend to be a stronger predictor of class membership than other words, but in this case it can be a negative as well as a positive predictor. Our knowledge words can then be *domain-specific* rather than *category-specific*.

3.7.2 EM algorithm over uncategorized documents

We have previously mentioned our troubles in finding relevant text corpora for the work on this thesis; this is an example of a general problem in TC research: categorized texts are hard to come by. On the other hand, one typically has abundant amounts of *un*categorized text available in digital format. Finding a way to improve a classifier by using uncategorized texts thus has the potential of increasing its performance significantly.

Wu (2006) presents a method for improving a Naive Bayes classifier for images through using an EM algorithm on uncategorized data. We have attempted to adapt this method for use on text categorization with BBR.

Castelli and Cover (1996) note that unlabeled data (e.g. uncategorized documents) can boost learning accuracy because it provides information about the joint probability distribution. The idea in our method, is to let the trained classifier assign "pseudo-categories" for the uncategorized documents, and then try to use these pseudo-categories to improve the classifier through a variant of the Expectation-Maximisation (EM) algorithm, introduced by Dempster et al. (1977).

Suppose that we have our training set, which is a (relatively small) set of categorized documents; and that we have a (possibly much larger) set of uncategorized documents. After training our BLR model on the categorized documents, we can classify the uncategorized documents, thus giving them pseudo-categories. We can then retrain our BLR model on the set consisting of the categorized documents and the uncategorized documents that now have been assigned pseudo-categories. This gives us a new classifier which we can use to re-classify the uncategorized documents, which in their turn enable us to re-train the classifier, and so on. The algorithm thus becomes:

1. Train classifier on the training set.
2. Classify the uncategorized documents with the current classifier, yielding a pseudo-training set.
3. Re-train the classifier on the union of the training set and the current pseudo-training set.
4. Repeat steps 2 and 3 until convergence.

Notice that we do not change the categories of the documents in the training set, we merely augment the training set with a pseudo-training set. By iteratively re-training our classifier, we hope to gradually improve the classifications in the pseudo-training set, which can then improve the classifier further.

4

Data analysis: The text corpora

In all forms of natural language processing one employs *text corpora* as data. Mitkov (2003) defines a text corpus as "a large body of linguistic evidence typically composed of attested language use." This usually amounts to a large number of texts in digital format, collected from a specific domain, and processed to suit the task in question. A corpus for text categorization consists of a set of documents that has been categorized manually.

In this chapter we shall first consider the text corpora used as data in this project, before discussing the processing choices made.

4.1 Choice of text corpora

Finding and amending suitable text corpora turned out to be a substantial part of our work in this project. As mentioned in the introduction, some of the motivation behind this work was to examine ways of reducing the search space for the BioTracer search engine. Since BioTracer's search space consists of Medline abstracts, it was of the essence that our corpus be from the medical domain.

There has been made one extensive corpus of medical texts, the Ohsumed collection (Hersh et al., 1994), a subset of which has previously been amended for text categorization. This is freely available, so initially we were planning to make do with this corpus alone.

In the treatment of text corpora for NLP, however, sources of error abound. The processing of such corpora involves handling very large data files; a small error can make a big difference, and these errors can be hard to find.

The first version we used of Ohsumed contained such an error, which rendered our results unusable. Unsure of where the error lay, we had to turn to other data sources, and ended up using two additional text corpora, both from the Genomics track of the TREC conference, before we eventually got a hold of a correctly processed version of Ohsumed. Thus we have ended up using three fairly different corpora. We shall now discuss each of these in detail.

4.1.1 The TREC Genomics track

The Text REtrieval Conference (TREC) is a series of workshops co-sponsored by the National Institute of Standards and Technology (NIST). A TREC workshop focuses on a set of specific information retrieval tasks, and these are specified in so-called *tracks* (cf. the TREC homepage[1]). Each track lays out a task in detail and provides a data set. TREC is something of a standard benchmark in the IR community.

Because of the growing size and complexity of the biomedical literature, there is increasing effort devoted to structuring this knowledge in databases. The Genomics track started in 2003,

[1] URL: http://trec.nist.gov/tracks.html

and will end in 2007. It focuses on several tasks in the genomics domain, some related to annotating genes, others to more traditional retrieval tasks.

Each year, new versions of the data sets for each track are released. We have used portions of the data sets from 2004 and 2005, we will refer to these as TREC-04 and TREC-05, respectively. TREC-05 is the more suited for our field of inquiry. However, this corpus was only available in a somewhat outdated data format (SGML) which our software could not handle. We eventually managed to parse the SGML, but at the stage when it was still unclear whether this would work out, we turned instead to TREC-04.

TREC-04

The data from the TREC 2004 Genomics track we have employed, is from the so-called *ad hoc retrieval task*. This is essentially a task of standard information retrieval, where a set of documents are labeled as "relevant", "not relevant" or "possibly relevant" according to 50 search queries. The documents are Medline abstracts from the years 1994 to 2003, and the queries are actual queries done by researchers in biology.

Information retrieval and text categorization are different tasks, but they do have a lot of traits in common. We thus thought, as this was the most suitable corpus we had available at the time, that it should be possible to use for an initial evaluation of our TC software. 50 categories would, however, be too many for a TC algorithm to handle—there would be too few documents per category for the classifier to be able to make good generalizations. For this reason, our approach was to merge some of the categories so that we could arrive at a corpus of under 10 separate categories.

Figure 4.1 shows the nine clusters we created from the 50 topics. We selected six of these clusters as our categories; these are shaded in figure 4.1. For each of these clusters, we then selected the two topics for which the number of documents deemed relevant was highest (these topics are also shaded in the figure), and we used these documents as the positive examples of each category. We excluded documents that were deemed relevant for more than one topic. The complete definition of the original 50 topics are defined in the file `04.adhoc.topics.txt`.[2]

TREC-04 is the only corpus where we have had to do the split between test and training sets ourselves. We will end this section by discussing briefly the choices made.

The first approach we attempted was to use one of the selected topics in each cluster as training set and the other as test set. This gave very poor results. In hindsight, it is easy to see why this would be the case: The two (or, in one case, three) topics that any of our categories consist of, encompass somewhat different documents. If we use a corpus generated in the way described above for TC, then we are testing not only the classifier's ability to predict a category, but its ability to make the generalization across two slightly different topics.

Thus, our second approach, leading to the corpus we have used in this thesis, was to let the test and training sets consist of documents from both topics in equal proportions.

In this second approach, we were free to choose the respective sizes of the test and training sets. Concerned that the results be good enough, we made a "90-10 split" (i.e. we used 90% of our corpus as our training set, and the remaininig 10% as our test set). In the TC literature, 50-50 splits seem to be the rule, and our other two corpora are divided in this manner. However, we have not seen any justification for why this choice should be better than a 90-10 split, which is common in computational statistics. Since our data set was of rather modest size, we gave priority to using as much as possible of this data for training.

[2] URL: http://trec.nist.gov/data/genomics/04.adhoc.topics.txt

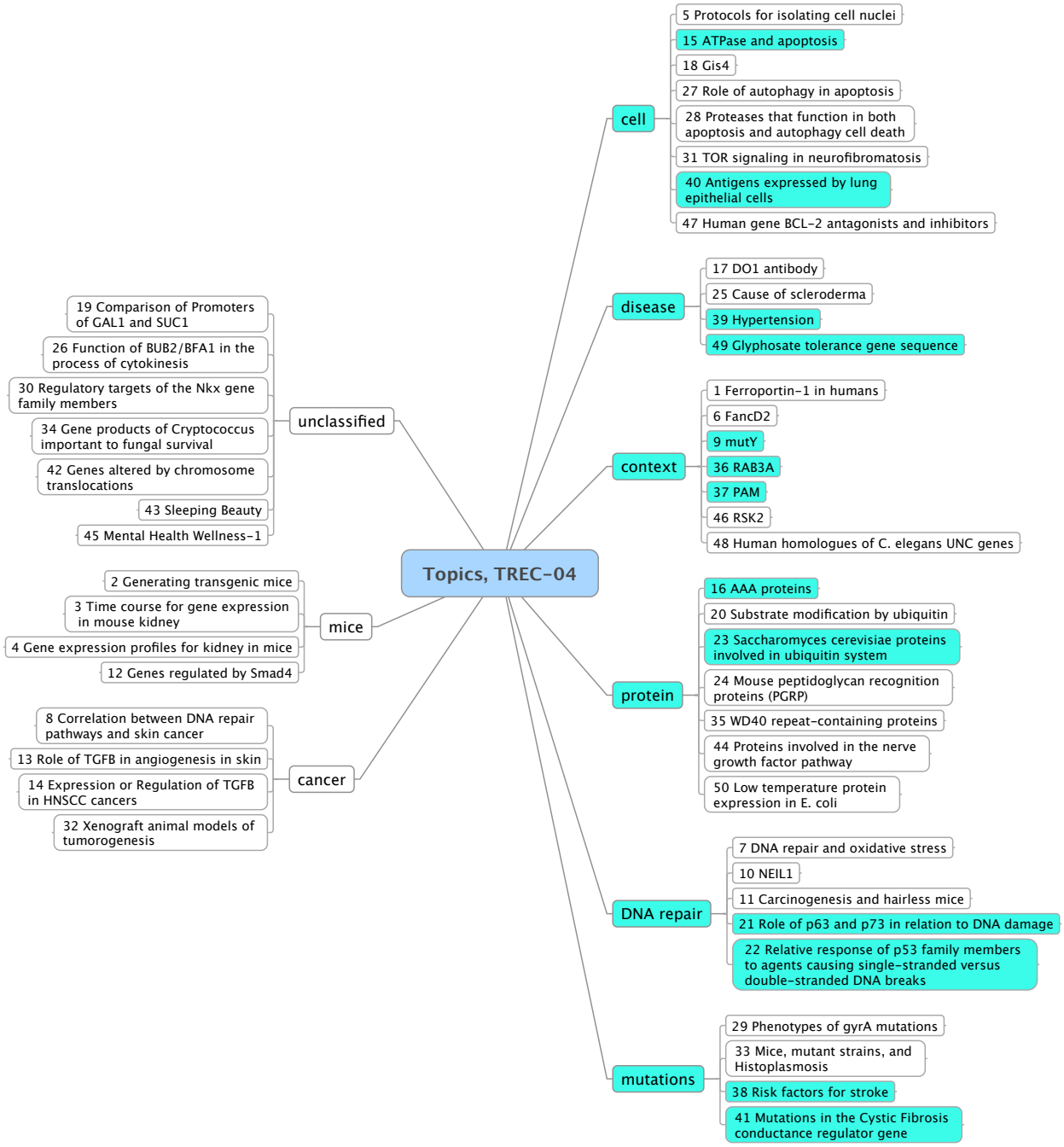


Figure 4.1: Manually created clusters of the 50 topics in the TREC-04 corpus. The shaded clusters and the shaded topics for each cluster, represent the categories used. Five of the categories contain two topics, whilst the last category contains three topics.

Table 4.1: Categories of the TREC-05 corpus.

Category	Description
A	Articles about alleles of mutant phenotypes
E	Articles about embryologic gene expression
G	Articles for GO annotation
T	Articles about tumor biology

As we shall see, this did indeed produce very good results, to the extent that we ask ourselves whether the 90-10 split might have given too small a test set (cf. discussion in section 6.1).

TREC-05

The second data set from TREC that we have used, is from the categorization task of the TREC 2005 Genomics track. This employs data from the Mouse Genome Informatics system (Dunham, 2002), which contains full text articles from three journals over two years. The journals are *Journal of Biological Chemistry*, *Journal of Cell Biology* and *Proceedings of the National Academy of Science*. TREC has chosen to use the articles from 2002 as training set, and the articles from 2003 for testing; we have done the same. There were 4 categories (cf. table 4.1). One of these (G) was from a previous TREC track, where the task was to single out the articles for which it was relevant to apply tags from the gene ontology (GO) hierarchy. The remaining three encompassed various topics in genomics research.

As mentioned, we had some issues with the SGML data format that this corpus is distributed in. After extended efforts at converting the corpus to XML, we ended up taking a different approach. We had access to the relevance judgements for all the articles in the corpus, and through Medline we also had access to the abstracts of all articles of the three journals from which the articles in the corpus were collected. We were thus able to create our own version of the corpus, using only the abstracts of the articles originally included in the categorization task of the TREC 2005 Genomic track.

Our TREC-05 corpus is therefore of a somewhat lesser quality than the corpus on which it is based, since using full-text articles are preferrable to using only abstracts.

4.1.2 Ohsumed

The Ohsumed corpus was originally created for a project at the Oregon Health Sciences University (Hersh et al., 1994). It is a subset of the Medline database, bibliographic database of important peer-reviewed literature maintained by the United States National Library of Medicine (NLM). Rather than full-text articles, Medline contains only the abstracts of the referenced articles. The total Ohsumed collection contains 348 566 references, collected from 270 journals covering the years 1987 to 1991.

The subset we have used for our experiments was prepared by Thorsten Joachims for testing an SVM model for text categorization (Joachims, 2002). It consists of all the abstracts in the first 20 000 articles from the year 1991 that have been indexed with a heading from the "diseases" category from the MeSH hierarchy.

Table 4.2: Categories of the Ohsumed corpus. These are originally subcategories of the *diseases* top-level heading of the MeSH hierarchy.

Category	MeSH heading
1	Bacterial Infections and Mycoses
2	Virus Diseases
3	Parasitic Diseases
4	Neoplasms
5	Musculoskeletal Diseases
6	Digestive System Diseases
7	Stomatognathic Diseases
8	Respiratory Tract Diseases
9	Otorhinolaryngologic Diseases
10	Nervous System Diseases
11	Eye Diseases
12	Urologic and Male Genital Diseases
13	Female Genital Diseases and Pregnancy Complications
14	Cardiovascular Diseases
15	Hemic and Lymphatic Diseases
16	Neonatal Diseases and Abnormalities
17	Skin and Connective Tissue Diseases
18	Nutritional and Metabolic Diseases
19	Endocrine Diseases
20	Immunologic Diseases
21	Disorders of Environmental Origin
22	Animal Diseases
23	Pathological Conditions, Signs and Symptoms

Medical Subject Headings (MeSH) is a controlled vocabulary developed by NLM.[3] Every journal article in Medline is indexed with several headings from MeSH, usually between 10 and 15 each. As of 2005, MeSH contained 22 568 subject headings. There are 16 top level categories, indexed by letters A through N, V and Z. Table 4.2 shows the 23 subcategories of the diseases category, C. These then have further subcategories. As an example consider heading C01: Bacterial Infections and Mycoses. This has the subcategories: Bacterial Infections (C01.252), Brain Abscess (C01.323), Central Nervous System Infections (C01.395), Infection (C01.539), Mycoses (C01.703) and Zoonoses (C01.908). And the hierarchy continues downwards: C01.252 has 12 subcategories, all but two of which in their turn have further subcategories.

The categorization task for our subset of the Ohsumed corpus is to assign each abstract to one of the 23 subcategories in table 4.2.

[3]Further details available at the MeSH homepage. URL:http://www.nlm.nih.gov/mesh/introduction2008.html

4.2 Processing

As we have seen in section 2.2, there are several ways of representing text. We have chosen to process our text corpora at the word level. As we shall return to, choosing the multi-word level would have had certain advantages. So far, though, experimental results for TC systems operating on the multi-word level have been discouraging (Sebastiani, 2002). Lewis (1992) argues that multi-word indexing has superior semantic qualities, but inferior statistical qualities, in that it results in "more terms, more synonymous or nearly synonymous terms, lower consistency of assignment (since synonymous terms are not assigned to the same documents), and lower document frequency for terms".

4.2.1 log-transformation

We presented the TF-IDF feature weighting in section 2.2.2; this is the one standardly used. However, Liau et al. (2003) introduce a novel weighting scheme, which tries to improve upon TF-IDF by applying a log transformation to the TF factor. The log TF-IDF weight is given by:

$$w_{ij}^{(\log)} = \log(f_{ij} + 0.5) \log \frac{|\mathcal{D}|}{D(t_i)} \tag{4.1}$$

The log transformation is introduced to amend unfavorable linearity. The intention is to assign more importance to differences in the lower end of the scale.

The results in Liau et al. (2003) indicate that this weighting scheme gives better results. We have tested both weighting schemes (2.1) and (4.1) on our corpora for comparison.

5

Results

We presented the BBR toolkit for (binary) Bayesian logistic regression in section 3.6; in this chapter we present the results from our experiments with testing this toolkit on the data sets described in the previous chapter. To examine the possibilities of Bayesian logistic regression, we do a thorough run-through of BBR's options and parameters. Such a study has to our knowledge not been made earlier—the authors of the software present some cursory results in Genkin et al. (2007), but provide little detail as to their parameter choices. As a benchmark, we also present results from the support vector machine toolkit SVM^{light} for comparison. We have found no other systematic comparison between the BLR and SVM methods in the literature on text categorization.

We will begin by discussing the choice of evaluation measure, and see how this relates to the first parameter we will consider, the threshold tuning.

5.1 Choice of evaluation measure and threshold tuning

We presented the different evaluation measures in section 2.5, and established the importance of having a single measure, rather than using both precision and recall. In experiments like ours, as we shall look into correlations and compare different parameter settings, a single numeric response is crucial.

Since precision and recall are the most common measures for citing results, the precision/recall breakeven point (PRBEP) might seem a natural choice. It is an intuitive measure, giving in a single number both the percentage of the documents belonging to the category c that are correctly classified as doing so (recall), and the percentage of the documents classified as belonging to c that actually do so (precision).

PRBEP has a drawback, however, in demanding a particular tuning of the *classification threshold*, that tends to be suboptimal. BBR outputs a probability value for each document belonging to the category in question, and then classifies all the documents where this probability is above a threshold θ as belonging to the category. As we shall return to, there are various available strategies for finding the value of θ, and these can be of some importance for the quality of the classifier. Since PRBEP is based on adjusting θ to the point where precision and recall are equal, it can come to choose thresholds that deteriorate the classifier's performance by most other measures. Since the choice of threshold tuning will form part of our analysis, PRBEP was not a viable option.

The measure F1, i.e. the F_β measure of equations (2.11) and (2.12) with $\beta = 1$, presents itself as a natural alternative. Short of precision and recall, it is the most common measure in TC, as well as in information retrieval, which it was developed for.

However, there are also evaluation measures that, unlike PRBEP and F1, are developed especially for text categorization, such as T11U and T13U (given in equations (2.14) and

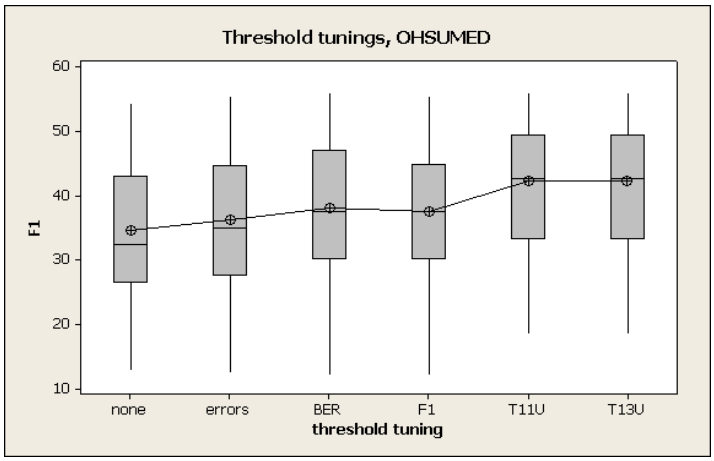


Figure 5.1: The performance on the OHSUMED corpus of BBR with a Laplace prior and 5 different threshold tunings: sum of errors ("errors"), balanced error rate ("BER"), F1, T11U and T13U. "None" indicates a threshold fixed at 0.5.

(2.15)). In being tailored for the classification task, they might be said to provide more suitable measures. Particularly T13U, with its large emphasis on "true positives", might give a better picture of the quality of a classifier for certain tasks. However, whereas F1 has a clear interpretation in being the harmonic mean of precision and recall, it is harder to see what a particular T11U or T13U score signifies. They are also far less widespread measures than F1. For these reasons we have considered F1 the best choice of performance measure, and will use it throughout this chapter.

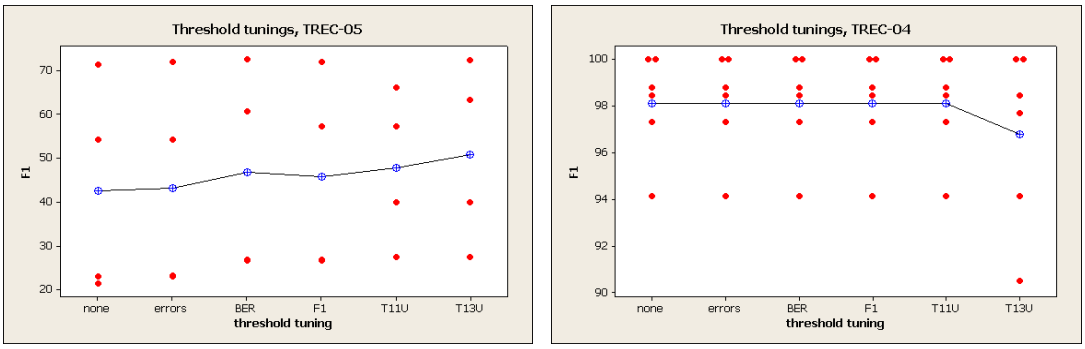
5.1.1 Threshold tuning

As established, the classification module of BBR produces a probability for each document belonging to the category considered, $P(c|\mathbf{d},\boldsymbol{\beta})$. The classification is then done by way of a threshold θ—all documents d for which $P(c|\mathbf{d},\boldsymbol{\beta}) > \theta$ are assigned to the category c.

We can choose to let θ be constantly equal 0.5, or let BBR find the θ that maximizes some function on the elements in the contingency table (table 2.1) for the training set. The latter option is what is called *threshold tuning*. The functions maximized are usually estimates of evaluation measures.

Since we use F1 as our only response in our experiments, we would expect the F1 tuning, which maximizes the estimate of F1, to outperform all other choices of tuning functions by a good margin. However, figure 5.1 tells a different story. We see that for the OHSUMED corpus, F1 is clearly outperformed by T11U and T13U. These two seem to perform about equally well.

Obviously, F1 tuning gives better results on the training set (since the threshold is chosen to maximize the F1 score on the training set), but, evidently, this does not always carry over to the test set. Figure 5.2 shows the results for the TREC corpora (since they have only 4 and 5 categories respectively, these are shown as individual value plots rather than as boxplots), we see that these are similar to those for OHSUMED, but with some differences. The advantage of T13U tuning is far smaller for TREC-05 than for OHSUMED, and T13U is actually the worst performer for TREC-04. This is probably due to T13U's strong emphasis on true positives



(a) **(b)**

Figure 5.2: The performance on the TREC-04 and TREC-05 corpora of BBR with a Laplace prior and 5 different threshold tunings: sum of errors ("errors"), balanced error rate ("BER"), F1, T11U and T13U. "None" indicates a threshold fixed at 0.5.

(cf. equation (2.15)); it is good at preempting trivial rejectors, which tend to be the problem for classifiers with low scores, but for classifiers with high scores, this one-sided emphasis makes T13U cause a loss in precision. We might say that T13U boosts recall at the expense of precision. Figure 5.3, which shows the F1 scores for each of the 23 categories of Ohsumed for both of these threshold tunings, further illustrates this effect: We see that T13U-tuning outperforms F1-tuning by an especially wide margin for the categories with low scores.

Since the performance of the T13U tuning is somewhat uneven, we have for the rest of the experiments, unless otherwise noted, used F1 tuning. Nonetheless, these results show that maximizing the chosen evaluation measure on the training set is not necessarily the best form of threshold tuning—maximizing some other function over the contingency table can yield better results.

5.2 The effects of the prior distribution

As explained in section 3.4 the prior distribution of the β parameters play a central role in Bayesian regression.

Figure 5.4 shows the differences between using the Laplace and Normal prior distributions for the three corpora. We see that the Laplace prior outperforms the Normal prior for all of them. Figure 5.4a shows the result for the Ohsumed corpus. This is where the difference is most pronounced—the Normal prior deteriorates the results substantially. In the boxplot 5.4b, for the TREC-05 corpus, there is a larger scattering of the data points, and we see that the Normal prior performs better than Laplace in some cases, but the trend is still evident. In figure 5.4c, for the TREC-04 corpus, the difference is harder to make out, and a large p value (0.87) tells us it is far from statistically significant; only on close scrutiny can we see that the results for the Laplace prior are still ever so slightly better.

It is worth noting both that there is a large difference in how successful BBR classification is for the three corpora—the results for TREC-04 are far better than those for TREC-05, which in their turn are much better than those for Ohsumed; and also that the positive effect of the Laplace prior is bigger the worse the results are. Thus, we might see this as an indication that for a hard classification task like Ohsumed, the sparseness in representation

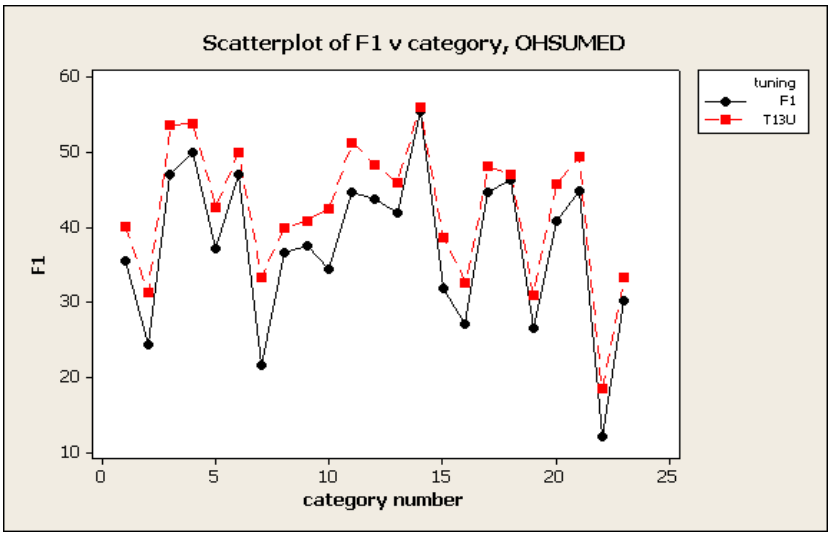
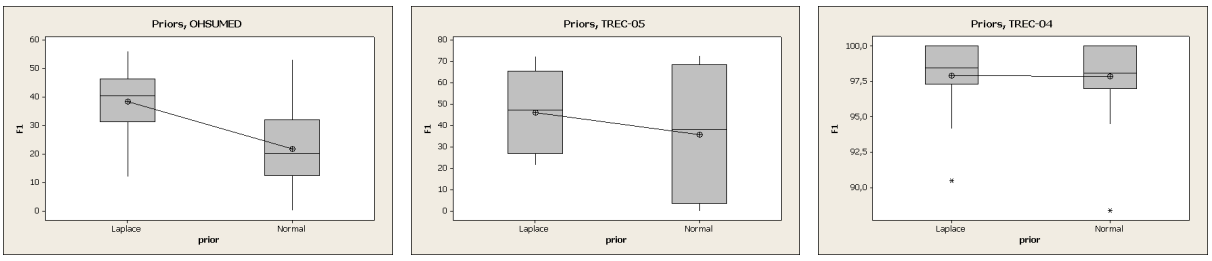


Figure 5.3: The F1 scores for each of the 23 categories of Ohsumed for the threshold tunings based on the F1 and T13U measures.



(a) **(b)** **(c)**

Figure 5.4: Boxplots from one-way ANOVA for F1 versus prior distribution in the GLM model, where the prior is either Normal or Laplace. We see that the Laplace prior gives substantially better results for two of the corpora.

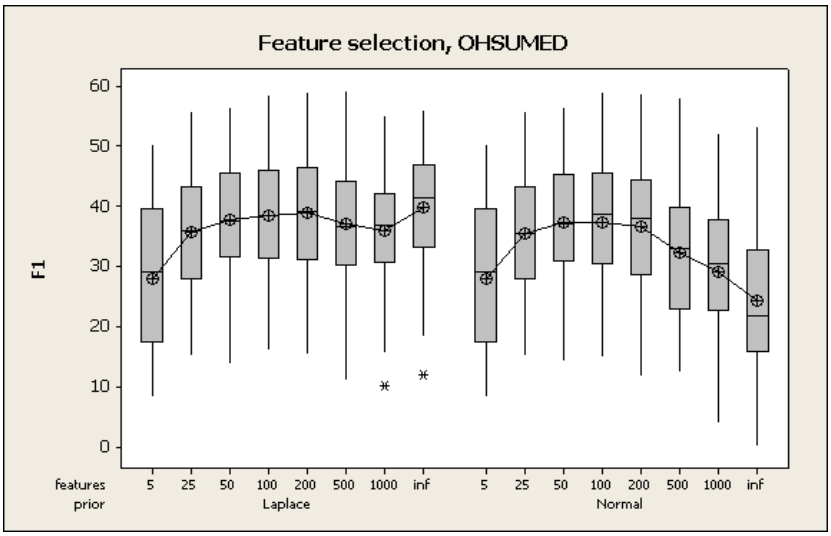


Figure 5.5: The effect of feature selection for the two different prior distributions and different values p of features selected. p = inf denotes that no feature selection is made.

given by the Laplace prior is of larger importance. As mentioned previously, this sparseness might be seen as an inherent feature selection in the Laplace prior. For the Ohsumed corpus, there are 30436 indexed words. This entails that a BLR model with a Normal prior, will try to estimate 30436 β values. The model we trained with a Laplace prior, on the other hand, typically included no more than 600 terms. Since this makes the former model prone to overfitting, a regression model with a Normal prior is usually accompanied by an additional feature selection (cf. Genkin et al. (2007)).

5.2.1 Feature selection and its criteria

Figure 5.5 shows the effect of feature selection for both priors on the Ohsumed corpus. We can see how the performance of the ridge regression model (with Normal prior) improves substantially as the number of features p selected increases, up until $p = 100$, and then deteriorates. Ridge regression has its worst performance when no feature selection is made (the rightmost boxplot); it performs better when modeling only 5 features, than when trying to take all the features into account. On the other hand, the lasso regression model (with Laplace prior) performs at its best without any feature selection.

The results are similar for the two other corpora, although, as was the case for the priors in figure 5.4, the trends are not as evident for the two TREC corpora as it is for Ohsumed.

In the tests shown in figure 5.5, we have employed the Pearson product moment correlation (cf. equation (2.4)) as our feature selection criterion. BBR offers a choice of four criteria, as described in section 5.2.1. The boxplots in figure 5.6, show how classification performance (F1) on the Ohsumed corpus varies depending on which feature selection criterion is used. We see that Pearson's product-moment correlation, the χ^2 test and bi-normal separation perform on par, whereas Yule's Q shows substantially inferior performance on this data set. The tests on the TREC corpora did not yield very different results, and we have used the Pearson correlation exclusively in the rest of our tests.

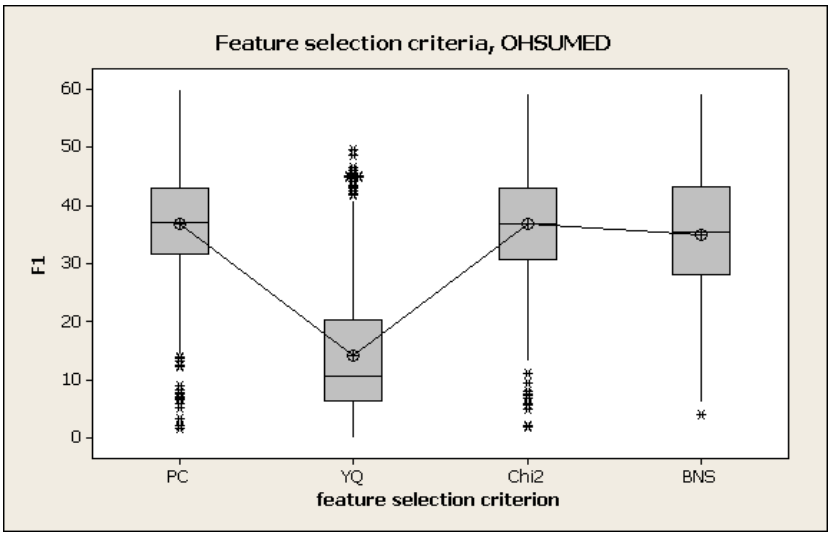
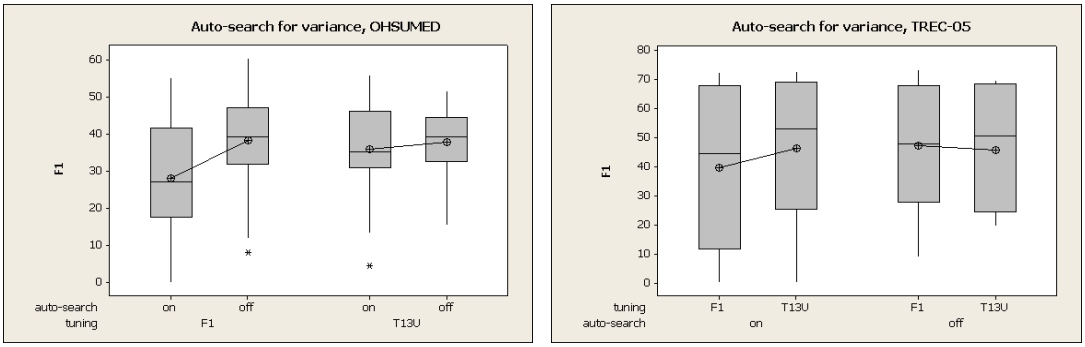


Figure 5.6: The boxplots show how classification performance (F1) on the Ohsumed corpus varies depending on which feature selection criterion is used—Pearson product-moment correlation (PC), Yule's Q (YQ), the χ^2 test (Chi2) and bi-normal separation (BNS).

5.2.2 Adjusting the prior variances

By default, BBR sets the prior variances of the β values to $\frac{|\mathcal{T}|}{m}$, where m is the mean 2-norm of the documents in the training set, and $|\mathcal{T}|$ is the number of distinct feature terms that occur in them.

BBR also offers another option for setting the prior variances, so-called *auto-tuning*. This is an attempt at automating the process of cross-validation, eliminating the need for the user to supply a grid of candidate values. Figure 5.7a shows the results of this auto-tuning of the variance for Ohsumed with threshold tunings F1 and T13U. We see that the improvement for the case of F1 tuning is substantial, and actually achieving better performance than T13U tuning, which had a clear edge without auto-tuning. When we look at the results for TREC-05 shown in figure 5.7b, we see that the situation is more or less the same for this corpus—except



(a) **(b)**

Figure 5.7: The effects of autotuning on the Ohsumed and TREC-05 corpora for two different threshold tunings.

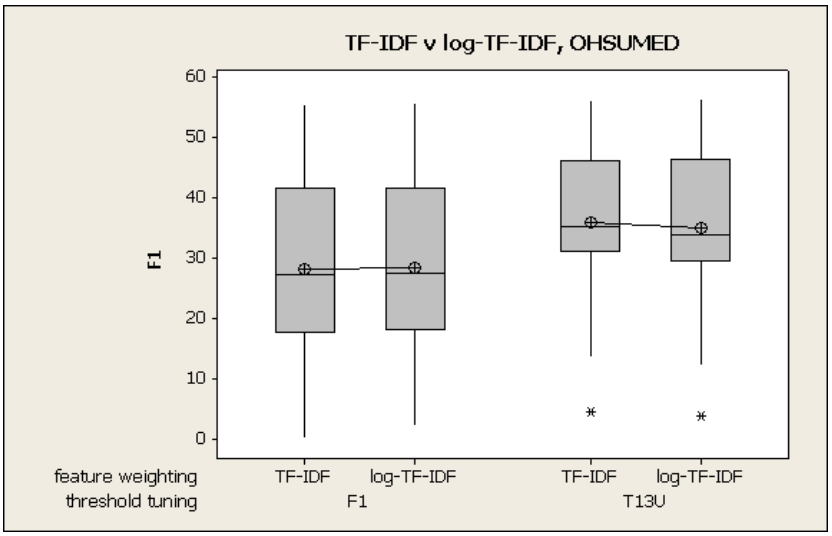


Figure 5.8: The effects of two different weighting schemes, TF-IDF and log TF-IDF.

for the auto-search causing a slight deterioration, rather than a small improvement, of the performance with T13U threshold tuning.

However, although auto-tuning does bring substantial improvements in performance, our experiments show that it is outperformed by a normal cross-validation procedure. We have employed 10-fold cross-validation over the candidate values 0.001, 0.01, 0.1, 1, 10, 100, 1000. The run-time of the cross-validation procedure is somewhat longer than that of the auto-search, so one might choose auto-search if the run-time is a critical factor, but the classifier trained with cross-validation performs better on all three of our corpora.

5.3 Log-transformed TF-IDF weighting

In section 4.2.1, we presented the log TF-IDF weighting scheme. log TF-IDF is an attempt at an improvement of the standard TF-IDF weighting scheme. The results of Liau et al. (2003) were promising, but the results for our corpora do not corroborate them. Figure 5.8 shows our results on the Ohsumed corpus for the two different weighting schemes. We see that while there is a small improvement when the classifier is tuned with F1, the classifier which is tuned with T13U and is the better of the two, performs less well when applied to data processed with the log TF-IDF weighting scheme. The results for the two TREC corpora are equally inconclusive—our results do not provide ample ground for any conclusion on whether the log TF-IDF scheme can be advantageous.

5.4 2^k experiment

To investigate the effects and interactions further, we have conducted a 2^k factorial experiment on the TREC-05 corpus, using a selection of the parameters as factors. We here outline the design of the experiment and present its results.

5.4.1 Design

We have included $k = 8$ factors and used a full factorial design to be able to investigate all second-order correlations as well as the primary effects. Table 5.1 shows the factors used.

2^k experiments require the factors to have 2 levels each—a high level, and a low level.

The factors prior distribution, feature weighting and cosine normalization have only two possible levels, whereas for the five others, we have had to make a choice of what the high and low levels should be.

We have not previously discussed *cosine normalization*. This is a method that is commonly used in the vector model of information retrieval (cf. Baeza-Yates and Ribeiro-Neto (1999)), and involves normalizing the feature weights w_{ij} so that the resulting document vector $\mathbf{d}^{(\text{norm})}$ has length 1. This procedure takes its name from the fact that the vector product of two vectors thus normalized, equals the cosine of the angle between them.

Of the six possible values for *threshold tuning*, we decided to include T13U and F1. T13U is the high level—can be viewed as a form of accentuation of true positives.

As we saw in section 5.2.2, *cross-validation* of the prior variances outperformed the autosearch algorithm. Since these two options are mutually exclusive, we have only included cross-validation in the factorial experiment. The crossvalidation is 10-fold over the following geometric progression centered on 1: (0.001, 0.01, 0.1, 1, 10, 100, 1000).

For *feature selection*, we chose as our high level 100 features selected by the Pearson product moment correlation criterion, which is the combination that yielded the best results in section 5.2.1.

The default *convergence threshold* of BBR is $5 \cdot 10^{-4}$. We chose $5 \cdot 10^{-5}$ as the high level for this factor.

A related parameter is the *maximum number of iterations*. This gives the maximum number of passes allowed over the parameter vector β during fitting. The default level is 1000. We chose 10000 as the high level.

These 8 factors run on the 4 categories of TREC-05, gives a total of 1024 experiments.

5.4.2 Results

As can be seen from figure 5.9, there are only four effects that are significant at the 5% significance level, three of which are second-order effects. Thus, feature selection (H) is the only primary factor that is proven significant at this level. We notice that the effect for this factor, unsurprisingly, is positive.

It is slightly less intuitive that the three significant second-order effects are all negative. These three interactions, AD, AH and DH, are all combinations of factors that by themselves have positive effects: Laplace prior (A), cross-validation of variance (D) and feature selection (H). As can be seen from the interaction plot in figure 5.10, it is not so much the case that these combinations have a negative effect; the situation is rather that the positive effect of both of the factors in question together, is less than the sum of the positive effects they have separately. We see that feature selection improves the performance slightly when the Laplace prior is on the high level as well as when cross-validation is on high level, but in both cases it improves performance substantially less than when the other factor is on the low level. The Laplace prior and cross-validation actually cause a slight reduction in performance when combined with one of the other two factors at high level, but in both cases this negative effect is minimal compared to the positive effect of having both of the factors at the high level

Table 5.1: Factors for 2^k experiment.

	Factor	High level	Low level
A	Prior distribution	Laplace	Normal
B	Threshold tuning	T13U	F1
C	Feature weighting	log TF-IDF	TF-IDF
D	Cross-validated variance	yes	no
E	Max. no. of iterations	10000	1000
F	Convergence threshold	$5 \cdot 10^{-5}$	$5 \cdot 10^{-4}$
G	Cosine normalization	yes	no
H	Feature selection	yes	no

compared to having both at the low level.

Figure 5.10 also indicates that many effects are close to being significant, but without breaking the 95% level. Table 5.2 shows the *p*-values of all effects of the first and second orders. We notice that while the positive effect of the Laplace prior is not significant to the 95% level, probably due to the Normal prior's good performance when used with feature selection, it is significant at the 90% level. At this level, the interactions between threshold tuning on the one hand, and cross-validation and

The only positive effect of the second order which is not completely negligible, is that between cross-validation and cosine normalization.

5.5 Results for the extensions to BBR

5.5.1 Informative priors

We presented the idea of tuning the priors of individual features based on domain knowledge in section 3.7.1. In this section we describe our implementation of this idea, and present the results it yielded.

We have attempted to create informative prior distributions for the indexed words of the Ohsumed corpus, using the MeSH hierarchy (described in section 4.1.2) as our source for the knowledge words—i.e. the words we wish to assign greater importance to, by way of their prior distributions.

It should be noted that the documents in our version of the Ohsumed corpus do not contain the MeSH terms assigned to them. The MeSH hierarchy nevertheless seemed a suitable source for the knowledge words (KWs). The medical subject headings in MeSH are used to index health-related documents to facilitate searching, and can therefore be expected to be information-bearing. Their occurence in a document can thus potentially be a good predictor

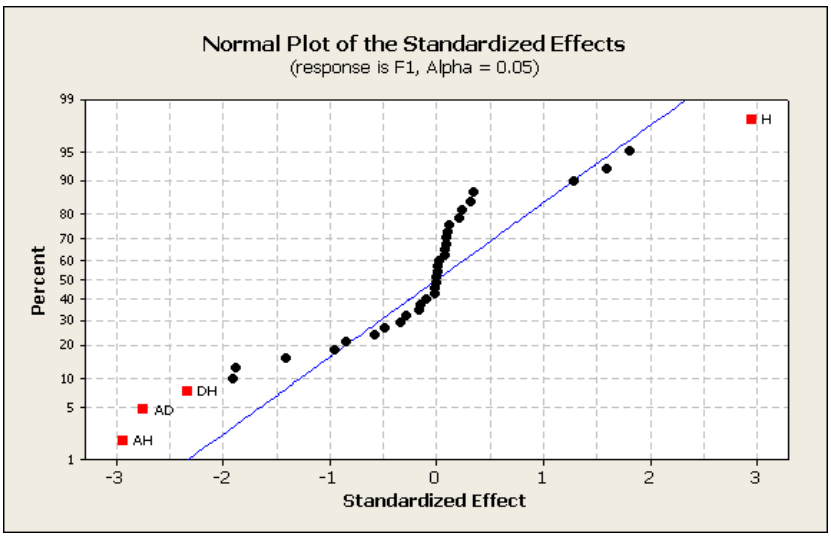


Figure 5.9: Effects plot of the 2^k experiment.

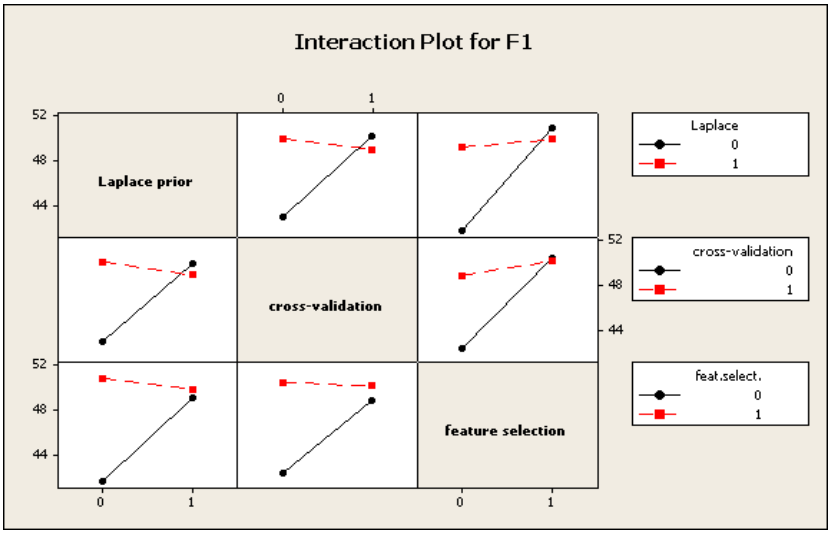


Figure 5.10: Interaction plot for the three significant second-order effects in the 2^k experiment on the TREC-05 corpus.

Table 5.2: Results from the 2^k experiment. Statistically significant effects at the 90% significance level, are marked with an asterisk. Those significant at the 95% level are marked with two asterisks.

Term	Effect	Coef	SE Coef	T-value	p-value	
Constant		47.939	1.2036	39.83	0.000	
prior	2.875	1.438	0.7997	1.80	0.073	*
threshold	-1.362	-0.681	0.8021	-0.85	0.396	
log-tf-idf	0.198	0.099	0.9316	0.11	0.915	
cross-val	2.974	1.487	0.9316	1.60	0.111	
iterations	-0.042	-0.021	1.1889	-0.02	0.986	
convergence	-0.248	-0.124	1.2036	-0.10	0.918	
cosine	0.558	0.279	0.8021	0.35	0.728	
feats	4.711	2.356	0.8014	2.94	0.003	**
prior*threshold	-1.386	-0.693	0.7261	-0.95	0.340	
prior*log-tf-idf	0.345	0.172	0.7402	0.23	0.816	
prior*cross-val	-4.067	-2.033	0.7402	-2.75	0.006	**
prior*iterations	0.111	0.055	0.7912	0.07	0.944	
prior*convergence	0.335	0.167	0.7997	0.21	0.834	
prior*cosine	-0.846	-0.423	0.7261	-0.58	0.560	
prior*feats	-4.264	-2.132	0.7260	-2.94	0.003	**
threshold*log-tf-idf	-0.718	-0.359	0.7417	-0.48	0.629	
threshold*cross-val	-2.783	-1.392	0.7417	-1.88	0.061	*
threshold*iterations	0.005	0.002	0.7912	0.00	0.998	
threshold*convergence	-0.259	-0.129	0.8021	-0.16	0.872	
threshold*cosine	-0.414	-0.207	0.7274	-0.28	0.776	
threshold*feats	-2.781	-1.391	0.7270	-1.91	0.056	*
log-tf-idf*cross-val	0.484	0.242	0.7575	0.32	0.749	
log-tf-idf*iterations	-0.000	-0.000	0.9202	-0.00	1.000	
log-tf-idf*convergence	0.007	0.003	0.9316	0.00	0.997	
log-tf-idf*cosine	-0.501	-0.250	0.7417	-0.34	0.736	
log-tf-idf*feats	-0.219	-0.109	0.7413	-0.15	0.883	
cross-val*iterations	0.000	0.000	0.9202	0.00	1.000	
cross-val*convergence	0.030	0.015	0.9316	0.02	0.987	
cross-val*cosine	1.896	0.948	0.7417	1.28	0.202	
cross-val*feats	-3.466	-1.733	0.7413	-2.34	0.020	**
iterations*convergence	-0.042	-0.021	1.1889	-0.02	0.986	
iterations*cosine	0.127	0.063	0.7912	0.08	0.936	
iterations*feats	0.143	0.072	0.7912	0.09	0.928	
convergence*cosine	0.195	0.097	0.8021	0.12	0.903	
convergence*feats	0.134	0.067	0.8014	0.08	0.933	
cosine*feats	-2.057	-1.029	0.7270	-1.41	0.158	

of its category.

Figure 5.12 shows the top 18 lines of the text file defining the MeSH hierarchy.[1] This excerpt contains the mesh terms for 15 different body regions, 13 of them are the subterms of "abdomen", the remaining 2 are the subterms of "back".

We see that many of the headings consist of two words. There are also many examples of headings containing three or more words, e.g. "Eye Infections, Bacterial". Since our document representation is based on indexing single words rather than multi-word phrases (cf. section 2.2), we use each word of such a heading as a separate KW.

We have chosen to increase the variances in the priors of the KWs rather than their modes. As mentioned in section 3.7.1, adjustment of the modes would require selecting separate KWs for each category. As our corpus consists of 23 different categories, adjusting the variances seemed a better option. Dayanik et al. (2006) also report slightly better results for increasing the variance than for increasing the mode.

We extracted four different sets of KWs, and tested these separately. The methods for extracting the words to be included in these four sets, were of differing complexity. The first set consists of all the headings in the MeSH hierarchy converted to unigrams (i.e. single words). We refer to this set as `All`. The second set, `C-cat`, is extracted from just the MeSH headings belonging to the same subtree as the OHSUMED documents (i.e. the *C* category, cf. table 4.2).

For the last two sets of KWs, we have attempted to incorporate words not only from the MeSH headings themselves, but also from their descriptions. Figure 5.13 shows an entry from the text file describing the MeSH hierarchy.[2] We see that each entry has many fields. The field named MH gives the subject heading itself, and MN gives its tree number, which tells us its place in the hierarchy of headings. We see from figure 5.13 that a heading can have more than one tree number. The field MS provides a description of what the subject heading designates. The third of our sets of KWs, `C+`, extracts the words from the fields FX, HN, MH, MN and MS from all the entries in category *C*. The fourth KW set, `C*`, uses all the same fields, in addition to PI, ENTRY and PRINT ENTRY.

The variances of the KWs given by one of the four sets, is set equal to the prior variance of the other words multiplied by 100, as suggested by Dayanik et al. (2006).

Figure 5.11 which shows our results, indicates that these efforts were not very successful. While the KW priors do not deteriorate performance significantly in these experiments, they certainly do not improve it. It is worth noting that the results get worse as the set of priors gets more advanced. We defer the further discussion of these somewhat disappointing results to chapter 6.

5.5.2 EM algorithm

We tested the application of the EM algorithm described in section 3.7.2, on an extended version of our OHSUMED corpus. Our distribution of OHSUMED contains 20000 categorized documents; for the EM algorithm we used an additional 30000 uncategorized documents. This proved insufficient. The EM algorithm converged after just a couple of iterations, and there was no noticable change in the performance of the classifier. We take this as an indication that the uncategorized portion of the corpus needs to be substantially larger than the categorized part, to have the desired effect.

[1] Available at ftp://nlmpubs.nlm.nih.gov/online/mesh/.meshtrees/mtrees2008.bin
[2] Available at ftp://nlmpubs.nlm.nih.gov/online/mesh/.asciimesh/d2008.bin

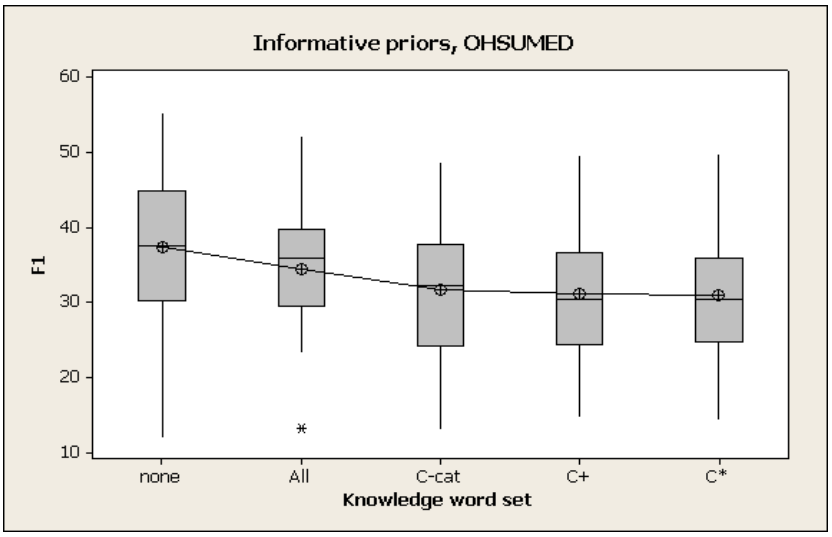


Figure 5.11: F1 scores for categorization of Ohsumed, using four different sets of knowledge words that are assigned higher prior variance.

```
Body Regions; A01
Abdomen; A01.047
Abdominal Cavity; A01.047.025
Peritoneum; A01.047.025.600
Douglas' Pouch; A01.047.025.600.225
Mesentery; A01.047.025.600.451
Mesocolon; A01.047.025.600.451.535
Omentum; A01.047.025.600.573
Peritoneal Cavity; A01.047.025.600.678
Peritoneal Stomata; A01.047.025.600.700
Retroperitoneal Space; A01.047.025.750
Abdominal Wall; A01.047.050
Groin; A01.047.365
Inguinal Canal; A01.047.412
Umbilicus; A01.047.849
Back; A01.176
Lumbosacral Region; A01.176.519
Sacrococcygeal Region; A01.176.780
```

Figure 5.12: Sample headings from the MeSH hierarchy. These are all the subheadings for the body regions "abdomen" and "back".

```
*NEWRECORD
RECTYPE = D
MH = Botulinum Toxins
AQ = AD AE AG AI AN BI BL CF CH CL CS CT DU EC GE HI IM IP ME PD PH
PK PO RE SD SE ST TO TU UR
PRINT ENTRY = Botulin|T121|T131|NON|EQV|UNK (19XX)|770414|abbcdef
PRINT ENTRY = Clostridium botulinum Toxins|T121|T131|NON|EQV|UNK
(19XX)|850617|abbcdef
ENTRY = Botulinum Toxin|T121|T131|NON|EQV|UNK (19XX)|770421|abbcdef
ENTRY = Toxin, Botulinum
ENTRY = Toxins, Botulinum
ENTRY = Toxins, Clostridium botulinum
ENTRY = botulinum Toxins, Clostridium
MN = D12.776.097.156
MN = D23.946.123.179
PA = Anti-Dyskinesia Agents
PA = Poisons
FX = Botulism
MH_TH = NLM (1969)
ST = T121
ST = T131
RN = 0
AN = /antag permitted but consider also BOTULINUM ANTITOXIN
PI = Clostridium Botulinum (1966-1968)
PI = Toxins (1966-1968)
MS = Proteins synthesized as a single chain of ~150 kDa with 35%
sequence identity to TETANUS TOXIN that is cleaved to a light and a
heavy chain that are linked by a single disulfide bond. They have
neuro-, entero-, and hemotoxic properties, are immunogenic, and
include the most potent poisons known. The most commonly used
apparently blocks release of ACETYLCHOLINE at cholinergic SYNAPSES.
OL = use BOTULINUM TOXINS to search BOTULINUM TOXIN 1969-77 (as Prov
1969-71)
PM = 78; was BOTULINUM TOXIN 1972-77
HN = 78; was BOTULINUM TOXIN 1969-77 (Prov 1969-71)
MED = *329
MED = 471
M90 = *467
M90 = 576
M85 = *398
M85 = 508
M80 = *162
M80 = 260
M75 = *158
M75 = 253
M66 = *87
M66 = 209
M94 = *559
M94 = 726
MR = 20050630
DA = 19990101
DC = 1
DX = 19720101
UI = D001905
```

Figure 5.13: Sample description entry for a single heading from the MeSH hierarchy.

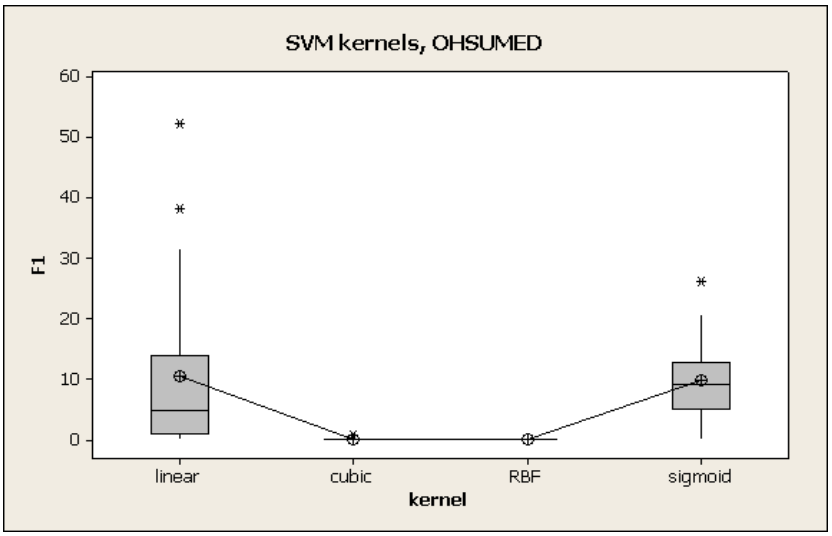


Figure 5.14:

5.6 Support vector machines by SVM^{light}

Our tests with support vector machines have been done with the SVM^{light} toolkit, developed by Thorsten Joachims.[3] It is common in the TC literature when using this toolkit as a benchmark, to simply employ a linear kernel and default settings. As the results were unsatisfactory, we attempted to adjust some settings, but limited ourselves to trying different kernels and different choices for the so-called trade-off parameter—as recommended by Joachims (2002).

As we have seen in section 2.7.4, the kernels are the central component of SVM's ability to handle data that are not linearly separable. SVM^{light} gives 4 choices of kernel. There is the standard linear kernel; this equals the vector product:

$$K_{\text{lin}}(\mathbf{d}_i, \mathbf{d}_j) = \mathbf{d}_i \cdot \mathbf{d}_j$$

Then there is the polynomial kernel

$$K_{\text{poly}}(\mathbf{d}_i, \mathbf{d}_j) = (u\mathbf{d}_i \cdot \mathbf{d}_j + 1)^v$$

where u and v are adjustable parameters. The radial basis function (RBF) kernel is given by:

$$K_{\text{RBF}}(\mathbf{d}_i, \mathbf{d}_j) = \mathrm{e}^{-\gamma\|\mathbf{d}_i - \mathbf{d}_j\|^2}$$

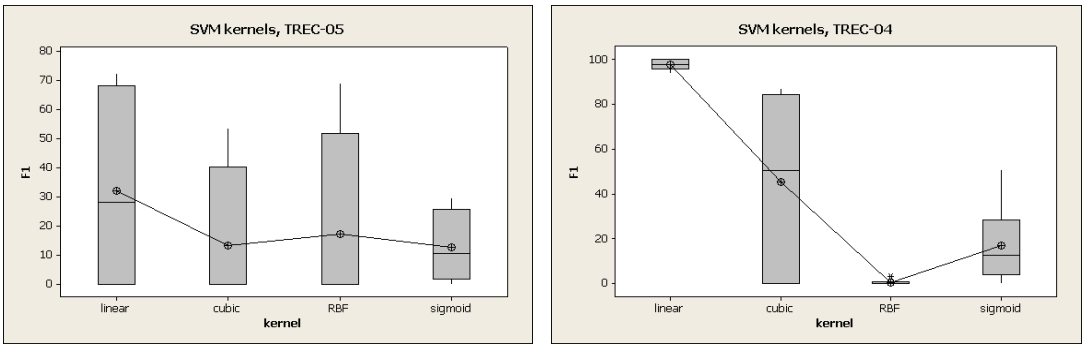
where γ is an adjustable parameter; the Gaussian radial basis function (given in (2.22)) is a variant of this. The sigmoid kernel is given by:

$$K_{\text{sigm}}(\mathbf{d}_i, \mathbf{d}_j) = \tanh(u(\mathbf{d}_i \cdot \mathbf{d}_j) + 1)$$

and has been quite popular for SVMs due to its origin from neural networks (Lin and Lin, 2003).

Figure 5.14 shows the F1 scores SVM^{light} achieves on the Ohsumed corpus with these four kernels and otherwise default settings. We see that the sigmoid kernel is the only one,

[3] Available at http://svmlight.joachims.org/



(a) **(b)**

Figure 5.15:

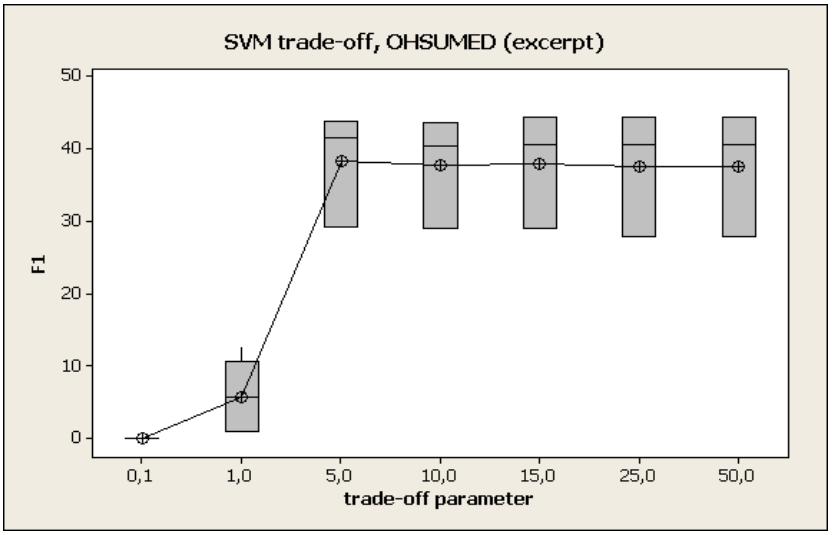


Figure 5.16: trade-off

apart from the linear kernel, that has a potential to improve the results for this corpus. For the other two corpora (cf. figure 5.15) none of the non-linear kernels seem to have any positive effect.

The *trade-off parameter* τ controls the trade-off between the training error (the number of errors the trained model makes on the training set) and the margin (the distance from the support vectors to the separating hyperplane, cf. figure 2.4). The lower the value of τ, the more training error is tolerated.

Figure 5.16 shows results for a linear SVM on the first 5 categories of the OHSUMED corpus, trained with different values of τ. Whereas decreasing τ to 0.1 deteriorates performance, we see that increasing τ to 5 gives a substantial improvement. Increasing τ further seems to give no positive effect, however.

As a sigmoid kernel showed some promise for the OHSUMED corpus, it is natural to consider a two-way analysis of variance for this factor along with the trade-off parameter τ. Figure 5.17 shows the resulting contrast plot. We see that a τ value of 5 does not improve the performance of the sigmoid classifier substantially,unlike what is the case for the

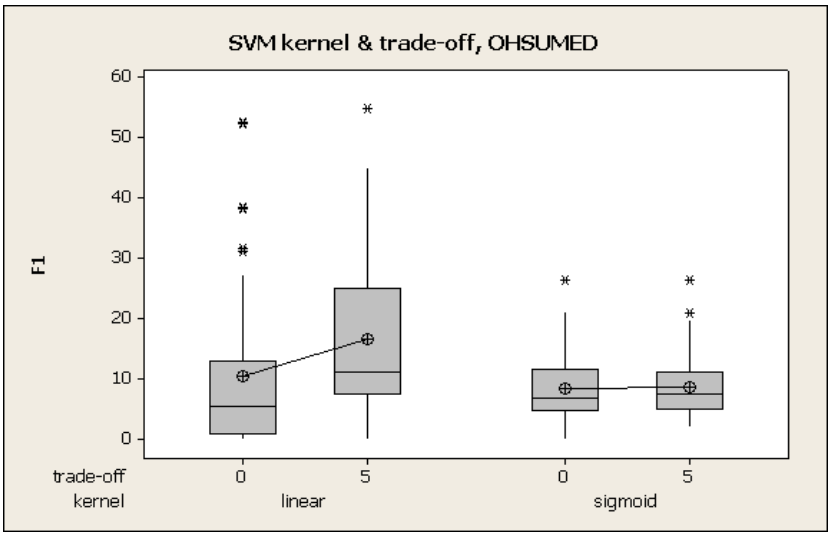
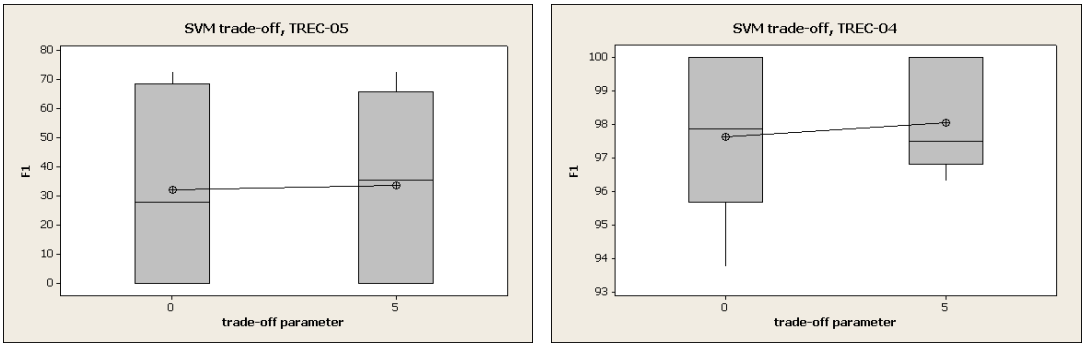


Figure 5.17:



(a) **(b)**

Figure 5.18:

linear classifier.Thus, we see that of the possible combinations of the two parameters we have considered, the combination of a linear kernel and a trade-off parameter $\tau = 5$ yields the best performance.

Figure 5.18 shows the results of increasing the τ value for the TREC corpora. We note that although the effects are minor, $\tau = 5$ results in better performance in both cases. As we have seen, the other 4 kernels can deteriorate the classifier's performance on these corpora; our tests show that this is also the case when $\tau = 5$.

This entails that the optimal parameter choice is the same for all three corpora.

5.7 Summary of results

Tables 5.3 and 5.4 summarize the best results obtained on our three text corpora by BBR and SVM^{light}, respectively.

Table 5.3: F1 scores for our three text corpora obtained by BBR with the following settings: Laplace prior; cosine normalization; prior variances determined by cross-validation; log TF-IDF feature weighting scheme and F1 threshold tuning.

Corpus	N	Mean	StDev	Min	Q1	Median	Q3	Max
OHSUMED	23	43.58	9.57	20.80	37.40	43.91	51.09	61.23
TREC-05	4	54.86	15.21	39.42	40.67	54.87	69.04	70.29
TREC-04	6	98.11	2.21	94.12	96.50	98.63	100.00	100.00

Table 5.4: F1 scores for our three text corpora obtained by SVM^{light} with the following settings: Linear kernel; trade-off parameter $\tau = 5$; log TF-IDF feature weighting scheme.

Corpus	N	Mean	StDev	Min	Q1	Median	Q3	Max
OHSUMED	23	16.45	12.79	0.00	7.34	11.15	24.99	54.71
TREC-05	4	33.40	31.90	0.00	0.00	35.20	65.70	72.80
TREC-04	6	98.05	1.58	96.30	96.80	97.50	100.00	100.00

6

Discussion

In the previous chapter, we presented the results of our experiments; here, we discuss some aspects of them more in depth.

6.1 The three corpora

Our three data sets of previously categorized documents are different in many respects, and we have seen how the BLR and SVM classifiers yield very different results for the three of them (cf. tables 5.3 and 5.4).

The Ohsumed corpus has shown itself to be the hardest corpus to categorize for the classifiers—the F1 scores for Ohsumed are consistently lower than those for TREC-04 and TREC-05. This is probably in part due to Ohsumed's many categories. The 23 categories both give a ample opportunity for error, and entails that many of the categories are bound to be very similar.

TREC-04 has gotten very good results overall. We hesitate as to what to make of this, however. TREC-04 is something of a makeshift corpus, as several ad hoc choices were made in its preparation. Due to the modest amount of data, we considered it most important to get as large a training set as possible, and used 90% of the documents to this end. It appears, however, that the remaining 10% may have been to small a test corpus. That we achieved 100% scores in several cases might be an indication of this.

TREC-05 has achieved scores in between the other two—substantially better that Ohsumed's, but far from the level of TREC-04. The results from TREC-05 might be the most indicative of the performance of the classifiers in a practical TC task, where we are unlikely to try to categorize documents automatically into as many as 23 categories in a flat structure.

6.2 Small effects and statistical significance

As we saw, particularly in our 2^k experiment (cf. table 5.2), the effects of many of the different parameter settings were nowhere near making the 90% percent level of significance, far less the 95% one. Cosine normalization can be taken as an example, with a p-value of 0.728. Can we conclude from our results that cosine normalization is ineffectual to a classifier's performance?

It is worth noting here, that the TREC-05 corpus has only four categories, and thus four measurements for each combination of parameters. We chose to use this corpus rather than Ohsumed, for this reason. For Ohsumed a full factorial experiment like the one we performed, would take a week or more, even on a fairly powerful server. The total number of response values would be $2^8 \cdot 23 = 5888$.However, the Ohsumed corpus would have given us more reliable results.

It is also worth mentioning, that in this case, unlike many other settings where 2^k experiments are used, there is no measurement error as such. The experiment would, if repeated with the same settings, give precisely the same results. This might be seen as an argument for considering effects of lower p-values than one would in an experimental setting with physical measurements.

We would claim that wee cannot conclude that the effect of a parameter is negligible merely because it is not significant on the 95% level. This is part of the reason why we in chapter 5, considered boxplots more than p-values and other statistics—boxplots can often give a clearer picture of whether the effect of a parameter is substantial enough to be subjected to further testing. That said, the 95% level is probably a good benchmark for concluding that one method is superior to another, and it could with advantage have been used more often in machine learning research.

6.3 Processing and log-transformation

The log TF-IDF feature weighting scheme gave some gains in performance, but not to an extent that was statistically significant. As mentioned in section 4.2.1, the rationale behind log TF-IDF is to remove unfavorable linearity by way of the log-transformation of the term frequency. Recall that the standard TF-IDF weight is given by $w_{ij} = f_{ij} \log \frac{|\mathcal{D}|}{D(t_i)}$, and that log TF-IDF alters this formula to $w_{ij}^{(\log)} = \log(f_{ij} + 0.5) \log \frac{|\mathcal{D}|}{D(t_i)}$. The log-transformation has the advantage of making the differences at the lower end of the scale more pronounced. This is desirable—we want the difference between 2 and 3 occurences of a certain word in a document, to be given more weight than the difference between 19 and 20 occurences (or 20 and 30 occurences, for that matter). However, log TF-IDF sometimes deteriorated performance. What disadvantages could it have? It might be at a disadvantage, however, in that it assigns only non-zero feature weights, thus sacrificing a certain sparsity. But this is taken care of, since a zero frequency results in a negative log TF-IDF value. In our processing we have removed these. Must be that in some cases the log transformation emphasis on the lower end, creates the wrong decisions. On average, however, it performs better, albeit not at a significant level.

6.4 Extensions to BBR

The results for our proposed extensions to BBR were disappointing. It must be stressed, however, that our experiments with these extensions were merely preliminary—the limited time available did not permit us to exhaust their options. We will here present some thoughts on why the extensions did not work well in our implementation of them.

6.4.1 Informative priors

For the informative priors extension, we chose to increase the prior variance of the knowledge words rather than their prior mode. As established in section 3.7.1, an advantage of adjusting the variance is that we allow the knowledge words to be either positive or negative predictors of each category. A drawback is that they are more probable to be anything in between. It is not unlikely that a lot of the words we choose as knowledge words are unlikely to be neutral in this respect—thus, we would want to specify that their presence in a document is

a clear indication of whether or not it belongs to a category. But, to avoid having to specify knowledge words specific to each category, we would not want to specify whether they be positive or negative predictors.

Increasing their variances was the closest we could come to achieving this. An increased variance should make it easier for the training algorithm to assign strongly positive or strongly negative β values to the knowledge words. Our results indicate, however, that this did not function optimally.

Our choice of knowledge words might also have been suboptimal, particularly combined with the choice of adjusting the variances. The fact that the results deteriorated for our more refined methods of extracting them, might indicate that choosing knowledge words that are central to the description of a category (as we might assume the descriptions for the MeSH headings that we used are) do not work well with increasing the variance. For knowledge words of this kind, one might need to adjust their modes separately for each category—assigning a positive mode to a word for the categories of which the word is a descriptor term, and a negative mode for the rest. This needs to be verified by further testing.

Dette er viktig å skrive ut, nevner i konklusjon.

Another aspect of the work with informative priors, is that our choice of processing the corpora at the word level, did not allow us to take phrases into account. We saw in section 5.5.1 that many of the MeSH headings we tried to use as our knowledge words, consisted of more than a single word. This is an unfortunate limitation of "bag-of-words" processing.

6.4.2 EM algorithm

Our attempts at boosting prediction accuracy with the help of uncategorized documents by using the EM algorithm, proved ineffectual. It is hard to say exactly what rendered this method unable to affect the classifier's performance. Wu (2006) employed this method for far larger sets of uncategorized data. Having gained access to an augmentation of the Ohsumed corpus doubling its size, we initially thought that this would be large enough for the EM algorithm to have a significant effect, but we were proved wrong. Ours are the first experiments we are aware of that attempts to employ this method to BLR. These initial tests give no clear indication of its effectiveness, but the potential in using uncategorized data is so large, that further testing, preferrably on a larger set of uncategorized data, is warranted.

6.5 Comparison to SVM

We have seen that in our tests, Bayesian logistic regression with the BBR toolkit, outperforms support vector machines with the SVM^{light} toolkit. It should be stressed that since BLR has been our main focus in this thesis, our experiments have involved far more extensive parameter tuning for BBR than what was the case for SVM^{light}. However, the differences in performance are large enough to be seen as an indication that BBR might indeed produce the better classifiers of the two toolkits.

7

Concluding remarks

In this thesis, we have given an overview of the current state of text categorization research. The most widespread methods have been surveyed, and what are arguably the two most promising method of today, support vector machines and Bayesian logistic regression, have been presented in detail.

Our extensive testing of Bayesian logistic regression with the BBR toolkit, showed that adjusting the prior variances was one of the most consequential parameter tunings. Cross-validation of the variances gave better results than BBR's self-tuning. We also saw that threshold tuning had a significant effect. Using the F1 estimate as a tuning function gave good results, but its being outperformed by T13U in certain situations give reason to believe that other tuning functions might occasionally be better suited. The Laplace prior distribution showed far better results than the Normal prior, but this advantage was lessened when the Normal prior was supplemented by feature selection.

We investigated two individual extensions to BBR: incorporating domain knowledge by way of the prior probability distributions of single words; and making use of uncategorized documents to boost learning accuracy. For making use of uncategorized documents to augment the training set, our results indicate that the amount of uncategorized data needs to be substantially larger than the original training set, to be able to affect the performance.

For incorporating domain knowledge in priors, we saw that a document representation on the word level could be a serious limitation, as many of the terms we would like to detect occur in phrases of two or three words. We also saw that increasing the variance rather than the mode of highly specific words, did not yield satisfactory results, indicating that increasing the mode might be a more viable option.

Our tests of the SVM^{light} toolkit for support vector machines show that the linear kernel performed better on our data sets than the more advanced kernels, and that lowering the toleration for training errors by increasing the trade-off parameter τ improved the classifier's performance. Comparing our results for BBR and SVM^{light}, we see that BBR performs substantially better. Although it should be taken into account that the parameters of BBR have been more carefully tuned, this is a clear indication that the performance of BBR is at least a par with that of SVM^{light}, and perhaps superior to it.

Surprisingly, we found that using the log TF-IDF feature weighting scheme in processing the corpora, as proposed by **?**, had only minor impact on the classifier's performance compared to using the standard TF-IDF weighting. This was the case for both BBR and SVM^{light}.

7.1 Future work

The most evident avenue for future work, would be to pursue our proposed extensions to BBR further. We have already discussed how this could be done in section 6.4.

We will here present two other ideas that invite further exploration. Adaptive threshold tuning springs out of the observation that F1 tuning does not always give the best results (cf. section 5.1.1). The elastic net, on the other hand, is inspired by the fact that both the Normal prior and the Laplace prior seem to have certain desirable qualities, giving some appeal to the idea of combining the best of the two distribution.

7.1.1 Adaptive threshold tuning

We saw in section 5.1.1 that F1 threshold tuning was outperformed by T13U tuning for the Ohsumed corpus, where results were mediocre , whereas F1 performed far better on the TREC-04 corpus, where results were very good. This gives reason to believe that a method for *adaptive threshold tuning* might be worth pursuing. By "adaptive" we mean that the tuning function takes the performance of the classifier into account, emphasising precision when results are relatively good (as with TREC-04), but giving higher priority to recall when results are mediocre or poor.

The F_β measures as defined in section 2.5.4 provide one option for doing this. The F1 measure, equals the F_β measure with $\beta = 1$, and gives equal weight to precision and recall. Higher values for the parameter β (not to be confused with the βs of our regression model) give higher priority to recall at the expense of precision—thus, making it more like the T13U measure. If we continue to use F1 as our evaluation measure, we could use an F_β measure as our tuning function, where the value of β was a function of the F1 score for a threshold $\theta = 0.5$. The precise relationship between the F1 score and the β value would have to be determined by trial and error.

7.1.2 The elastic net

Our 2^k experiment (cf. section 5.4) indicates that there is not a large difference in performance between lasso regression and ridge regression. This can be seen as an indication that both the Laplace and the Normal prior distributions have desirable attributes. As we saw in section 3.4, the Laplace prior has an advantage in creating a sparser model with fewer non-zero β parameters, whereas the Normal prior improves the results by reducing the values of all parameters, and thus reducing the number of βs that have an impact on the classifier. The *elastic net* regression method (Zou and Hastie, 2005) tries to capture the best of both worlds, by combining elements from both priors.

Zou and Hastie (2005) show that given an ordinary least squares solution to the regression problem, $\hat{\beta}^{(\text{OLS})}$ say, the solution of ridge regression with parameter λ_1, is given by

$$\hat{\beta}^{(\text{ridge})} = \frac{\hat{\beta}^{(\text{OLS})}}{1 + \lambda_1}. \tag{7.1}$$

Further, the solution of lasso regression with parameter λ_2, is given by

$$\hat{\beta}^{(\text{lasso})} = (|\hat{\beta}^{(\text{OLS})}| - \frac{\lambda_2}{2})_+ \text{sgn}(\hat{\beta}^{(\text{OLS})}) \tag{7.2}$$

where $\text{sgn}(\cdot)$ is the sign function, and $(x)_+$ denotes the positive part of the argument x, which equals x if $x > 0$ and 0 otherwise. The naïve elastic net solution is then

$$\hat{\beta}^{(\text{e.net})} = \frac{(|\hat{\beta}^{(\text{OLS})}| - \frac{\lambda_2}{2})_+}{1 + \lambda_1} \text{sgn}(\hat{\beta}^{(\text{OLS})}). \tag{7.3}$$

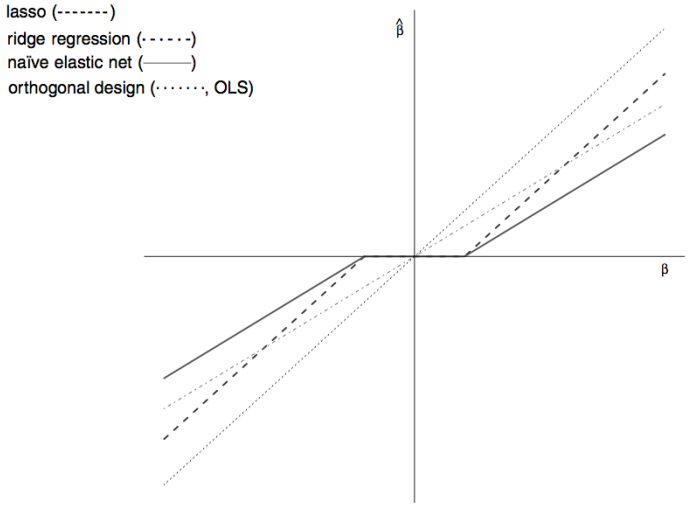


Figure 7.1: The elastic net (with shrinkage parameters $\lambda_1 = 1$, $\lambda_2 = 2$), compared to the other prior distributions and ordinary least squares regression. (Figure from Zou and Hastie (2005).)

We notice how (7.3) is a combination of (7.2) and (7.1), and in figure 7.1 this is shown graphically. We can see how the naïve elastic net solution, shown as a solid line, combines the shrinkage of the ridge solution by having a reduced gradient compared to the OLS solution, and the sparseness of the lasso solution by being zero for low OLS values.

(7.3) is called the *naïve* elastic net solution because it incurs a double amount of shrinkage—it first finds the ridge regression coefficients, which are already scaled down, and then does the lasso-type shrinkage. This straightforward double shrinkage introduces unnecessary extra bias. This can be corrected by introducing an augmented data set and penalty parameters (cf. Zou and Hastie (2005) for the details).

The elastic net has previously not been applied to text categorization. It was our intention to test it on our corpora. We briefly explored two possible ways of doing this. One possibility would be to alter the program code of the BBR toolkit to use this prior. However, our programming skills in C++ proved insufficient to follow this route.

The other possible route would be to change the input to BBR rather than the toolkit itself. Zou and Hastie (2005) outlines a way to transform an elastic net optimization problem to a lasso optimization problem; this would enable us to use BBR in its present form. Nonetheless, it still presents some implementational challenges, which we were unable to meet in the 20 weeks available for the work in this thesis.

These ideas for future work represent some of the possible avenues further inquiry into text categorization might take. The past fifteen years have seen great advancements being made in TC. We hope in this thesis to have given reason to believe that the coming years might see further improvement.

Bibliography

Apte, C., Damerau, F., Weiss, S. M., 1994. Automated learning of decision rules for text categorization. Information Systems 12 (3), 233–251.
URL citeseer.nj.nec.com/apte94automated.html

Baeza-Yates, R., Ribeiro-Neto, B., 1999. Modern Information Retrieval. ACP Press/Addison-Wesley.

Burges, C. J. C., 1998. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2 (2), 121–167.
URL citeseer.ist.psu.edu/burges98tutorial.html

Castelli, V., 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), 2102–2117.
URL http://dblp.uni-trier.de/db/journals/tit/tit42.html#CastelliC96

Cohen, W. W., Singer, Y., 1996. Context-sensitive learning methods for text categorization. In: Frei, H.-P., Harman, D., Schäuble, P., Wilkinson, R. (Eds.), Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval. ACM Press, New York, US, Zürich, CH, pp. 307–315.
URL citeseer.nj.nec.com/cohen96contextsensitive.html

Dayanik, A., Lewis, D. D., Madigan, D., Menkov, V., Genkin, A., 2006. Constructing informative prior distributions from domain knowledge in text classification. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM Press, New York, NY, USA, pp. 493–500.

Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W., Harshman, R. A., 1990. Indexing by latent semantic analysis. Journal of the Society for Information Science 41 (6), 391–407.

Dempster, A. P., Laird, N. M., Rubin, D. B., 1977. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society 39 (1), 1–38.

Domingos, P., Pazzani, M. J., 1997. On the optimality of the simple bayesian classifier under zero-one loss. Machine Learning 2-3 (29), 103–130.

Dunham, M. H., 2002. Data Mining: Introductory and Advanced Topics. Prentice-Hall.
URL http://www.seas.smu.edu/~mhd/book

Feldman, R., Sanger, J., 2007. The Text Mining Handbook – Advanced Approaches in Analyzing Unstructured Data. Cambridge University Press.

Fisher, R. A., 1936. The use of multiple measurements in taxonomic problems. Annals of Eugenics 7 (7), 179–188.

Fletcher, R., 1987. Practical Methods of Optimization, 2nd Edition. John Wiley and Sons.

Forman, G., 2003. An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research 3, 1289–1305.

Genkin, A., Lewis, D. D., Madigan, D., August 2007. Large-scale bayesian logistic regression for text categorization. Technometrics 49 (3), 291–304.
URL http://dx.doi.org/10.1198/004017007000000245

Hamilton, A. G., 1988. Logic for Mathematicians. Cambridge University Press.

Hersh, W., Buckley, C., Leone, T. J., Hickam, D., 1994. Ohsumed: an interactive retrieval evaluation and new large test collection for research. In: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval. Springer-Verlag New York, Inc., pp. 192–201.

Hersh, W. R., Bhupiraju, R. T., Ross, L., Johnson, P., Cohen, A. M., Kraemer, D. F., 2004. TREC 2004 genomics track overview.
URL http://trec.nist.gov/pubs/trec13/papers/GEO.OVERVIEW.pdf

Hull, D., 1994. Improving text retrieval for the routing problem using latent semantic indexing. Vol. 1994 of SIGIR ForumSpecial issue. Springer, London, pp. 282–291.

Ittner, D. J., Lewis, D. D., Ahn, D. D., 1995. Text categorization of low quality images. In: Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval. Las Vegas, NV, pp. 301–315.

Joachims, T., 1997. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In: Proceedings of the 14th International Conference on Machine Learning (ICML'97). pp. 143–151.

Joachims, T., 1998. Text categorization with support vector machines: Learning with many relevant features. In: European Conference on Machine Learning (ECML). Springer, Berlin, pp. 137 – 142.

Joachims, T., 2002. Learning to Classify Text Using Support Vector Machines – Methods, Theory and Algorithms. Kluwer/Springer.

Jurafsky, D. S., Martin, J. H., 2000. Speech and Language Processing: An Introduction to Natural Language Processing and Computational Linguistics and and Speech Recognition. Upper Saddle Riverand NJ:Prentice Hall.

Koller, D., Sahami, M., 1997. Hierarchically classifying documents using very few words. In: Proceedings of ICML-97, 14th International Conference on Machine Learning. Nashville, Tennessee, pp. 170–178.

Larkey, L. S., Croft, W. B., 1996. Combining classifiers in text categorization. In: Frei, H.-P., Harman, D., Schäuble, P., Wilkinson, R. (Eds.), Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval. ACM Press, New York, US, Zürich, CH, pp. 289–297.
URL citeseer.nj.nec.com/larkey96combining.html

Lewis, D. D., 1992. An evaluation of phrasal and clustered representations on a text categorization task. In: SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval. ACM Press, New York, NY, USA, pp. 37–50.
URL http://portal.acm.org/citation.cfm?id=133172&dl=GUIDE,

Lewis, D. D., Gale, W. A., 1994. A sequential algorithm for training text classifiers. In: Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval. Dublin, Ireland.

Liau, C., Alpha, S., Dixon, P., 2003. Feature preparation in text categorization. In: Proceedings of Autralasian Data Mining Worksop. Canberra, Australia.

Lin, H.-T., Lin, C.-J., 2003. A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods.
URL citeseer.ist.psu.edu/article/lin03study.html

Luenberger, D. G., 1984. Introduction to Linear and Nonlinear Programming., 2nd Edition. Addison-Wesley, Reading, MA.

Madigan, D., Genkin, A., Lewis, D. D., Argamon, S., Fradkin, D., Ye, L., 2004. Author identification on the large scale.
URL citeseer.ist.psu.edu/742899.html

McCullagh, P., Nelder, J. A., 1989. Generalized Linear Models. Chapman and Hall.

Mitchell, J., 1994. Machine Learning, Neural and Statistical Classification. Ellis Horwood, Ch. 3.

Mitchell, T. M., 1996. Machine Learning. McGraw Hill, New York, NY, p. 244.

Mitkov, R. (Ed.), 2003. The Oxford Handbook of Computational Linguistics. Oxford Handbooks in Linguistics.

Neumann, G., Schmeier, S., 1999. Combining shallow text processing and machine learning in real world applications. In: Joachims, T., McCallum, A., Sahami, M., Ungar, L. (Eds.), IJCAI99 Workshop on Machine Learning for Information Filtering.

Næss, A. B., july 2007. Statistical Machine Translation: Examining models for estimating translation probabilities. Project report, Norwegian University for Science and Technology (NTNU).

Pelckmans, K., Suykens, J. A., Moor, B. D., 2004. Morozov, ivanov and tikhonov regularization based ls-svms.
URL citeseer.ist.psu.edu/686505.html

Porter, M. F., 1980. An algorithm for suffix stripping. Program (Automated Library and Information Systems) 14 (3), 130–137.

Rocchio, J. J., 1971. Relevance feedback in information retrieval. Gerard Salton, editor, The SMART Retrieval System - Experiments in Automatic Document Processing, 313–323.

Salton, G., Wong, A., Yang, C. S., November 1975. A vector space model for automatic indexing. Commun. ACM 18 (11), 613–620.
URL http://dx.doi.org/10.1145/361219.361220

Schapire, R. E., Singer, Y., Singhal, A., 1998. Boosting and rocchio applied to text filtering. In: Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval. ACM Press, New York, US, Melbourne, Australia, pp. 215–223.

Schütze, H., Hull, D. A., Pedersen, J. O., 1995. A comparison of classifiers and document representations for the routing problem. In: Proceedings of SIGIR-95, 18th ACM International Conference on Research and Development in Information Retrieval. Seattle, Washington, pp. 229–237.

Sebastiani, F., March 2002. Machine learning in automated text categorization. ACM Computing Surveys 34 (1), 1–47.

Shannon, C. E., 1948. A mathematical theory of communication. The Bell System Technical Journal 27, 379–423.
URL http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society 58, 267–288.

van Rijsbergen, C., 1979. Information Retrieval, 2nd Edition. Butterworths, London.

van Rijsbergen, C. J., 1977. A theoretical basis for the use of co-occurence data in information retrieval. Journal of Documentation 2 (33), 106–119.

Vapnik, V. N., 1982. Estimation of Dependences Based on Empirical Data. Springer Verlag, NY.

Vapnik, V. N., 1995. The Nature of Statistical Learning Theory. Springer, New York, NY, USA.

Velldal, E., 2003. Modeling word senses with fuzzy clustering. Master's thesis, University of Oslo (UiO).
URL citeseer.ist.psu.edu/article/velldal03modeling.html

Wu, X., 2006. Incorporating large unlabeled data to enhance em classification. J. Intell. Inf. Syst. 26 (3), 211–226.
URL http://dblp.uni-trier.de/db/journals/jiis/jiis26.html#Wu06

Yang, Y., 1994. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In: Croft, W. B., van Rijsbergen, C. J. (Eds.), SIGIR. ACM/Springer, pp. 13–22.
URL http://dblp.uni-trier.de/db/conf/sigir/sigir94.html#Yang94

Yang, Y., 1999. An Evaluation of Statistical Approaches to Text Categorization. Information Retrieval 1 (1-2), 69–90.

Yang, Y., Chute, C. G., 1994. An example-based mapping method for text categorization and retrieval. ACM Trans. Inform. Syst. 12 3, 252–277.

Yang, Y., Liu, X., 1999. A re-examination of text categorization methods. In: Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval. Berkeley, California, pp. 42–49.

Yang, Y., Pedersen, J. O., 1997. A comparative study on feature selection in text categorization. In: Fisher, D. H. (Ed.), Proceedings of ICML-97, 14th International Conference on Machine Learning. Morgan Kaufmann Publishers, San Francisco, US, Nashville, US, pp. 412–420.
URL `citeseer.ist.psu.edu/yang97comparative.html`

Zhang, T., Oles, F. J., 2001. Text categorization based on regularized linear classification methods. Information Retrieval 4 (1), 5–31.
URL `citeseer.ist.psu.edu/zhang00text.html`

Zou, H., Hastie, T., 2005. Regularization and variable selection via the elastic net. Journal Of The Royal Statistical Society Series B 67 (2), 301–320, available at http://ideas.repec.org/a/bla/jorssb/v67y2005i2p301-320.html.