Using Support Vector Machines to solve large-scale and complex real world text categorization problems

Guido van Schie
(guidovanschie@hotmail.com)

24th August 2004

Supervisors: Prof.dr. G.R. Renardel de Lavalette
Dr. M. Biehl

Abstract

With the massive growth of the use of computers and the internet in the past decade, there has been an explosion on the volume of electronic documents and mail. Due to this, people are becoming unable to make use of all this information, and in order to keep it comprehensible to people, it is necessary to order these documents into (hierarchical) categories. Classifying natural language text documents into a fixed number of predefined categories, is called text categorization or text classification (TC) and is used for tasks like: email ordering / spam filtering, topic identification, document organization and Web searching (e.g. Yahoo!). However, this increasing amount of available information, also increased the size and complexity of these tasks. Doing these tasks manually, became therefore, very time-consuming and costly. This resulted in an increasing demand for automatic text classifiers.

In the past decade, a machine learning algorithm called Support Vector Machines (SVM), gained a lot of popularity for constructing automatic text classifiers. In this thesis I will do an elaborative study on automatic text classification in general, and on these Support Vector Machines (SVM) in particular. Based on this study I will set up a research, that will deal with the choices that can be made in the design process of an automatic text classifier, and conduct this research on a large-scale and complex real world problem, to see what choices can best be made for such problems. The problem that will be used in this research, is the text classification task of 2ehands.nl.

Based on the results of this research, a guide will be created with which such problems can be solved.

Used variables in chapter 1

D	=	Set of all possible documents.
D_T	=	Set of training documents; $D_T = \{d_1, \ldots, d_n\} \subseteq D$.
n	=	Number of documents in the trainset D_T.
d_i	=	A document.
t_j	=	A feature, to represent a document with.
$\vec{a_i}$	=	Feature vector of document d_i; $\vec{a_i} = (a_{i1}, \ldots, a_{if})$,
a_{ij}	=	The weight that represents the importance of feature j to document i.
f	=	Number of features.
nf_{ij}	=	represents the frequency of feature j in document i.
df_i	=	represents the number of documents in which feature i occurs at least once.
gf_i	=	represents the total number of times feature i occurs in the whole collection.
C	=	Set of classes; $C = \{c_1, \ldots, c_m\}$.
m	=	Number of classes.
$\text{class}(d)$	=	$D \rightarrow C$; $\text{class}(d) \in C$ for all $d \in D$.
c_i	=	Class label of document d_i; $c_i = \text{class}(d_i)$
$\|\vec{w}\|$	=	$\sqrt{\sum_{i=1}^{n} w_i^2}$ (Vector norm).

Used variables in chapter 3

The variables used in chapter 3 differ from those used in chapter 1, because this is the standard notation for describing SVM.

d	=	Number of features.
$\vec{x_i}$	=	Feature vector of a document; $\vec{x_i} \in \mathbb{R}^d$.
y_i	=	Class label of a document; $y_i \in \{-1, +1\}$.
n	=	Number of documents.
$\vec{w}$	=	Weight vector.
b	=	Bias.
$\vec{\alpha}$	=	Lagrange multipliers; $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
$\vec{\xi}$	=	Slack variables; $\vec{\xi} = (\xi_1, \ldots, \xi_n)$
k	=	Number of classes.
$\|\vec{w}\|$	=	$\sqrt{\sum_{i=1}^{n} w_i^2}$ (Vector norm).

Contents

Introduction

With the massive growth of the use of computers and the internet in the past decade, there has been an explosion on the volume of electronic documents and mail. With this increasing amount of available information, people are becoming unable to make use of all this information. The best way to keep all this data comprehensible to people is by ordering the documents into (hierarchical) categories. This way people can efficiently find what they want, by browsing through the categories. Classifying natural language text documents into a fixed number of predefined categories, is called text categorization or text classification (TC). In this thesis these terms are used interchangeably as well as the terms category and class. Text categorization is used in a lot of different fields like:

- **Topic identification**
  Determining the topics of documents.
- **Email ordering**
  Ordering email by for example placing all email from work in a 'work' folder, all personal email in a 'personal' folder and all received spam email in a 'spam' folder.
- **Document organization**
  Organising files in folders and subfolders.
- **Web searching**
  A hierarchic category based search engine for the web, with which people can browse through topics to find the pages they are looking for (e.g. Yahoo!).

Doing these classification tasks manually, is very time-consuming and costly, therefore people started looking for ways to automate these tasks.

Until the late 80's, most automatic text classifiers were manually constructed by experts who encoded their knowledge on how to classify documents in a set of rules. Usually the experts created one rule per category where each rule was of the form: IF (classifying Boolean statement for category i) THEN category = i. The advantage of this manual construction of an automatic classifier is that the classification task is now done automatically, but the drawback is that now building a classifier has become time-consuming and costly, because building such a classifier requires a lot of expertise.

Since the early 90's a lot of research has been done on how to apply machine learning techniques to text categorization. The great advantage of constructing classifiers by using machine learning techniques over the manual construction of classifiers using a set of rules, is that the classifiers do

not have to be built by experts anymore, but are automatically built by an inductive process that "learns" from previously classified documents. Research has shown that the use of machine learning techniques for the construction of text classifiers has been very successful and can compete with (and sometimes even outperform) manual classification by experts. In this thesis we will take a closer look at this type of automatic text categorization.

In order to build such an automatic text classifier that is able to determine the class of a document, a bunch of documents that are already classified are needed, so that the machine learning algorithm can learn from these documents how to do the classification task. These documents, that in fact are just sequences of words, understandable to humans, first need to be transformed into a form that is understandable to the machine learning algorithms. This form is called a feature vector and is some suitable vector representation of the document. A feature is some appropriate content identifier that forms a representation of a document. Such a feature vector of a document consists of features with which the document is represented, and weights for these features that determine the importance of each feature for this document. The feature vector representations of a set of documents can then be used to learn the machine learning algorithm how to classify documents.

There are several different machine learning algorithms, and each one has its own different way of learning from a set of documents. Machine learning methods like k-Nearest Neighbor, Naïve Bayes and Neural Networks have been used for years now to make automatic text classifiers. However, in the late 90's a 'new' machine learning algorithm, named Support Vector Machines (SVM for short) came into the picture. This method was in fact already proposed in 1974 by the Russian mathematician Vladimir Vapnik [45], but was not considered a very popular machine learning method for quite some time, because of its large needs of computer processing power, and its lack of applicability. However, in the late 90's and the beginning of the 21st century, some extensions to, and changes of, the original algorithm, in combination with the increasing processing power of the newest computers, led to an increasing popularization of the SVM method. All this made that the SVM algorithm became a noticeable competitor to the standard machine learning algorithms in these years and is now even considered one of the best machine learning algorithms for automatic text classification problems. This is why we will have a thorough look at these SVMs, and their use to automatic text categorization problems, in this thesis.

With the massive growth of electronic documents and mail, not only the need for automatic text classifiers for practical problems increased, but the size and complexity of these problems increased as well. Therefore it has

become interesting to take a look at how well text classifiers work for these large and complex real world text categorization problems, and how these classifiers can best be constructed. The objective of this thesis will therefore be:

Making a contribution to solving large-scale and complex real world text categorization problems.

In order to make this contribution we will make a guide on how to construct automatic text classifiers for such real world text categorization problems in large and complex domain. We will make clear what choices can best be made in the design process, and how to fine-tune the classifier so that it performs best. The main research question of this thesis that will be answered will therefore be:

What choices can best be made in the design process of an automatic text classifier for large-scale and complex real world text categorization problems?

In order to give an answer to this question, we will first do an elaborate study on the design process of an automatic text classifier. After that we will set up a research that will deal with the choices that can be made in the design process, and conduct this research on a real world problem in large and complex domain, to see what choices can best be made. This real world problem will be the text classification task of 2ehands.nl.

2ehands.nl is a website where supply and demand of secondhand products come together. People who would like to sell a secondhand product, can place a message with a description of the product on 2ehands.nl, and people who are looking for a certain secondhand product can go to 2ehands.nl and look for a message wherein someone is offering the product they are looking for. On average, a total of 50000 people a day are visiting 2ehands.nl, and each day an average of 5000 new messages are placed on 2ehands.nl. To make all these messages conveniently arranged and well searchable for the people who are looking for secondhand products, people that are offering their secondhand products have to place their messages in one of a set of predetermined categories (and subcategories), that range from 'cars' to 'animals'. The people that are looking for second hand products can now easily browse through the categories and subcategories, in search for the products they are looking for. However, placing a product in the correct category and subcategory can often be a difficult task. When the number of categories and subcategories is large, people will lose track of all the possibilities and will place their products in the first category that looks more or less appropriate, even when this might not be the correct one. Placing

products in wrong categories will contaminate the website and will make the secondhand products less easy to find, and will therefore make the website less useful to its visitors. Letting the website administrator manual check every one of the 5000 messages that are placed each day, would cost far too much time and money. Therefore, it would be very interesting to automate this process of placing products in the correct categories, or at least help the people, in an intelligent way, to do this correctly themselves. An automatic text classifier for this real world problem would therefore be of great interest.

But what makes this problem of 2ehands.nl so difficult, and why does this real world problem belong to the large and complex domain?

1. This problem of 2ehands.nl is a very large problem with a lot of different categories and subcategories. A product can be placed in one of 144 categories and besides that in one of averagely 20 different subcategories per category. The more categories a problem has, the more categories the classifier has to distinguish between, and the more difficult it will be for the classifier to correctly classify new messages.

2. The data with which the classifier has to be trained is contaminated, because (until now), products have in some cases been placed in wrong categories, and were not checked manually. This is why some of the messages of the products that are being sold will have an incorrect classlabel attached to it. Like humans, classifiers find it difficult to learn from partly incorrect data. (Try learning to understand the definition of 'monkey' when you are sometimes told that a bird and a cow are also monkeys.) Checking all the traindata for correctness before we start training, is nearly impossible and would take far too much time, so we just have to use this contaminated data to train with. That this will most likely have a negative impact on the classifier is trivial.

3. Each message of a secondhand product that is offered for sale, will only contain a few sentences or in some cases even a few words. It is easy to understand that when messages contain little information, it will be hard for a classifier to tell to which categories these messages belong.

4. The messages will often contain a lot of strange abbreviations, a lot of incorrect Dutch sentences and a lot of spelling mistakes. This will also make the construction of a good classifier more difficult.

5. The messages will have an extremely imbalanced distribution over the set of categories. Some categories will contain more messages than the others, simply because some types of products will be offered for sale more frequently than other types of products. An animal, for instance,

will be offered for sale far less frequently than a car. Training with this imbalanced data could influence the performance of the classifier.

Considering these five facts, we have good reason to state that the problem of 2ehands.nl belongs to the large and complex domain and is a good problem to conduct the research on, in order to give answer to the research question.

We will conclude this introduction by describing how this thesis will be set up. We will start in chapter 1 with an exhaustive and independent overview of all the aspects of automatic text categorization, by describing what choices can be made in the design process of an automatic text classifier. We will describe some different methods to convert text documents into feature vectors, and a number of different machine learning algorithms that construct automatic text classifiers by learning from these feature vectors. Subsequently we will explain in chapter 2 why we will use the SVM machine learning algorithm in this thesis, by showing why SVMs are so suitable for text categorization problems. We will do this by showing both theoretical and practical cases. In chapter 3 we will then take a thorough look at these SVMs, and describe its mathematical foundation and design questions. After describing every aspect of automatic text categorization and SVMs in the first three theoretical chapters (1 through 3), we will start with our own research in chapter 4. In this chapter we will set up the research with which we will give an answer to the research question of this thesis, and thereby satisfy the objective of this thesis. Chapter 5 will then show the results of the research we have set up in chapter 4, and chapter 6 will show the conclusions that can be drawn from these results. Finally we will conclude this thesis with some notes about issues that were left unanswered in this thesis and will need to be investigated in future research.

1 Automatic Text Categorization

In the introduction of this thesis, we have seen that there is an increasing demand for automatic text categorization (ATC). In this chapter we will now describe what ATC exactly is, and take an extensive and objective look at all the different aspects of ATC. Each section of this chapter will deal with an aspect of the design process of an automatic text classifier. But first of all, let us begin by formulating ATC:

Given a set of documents $D_T = \{d_1, \ldots, d_n\} \subseteq D$ (where D is the set of all possible documents), and an associated class label $c(d_i) \in \{c_1, \cdots, c_m\}$ for each one of these documents, ATC denotes the activity of automatically building, by means of machine learning techniques, an automatic text classifier that is capable of estimating the true class label of a new unseen document $d \in D$.

To make natural language text documents –which typically are strings of characters– suitable for machine learning methods to learn from, we will need to transform the documents into some sort of suitable vector representation, called a feature vector. A feature is some appropriate content identifier that forms a representation of the document, and in order to make the total feature vector a good representation of the document, some well chosen features are needed. A feature can be anything, like the language a document is written in, the length of a document, or the occurrence of a word. A feature vector $\vec{a_i} \in \mathbb{R}^f$ of a document d_i then consists of f weights, where each weight represents the importance of the associated feature for representing the document. The constructed feature vector of a document can then be used as input for the machine learning algorithms and constructed classifiers.

Before the texts are transformed into feature vectors, some pre-processing tasks can be performed that can make the learning task easier and the classification task better. In section 1.1 we will take a look at these pre-processing tasks. In section 1.2 we will see what kind of features we can choose to make a good representation of a document and in section 1.3 we will then take a look at some different methods with which weights can be assigned to these features, in order to transform a document into its feature vector. For ATC problems these feature vectors normally tend to be very large, what makes ATC problems hard to deal with. In section 1.4 we will discuss some algorithms that are able to drastically reduce the size of a feature vector, without harming the representativeness for its document. Finally, in section 1.5, we will take a look at a few machine learning techniques that can, in some sort of way, automatically construct a classifier, by learning from the feature vector representations of a set of pre-classified documents.

1.1 Pre-processing

Some pre-processing tasks, that can be applied before a document is transformed into a feature vector, can make the learning task easier and the classification task better. In this section we will discuss three of such pre-processing tasks.

1.1.1 Stopword extraction

The pre-processing task that is almost always used when constructing feature vectors for a text classifying task, is stopword extraction i.e. removing words that carry no information but occur frequently in almost every text. Stopwords are functional or connective words that are assumed to have no information content. Typical stopwords are words like: "the", "of", "and", "to" and "a". These words are the top five most frequent words in English natural texts, but contain no information of what a text is about and are therefore no good content classifiers. A stoplist is the list of all the stopwords (usually around a 100 words) that can be eliminated from the documents without harming the distinguishability of the documents.

1.1.2 Stemming

A word stem is the main part of a word to which affixes are added to form a new word or functioning of that word. By word stemming we mean the process of removing the affix of a word, so that only the word stem of the word will be left over. For example, word stemming will convert the words "extraction", "extracted", "extractable" and "extractor", to the same word stem: "extract". This way words that have the same word stem will be grouped and will be treated as one and the same word. The frequency of occurrence of the word stem for example, will be equal to the sum of the frequencies of occurrence of the variant words.

The underlying assumption of word stemming is that words that have the same word stem will have similar meanings. This however is obviously not always true. Take for example the (homonymous) word stem, "board", of the words "boarding" and "board" in the sentences "the sailors were boarding" and "board of directors". These two words have the same word stem: "board", but have a completely different meaning. Examples like this made the use of word stemming in text classification controversial. Nevertheless, research in information retrieval has shown that using word stems instead of words in some cases can significantly improve classification.

There are two main ways of finding the word stem of a word. The first one is by using a set of suffix stripping rules. The second one is by looking up

the word stem of a word in a dictionary.

Suffix stripping

A suffix stripping algorithm consists of a set of rules that determine how a word has to be altered to convert it to his word stem. One of the best, most used and most sophisticated suffix strippers is the stemming algorithm developed by Porter [36]. Porter's algorithm is based on a series of steps that each remove or replace a certain part of the word by way of substitution rules. Each rule can have a condition rule which has to hold for the substitution rule to count, e.g. the resulting stem must have a certain minimal length.

The advantage of such a suffix stripping stemming method is that it is very robust and fairly easy to implement. The disadvantage is that usually a lot of words are reduced to incorrect word stems. In Dutch text stemming [18], for instance, a lot of errors will be generated because of the large number of so-called strong verbs in the Dutch language. Strong verbs are verbs whose past and participle forms have root vowels which differ from that in the present tense root (i.e. present tense nemen (to take) has a past tense nam (took) and a participle genomen (taken)). Such forms will not be reduced to the same word stem and will cause an stemming error.

Dictionary lookup

One way to avoid the under-stemming errors that occur when a strong verb is converted into a word stem, is by using a dictionary wherein the word stems can be looked up. It is obvious that looking up a word stem in a dictionary will provide better stemming results than suffix stripping, but it has its drawbacks, One of the drawbacks of using dictionary lookup is that it is time and memory consuming. It costs time to look up every word in the dictionary and it costs memory space to store the dictionary. Another drawback of this method is that it is not as robust as suffix stripping, because of the limited available words in a dictionary. Words that do not occur in the dictionary will not be converted into a word stem. Other words that have multiple occurrences with different word stems in the dictionary will be converted into the most frequent word stem. A bigger dictionary means less errors, but also implies the need for more time and memory capacity.

A logical next step is to combine both stemming methods. By combining the two methods, a stemmer can get the precision of the dictionary lookup method while getting the robustness of the suffix stripping method. Such a method will at first try to lookup the word stem in the dictionary; by failure it will apply the suffix stripping method to convert a word into his

word stem. This combination of methods has shown to provide very good stemming results [18].

1.1.3 Class imbalance

In a lot of text categorization problems, the documents that are available for training the classifier, are extremely non-uniform distributed over the categories. This means that there is not an equal amount of data per category. One category can be represented by a large number of examples, while the other is represented by only a few. The class imbalance problem can cause a significant reduction in the performance of a text classifier, that is trained with a method which assumes a balanced distribution of the classes. There are two main kinds of methods to tackle the class imbalance problem: over-sampling and down-sizing (see [20]).

Over-sampling

Over-sampling methods handle the class imbalance problem by creating new examples for the underrepresented class, by over-sampling the examples that are already available for the class. One common used way of over-sampling is by replicating examples of the small classes at random until they contain as many examples as the larger classes. Another way is by over-sampling the smaller data set in a focused manner, concentrating on the data located close to the boundaries.

Down-sizing

Another way of getting the classes balanced is not by creating examples for the underrepresented classes, but by eliminating examples of the over-sized classes. One way to do this is by randomly removing examples from the over-sized class, until it matches the size of the other classes. Because of the obvious drawback of this method, that it can remove potentially useful data, another method is introduced. This method works by down-sizing the larger data set, concentrating on saving the points near the boundaries.

1.2 Features

One of the most important things when building an automatic text classifier is the choice of the features with which the documents will be represented. You need to choose features that form a good representation of the text documents, if you want to construct a good text classifier. In this section we will discuss four of the most important ways to transform a document into a feature vector: bag of words, word n-grams, linguistic approach and character n-grams.

1.2.1 Bag of words

The most successful and most used way of representing a text in automatic text classification is the Bag Of Words (BOW) approach. In short BOW just means that every distinct word of a text is used as a separate feature. This method is called Bag Of Words, because originally this approach –as the term Bag Of Words implies– was used to construct a histogram of word frequencies. Later when more sophisticated weights than just the frequencies of words were used to represent the importance of the features, and the term "Bag" was not really applicable anymore, the term Bag of Words was still used for this method. In section 1.3 some of these more sophisticated ways of representing the importance of features will be discussed in detail.

The process of making a BOW representation of a text approach is fairly simple. After converting all letters into lowercase letters and removing punctuation in all the documents, the documents are split by white spaces into separate words. The collection of distinct words now forms the vocabulary of this set of documents and every distinct word of the vocabulary forms a separate feature. A document can now be represented by assigning values to each feature, reflecting the presumed importance of that feature to the document.

This way of representing a text by using every word as a feature is based on the following observations made in information retrieval research:

- The frequency of occurrence of distinct words in a document represents the importance of that word for the category it belongs to. This can be explained by the following quote from H.P. Luhn [27]:

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

- The ordering of the words in a text is of minor importance for almost every text categorization task.

One thing about using every distinct word of a set of documents as a feature is that it becomes possible that every word of a natural language becomes a feature. One can imagine that, due to the immense amount of distinct words of a natural language, this can lead to an enormous feature vector. This high dimensionality of the feature space is one of the main problems when using the BOW approach for representing a document. High dimensional feature vectors require enormous amounts of computational time and memory resources to work with and can decrease the performance of the

constructed classifier by overfitting the problem. Overfitting is the phenomenon by which a classifier is tuned also to the contingent, rather than just the necessary (or constitutive) characteristics of the training data. In 1.4 the problems of high dimensional feature space and of course some solutions to these problems will be discussed in detail.

The stemming pre-processing task discussed in 1.1.2, can be very useful here, because more or less the same words will be grouped together and that can drastically reduce the number of different words that need to be taken into consideration. This way the vocabulary will be smaller and therefore reduces the dimensionality of the feature vector. Word stemming can reduce the size of the vocabulary with thirty to fifty percent. Another pre-processing task that is almost always used with the BOW approach is the task where stopwords are removed from the documents (1.1.1). This pre-processing task also reduces the dimensionality of the feature vector (with the number of stopwords), but besides that it can improve the performance of a classifier significantly. Performance can be improved because the removed stopwords, that do not contain any information of what a document is about whatsoever, will no longer interfere with the important features that do represent the documents.

1.2.2 Word n-grams

A very basic observation about the BOW representation is, that a great deal of the information from the original document is discarded. Paragraph, sentence and word order is disrupted, and syntactic structures are broken. The goal of using phrases as features is to attempt to preserve some of the information left out of the bag of words. Statistical phrases (as word n-grams are also called) is one way to preserve some dependencies between words and relative position of words. In most literature the word n-gram approach is usually just called the n-gram approach, but the word n-gram name is handled to make a distinction between this approach and the character n-gram approach that will be discussed in 1.2.4. The word n-gram approach can be seen as an extension to the BOW approach, with the only difference that, instead of a single word, a combination of words is used as a feature. In literature two types of n-grams are used, the first only constructs features that consist of exactly n words, while the second one can also construct features consisting of less than n words (word sequences of length $\leq n$).

When n successive words are used in the construction of features, word pairs that occur frequently together, can be represented by one single feature. For instance the word pair "wireless mouse", that consists of the two separate words "wireless" and "mouse", will frequently occur together and will together form a feature that can be a better category representative than the

two separate single word features apart. The word "wireless" will probably occur frequently in the "phone" category and the word "mouse" will probably occur very frequently in the "animal" category. Both words will also occur in the "computer" category, but the two words combined will only occur in the "computer" category and no longer in the "phone" and "animal" categories.

The use of word n-grams is still a bit controversially. Some reports comment that inclusion of n-grams may only duplicate information of existing unigrams (or one word features), while other reports show that the addition of n-grams (up to sequences of length $n = 3$) to the single words model, can improve the performance of a classifier (see [6][16][31]).

One of the main problems of the BOW approach was the problem of the high dimensionality of the feature space. With the use of n-grams, the number of possible features increases considerably, and the feature vector normally becomes even larger than with BOW. One can imagine the enormous amount of different sequences of length n that can be made from combining the words of a vocabulary. When the n-gram type is used where word sequences of length $\leq n$ are used, the number of possible features becomes enormous. Even when we mention that most of these possible features can not, or will not occur in normal natural language texts, the feature vector will still at least be an order of magnitude higher than with the BOW approach. This makes dimensionality reduction (as will be discussed in 1.4) an absolute must, when the n-gram approach is used.

1.2.3 Linguistic approach

With the use of n-grams instead of the BOW approach, some information about the word order is saved, but still a lot of information, crucial for humans to understand a text, is discarded. The phrases constructed with the n-gram approach are just statistical phrases and have nothing to do with the syntactic structures between words in a document. For example the sequence of words "another gold" in the sentence "Leontien wins another gold medal" is equally meaningful to the n-gram approach as "gold medal", while for humans only the latter one really makes sense. By selecting only those phrases that make sense to humans, some of the intelligence of humans about using syntactic structures between words can be copied. One way to obtain these most useful phrases, mostly indicating the objects and subjects of a text, is by only selecting the sequences that are recognized as noun phrases. Noun phrases can be obtained by using two separate algorithms: one for assigning part of speech tags (noun, verb, adverb, preposition, etc.) to the individual words, and one for grouping the tags into noun phrases. A well-known part of speech tagging algorithm is the rule based algorithm

developed by Eric Brill [4]. The grouping of the tags into noun phrases is simply done with a regular expression that matches sequences of nouns or adjectives terminating in a noun.

By using this noun phrases as features, some of the syntactic structures between words are used, but another really important thing to humans, about text is still ignored, namely, the semantic relationships between words. In particular, potentially useful information about synonymy and hypernymy is not directly used. A synonym is a word that has the same, or nearly the same, meaning as another word in a language and a hypernym of a word is a word that is more generic than the given word. Hypernyms are often called the "is a" relationship. For example, a "processor" is a "computer part", what makes "computer part" the hypernym of "processor". Making use of hypernyms and synonyms when constructing features, could simulate the way humans make use of their knowledge about words when reading a document. One way to try and make use of the sematic relationships between words, is by using WordNet [30]. WordNet is an online lexical reference system developed at Princeton University that contains information about synonyms and hypernyms of words. By mapping synonym words into the same word, and by generalizing words by looking up the hypernym of a word, some linguistic properties are taken into account for feature construction.

A major problem with this way of constructing features is the problem known in the natural language processing community as "word sense disambiguation" (funny enough this is a text categorization task by itself). For example, the previously mentioned word "processor" has two completely different synonyms, namely "the central processing unit of a computer" and "someone who processes things (foods or photographs or applicants etc.)". As a result of that, the word "processor" has two completely different hypernyms, namely respectively "computer part" and "worker". One can imagine the major negative impact on the performance of the classifier when the wrong choices are made. Despite the many efforts, however, researches [41][3] have shown that the use of these more sophisticated methods do not result in an improved performance. Quite disappointingly, many times it has even proven to drop down accuracy significantly. The main opinion about the use of sophisticated linguistic properties of documents is that a lot more research has to be done, before such methods will have a positive effect on the performance of text classification. Quite convincingly, Lewis [25] argues that the likely reason for the discouraging results is that, although features based on phrases have superior semantic qualities, they have inferior statistical qualities with respect to features based on single words. While linguistic methods may eventually prove essential, nowadays their performance still is not good enough.

1.2.4 Letter n-grams

A rather different way of representing a document is by leaving the approaches where features are based on (combinations of) words and start looking at building features from combinations of characters. The character n-gram approach is such a method that constructs features based on combination of characters. A character n-gram is a slice of a string consisting of n characters. Like the word n-gram approach, there is a variant of this method, with which features consisting of slices of length $\leq n$ are used. In most literature contiguous characters are used for feature construction, but in theory every combination of characters (i.e. the first and third character of a word) can be used. In order to help with matching beginning-of-word and end-of word situations, blanks are added to the beginning and end of a string.

The following example from William B. Cavnar and John M. Trenkle [7], will show, what character n-grams the word "TEXT" would be composed of, when the variant is used where features consist of slices with length $\leq n$. The underscore character ("_") represents the beginning and end blanks.

bi-grams	:	_T, TE, EX, XT, T_
tri-grams	:	_TE, TEX, EXT, XT_, T_ _
quad-grams	:	_TEX, TEXT, EXT_, XT_ _, T_ _ _

In general, a string of length s, padded with blanks, will have $s+1$ bi-grams, $s+1$ tri-grams, $s+1$ quad-grams, and so on. When this word alone already results in the construction of 15 features, it is clear that this method will also come up with a very high dimensional feature space.

The key benefit that character n-gram-based features provide, lies in its robustness. Since every string is decomposed into small parts, any errors that are present tend to affect only a limited number of those parts, leaving the remainder intact. This makes word n-grams especially useful in noisy text categorization tasks where textual errors occur very frequently.

1.3 Feature weighting

After the set of features is determined, numerical weights have to be assigned to each feature, in order to construct the feature vector. The weights should be chosen in such a way, that every weight reflects the presumed importance of the associated feature, for purpose of content identification of its document. Depending on how complex the calculations of the weights may be (considering the available amount of time), several feature weighting methods have been developed, varying from simple methods that only represent whether a feature occurs in a text or not, to more sophisticated methods that use information theoretic ideas to calculate the weights. In

this section we will discuss six of the most used feature weighting methods (see [1][37][40] for more information)

When the weights of every feature of a document are calculated, a document d_i will be represented by a feature vector $\vec{a_i} = (a_{i1}, \ldots, a_{if})$ where f is the number of features and each a_{ij} is the weight that represents the importance of feature j to document i. Normally, only a few of these weights will be non-zero, because only a few of the enormous amount of features will be important for a specific document. This will lead to a very sparse feature vector. In the six methods that will be described next, log will be the base-10 logarithm and the variables that will be used will represent the following:

- nf_{ij} represents the frequency of feature j in document i.
- df_i represents the number of documents in which feature i occurs at least once.
- gf_i represents the total number of times feature i occurs in the whole collection.
- n represents the total number of documents in the collection.
- f represents the number of features in the collection.

Boolean weighting

The most simple way of assigning values to features, in order to reflect the presumed importance of the associated feature, is by using the Boolean weighting method. This method only represents whether a feature does occur in the document or does not, by respectively assigning ones and zeros to the associated features. Boolean feature weighting is very easy to calculate and is often used in very simple text classification tasks.

$$a_{ij} = \begin{cases} 1 & \text{if } nf_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Frequency weighting

A slightly more complicated feature weighting method uses the frequency of occurrence of a feature itself as the representing weight. This way of feature weighting is based on the observation made in information retrieval research that suggests that when a feature occurs more frequently in a document it probably will be more relevant to the topic of the document.

$$a_{ij} = nf_{ij}$$

Tf-idf weighting

Another observation made in information retrieval, suggests that when a feature occurs more frequently throughout all documents in the collection, it will poorly discriminate between documents. The Term Frequency - Inverse Document Frequency (tf-idf) feature weighting method takes this observation into account and assigns the weights a_{ij} to the feature j in document i, in proportion to the number of occurrences of the feature in the document, and in inverse proportion to the number of documents in the collection for which the feature occurs at least once. This way, a high degree of importance will be assigned to terms, that occur in only a few documents of a collection.

$$a_{ij} = nf_{ij} \cdot \log\left(\frac{n}{df_j}\right)$$

Tfc weighting

The previously mentioned tf-idf weighting method does not take into account, that documents may be of different lengths. The tfc weighting method is similar to the tf-idf weighting method, with the only difference, that in this case, length normalization is used.

$$a_{ij} = \frac{nf_{ij} \cdot \log\left(\frac{n}{df_j}\right)}{\sqrt{\sum_{k=1}^{k=f}\left[nf_{ik} \cdot \log\left(\frac{n}{df_k}\right)\right]^2}}$$

Ltc weighting

In order to reduce the effects of large differences in frequencies, the slightly different ltc weighting method can be used. The effects of large differences in frequencies are reduced by using the logarithm of the feature frequency, instead of the raw feature frequency.

$$a_{ij} = \frac{\log(nf_{ij}+1) \cdot \log\left(\frac{n}{df_j}\right)}{\sqrt{\sum_{k=1}^{k=f}\left[\log\left(nf_{ik}+1\right) \cdot \log\left(\frac{n}{df_k}\right)\right]^2}}$$

Entropy weighting

Entropy weighting is the most sophisticated method and is based on information theoretic ideas. Entropy weighting takes the distribution of the features over the documents into account.

$$a_{ij} = \log(nf_{ij}+1) \cdot \left(1 + \frac{1}{\log(n)} \sum_{k=1}^{n}\left[\frac{nf_{kj}}{gf_j} \cdot \log\left(\frac{nf_{kj}}{gf_j}\right)\right]\right)$$

where

$$\frac{1}{\log(n)} \sum_{k=1}^{n} \left[\frac{nf_{kj}}{fg_j} \cdot \log\left(\frac{nf_{kj}}{gf_j} \right) \right]$$

is the average uncertainty or entropy of feature j. This quantity is -1 if the feature is equally distributed over all documents, and 0 if the feature occurs in only one document.

1.4 Dimensionality reduction

As we have repeatedly seen in 1.2, one of the main problems of text categorization, is the high dimensionality of the feature space. A good representation of a document can easily result in a feature vector consisting of tens of thousands of features or more. Processing these large feature vectors becomes extremely costly in computational terms and hard to deal with for standard classification techniques. Extremely large feature vectors can also easily lead to the overfitting problem. Overfitting is the phenomenon by which a classifier is tuned also to the contingent, rather than just the necessary (or constitutive) characteristics of the training data. Classifiers which overfit the training data, tend to be extremely good at classifying data they have been trained on, but are remarkably worse at classifying other data. Experimentation has shown, that in order to avoid overfitting, a number of training examples roughly proportional to the number of features used, is needed. This means that to avoid overfitting in high dimensional feature space problems like text categorization, an extreme amount of training examples will be needed. In most text categorization problems it is (nearly) impossible to gather such amounts of training examples. Besides that, it will take a lot of time to train a classifier with such amounts of data.

The problems mentioned above, make the use of dimensionality reduction in text categorization problems almost inevitable. Dimensionality reduction techniques try to drastically reduce the dimensionality of the feature space from f to $f' \ll f$, without harming the distinguishability of a document. In order to indicate how much a specific dimensionality reduction technique reduces the feature space, a proportion $(1 - \frac{f'}{f})$ between the number of original features f, and the number of remaining features f' is used. This proportion is called the aggressivity level of a dimensionality reduction technique. A high aggressivity level brings about high benefits in terms of computational cost, and also drastically reduces overfitting. On the other hand aggressive dimensionality reduction is more likely to remove important information for purpose of text categorization. A low aggressivity level can however even improve the performance of a classifier, if the removed features happen to be noise terms. Therefore, to choose the best aggressivity level, usually re-

quires some experimentation.

Dimensionality reduction techniques can be classified into one of two categories: feature selection and feature extraction. The difference between these two is, that feature selection methods use some sort of subset of the original set of features as the new set of features, while feature extraction methods construct a whole new set of features, based on the original set. In the remaining part of this section, some methods of both feature selection and feature extraction will be discussed (see [1][17][50] for more information).

1.4.1 Feature selection

Feature selection, also known as term space reduction, is the term used for the activity of selecting a subset of f' features from the complete set of f features, in such a way that the remaining $f' \ll f$ features still represent the meaning of the documents. Feature selection methods try to remove those features that contain no, or little, information about the topics of the documents. It is well known that features are not necessarily all relevant and beneficial for representing a text, and that it therefore is possible to remove a considerable amount of features, without harming the representation of a document. The question that rests is, which features can best be removed?

The methods described below, select a subset of a set of features by means of the so-called filtering approach. This filtering approach uses some sort of function for measuring the importance of a feature for the categorization task, and a threshold to select which features should be removed from the feature space and which should be kept. Therefore, the threshold determines the aggressivity level of the feature selection approach, and by adjusting it, the size of the new feature set can be chosen.

The probabilities that are used in the functions below can be estimated by counting occurrences in the training set. Take for instance the probability $P(\bar{t_j}, c_i)$; this indicates the probability that for a random document d, feature t_j does not occur in d and d belongs to category c_i. Estimating $P(\bar{t_j}, c_i)$ can be done by counting the number of documents from class c_i in which feature t_j does not occur and dividing it by the total number of documents of the trainset.

Everyone of the functions $f(t_j, c_i)$ evaluates the importance of feature t_j with respect to class c_i. In order to evaluate the importance of feature t_j globally (i.e. category independent), either the weighted average $f_{avg} = \sum_{i=1}^{m} f(t_j, c_i) \cdot P(c_i)$ or the maximum $f_{max} = \max_{i=1}^{m} f(t_j, c_i)$ can be used.

Document frequency

Document frequency is one of the most simplest feature selection methods; it simply counts the number of documents in which each feature occurs and removes those features that occur less than some predetermined threshold. The assumption that is used with this method is, that rare features will either be non-informative for category prediction or not influential in global performance. Features with less frequency will have less effect on the performance of the classifier. However, this contradicts the intuitive feeling that low frequency features could in fact be relatively important. Document frequency therefore looks like an ad hoc method at first sight. Yiming Yang and Jan Pedersen however showed in [50] that there is a strong correlation between Document frequency and the more complex weighting methods Information gain and χ^2 (discussed below), what would make Document frequency a reliable measure for removing unimportant features. The great advantage of document frequency over methods like information gain and chi-square, is, that it is far less expensive to calculate.

Mutual information

The assumption that is made with mutual information, is, that a high occurrence of the feature t_j in the class c_i will give high information of t_j in c_i. When t_j and c_i are independent, $MI(t_j, c_i)$ will equal zero.

$$MI(t_j, c_i) = \log\left(\frac{P(t_j, c_i)}{P(t_j) \cdot P(c_i)}\right)$$

Information gain

Information gain not only considers occurrence of a feature in a class, but also the non-occurrence.

$$IG(t_j, c_i) = P(t_j, c_i) \cdot \log\left(\frac{P(t_j, c_i)}{P(t_j) \cdot P(c_i)}\right) + P(\bar{t_j}, c_i) \cdot \log\left(\frac{P(\bar{t_j}, c_i)}{P(\bar{t_j}) \cdot P(c_i)}\right)$$

Chi-square

Chi-square (χ^2) is a statistic tool that is used to measure the difference between the result of an observation and the expected result, according to an initial hypothesis. In this case the hypothesis is, that a feature t_j and a class c_i are independent. This means that when $\chi^2(t_j, c_i)$ equals zero, t_j and c_i will be independent. A larger value of $\chi^2(t_j, c_i)$ indicates more dependence between c_i and t_j, and will make feature t_k more interesting for the classification task and less interesting for removal.

$$\chi^2(t_j, c_i) = \frac{n \cdot [P(t_j, c_i) \cdot P(\bar{t_j}, \bar{c_i}) - P(t_j, \bar{c_i}) \cdot P(\bar{t_j}, c_i)]^2}{P(t_j) \cdot P(\bar{t_j}) \cdot P(c_i) \cdot P(\bar{c_i})}$$

where n is the total number of documents.

Correlation coefficient

$$CC(t_j, c_i) = \frac{\sqrt{n} \cdot [P(t_j, c_i) \cdot P(\bar{t_j}, \bar{c_i}) - P(t_j, \bar{c_i}) \cdot P(\bar{t_j}, c_i)]}{\sqrt{P(t_j) \cdot P(\bar{t_j}) \cdot P(c_i) \cdot P(\bar{c_i})}}$$

Relevancy score

$$RS(t_j, c_i) = \log\left(\frac{P(t_j|c_i) + b}{P(\bar{t_j}|\bar{c_i}) + b}\right)$$

where b is a constant damping factor.

Odds ratio

$$OR(t_j, c_i) = \frac{P(t_j|c_i) \cdot (1 - P(t_j|\bar{c_i}))}{(1 - P(t_j|c_i)) \cdot P(t_j|\bar{c_i})}$$

1.4.2 Feature extraction

Feature extraction, also known as re-parameterization, is the process of constructing new features as combinations or transformations of the original features, in order to reduce the dimensionality of the feature space. This makes that the original features will no longer exist in their original form in the new feature space, but will be represented in some other way.

Term clustering

Term clustering groups features that are semantically related, so that these groups can be used as the features, instead of the individual features. This way the dimensionality will be reduced from the number of individual features to the number of groups.

There are mainly two ways of term clustering: unsupervised clustering and supervised clustering. Unsupervised clustering groups features only by looking at the co-occurrence of features in the documents, and does not take into account to which category a document belongs. Supervised learning does take into account the class-labels of the documents and groups features according to their class distribution $P(c_i|t_j)$.

Latent semantic indexing

Latent semantic indexing (LSI) reduces the dimensionality by mapping a feature vector into a lower dimensional space. These new dimensions are derived from the original features with a technique called singular value decomposition. It is beyond the scope of this thesis to discuss the mathematics of this transformation, but a good and comprehensive description can be found in [1]. LSI combines the original features based on the patterns of occurrence of these features, and in such a way is able to reveal some underlying or latent structure of feature usage across documents. LSI differs from the term clustering method in such a way that, while with term clustering, the new features were intuitively interpretable, this is no longer is the with LSI.

1.5 Machine learning algorithms

Now that we have seen how we can pre-process documents, find some appropriate features to represent these documents, assign some weights to these features to represent the importance of the features for the documents, and reduced the constructed feature vectors to a workable length, it is now time to take the final step: automatically construct a classifier by learning from a set of example documents. Every machine learning algorithm has a different way of looking at the classification problem and therefore has a different way of solving it. In this section, six of the most important machine learning algorithms will be discussed.

1.5.1 Rocchio's algorithm

Rocchio's algorithm [39] constructs a classifier by learning a prototype vector for each category. A prototype vector for category c_i is computed as an average vector over all training document vectors that belong to category c_i. A new document can then be classified by comparing it to each of the prototypes, and assigning it to the category of the best matching prototype. In order to find the best matching prototype for a document, the distance between the feature vector of a document and a prototype vector can be computed, and the prototype with the smallest distance to the document feature vector, is the best match. The distance between two vectors can be computed by, for instance, the dot product or by using the Jaccard similarity measure.

1.5.2 Linear Least Squares Fit

A classifier constructed with the Linear Least Squares Fit (LLSF) method [48] exists of a solution matrix that maps a document feature vector to a category weights vector. The weights of the category weights vector represent to what extent the document belongs to each category. In case of the

training documents, the category weights vector will be binary: a document belongs to a category or not. In LLSF every document has two vectors associated to it, an input feature vector $\vec{a_i}$ consisting of f weighted features, and an output vector $\vec{o_i}$ consisting of m weights representing the categories. Classification of a new document can now be seen as determining the output vector for the new document. Building a classifier then boils down to computing an $m \times f$ matrix $\hat{M}$ such that $\hat{M}\vec{a_i} = \vec{o_i}$. LLSF computes the matrix $\hat{M}$ from the training data by computing a linear least-squares fit that minimizes the error on the training set according to the formula [49]:

$$\hat{M} = \arg\min_{M} = \|MA - O\|^2$$

Where $\arg\min_M(a)$ stands for the M for which the argument a is minimum. A is the $f \times n$ matrix consisting of n columns, each representing an input vector of the training documents, and O is the $m \times n$ matrix consisting of n columns, each representing an output vector of the training documents. n is the total number of training documents. The $\hat{M}$ matrix is usually computed by computing a singular value decomposition on the training set. In the resulting matrix $\hat{M}$ the generic entry $\hat{m_{ij}}$ represents the degree of association between category c_i and feature t_j. One of the disadvantages of the LLSF method is that it is computational costly to determine the $\hat{M}$ matrix.

1.5.3 Naïve Bayes

A Naïve Bayes classifier is constructed by using the training data to estimate the probability of each category given the feature weights of a new document. The probabilities are estimated by using Bayes theorem:

$$P(c_j|\vec{a_i}) = \frac{P(c_j) \cdot P(\vec{a_i}|c_j)}{P(\vec{a_i})} \quad (1)$$

The naïve part of the Naïve Bayes classifier, is the assumption that the features of $\vec{a_i}$ are conditionally independent, given the category variable. This assumption makes the computation of the Naïve Bayes classifiers far more efficient than the exponential complexity of Non-naïve Bayes approaches, because it does not use word combinations as predictors. The assumption simplifies the probability of a document $\vec{a_i}$ given the category c_j to:

$$P(\vec{a_i}|c_j) = \prod_{k=1}^{f} P(a_{ik}|c_j)$$

Another simplification of the formula can be achieved by leaving out the denominator in equation (1) above. The denominator can be left out, because it does not differ between categories. These simplifications yield to

the following equation for computing the probability of a category, given the feature weights of a new document:

$$P(c_j|\vec{a_i}) = P(c_j) \cdot \prod_{k=1}^{f} P(a_{ik}|c_j)$$

The construction of a Naïve Bayes classifier can now be done by estimating $P(c_j)$ and $P(a_{ij}|c_j)$, given a set of training examples. $P(c_j)$ can be estimated by calculating the fraction of training documents that is assigned to category c_j:

$$\hat{P}(c_j) = \frac{n_j}{n}$$

where n_j is the number of training examples that belong to category c_j, and n is the total number of training examples. $P(a_{ik}|c_j)$ can be estimated by:

$$\hat{P}(a_{ik}|c_j) = \frac{1 + n_{kj}}{f + \sum_{l=1}^{f} n_{lj}}$$

where n_{kj} is the number of times feature k occurred within documents from category c_j in the training set. Although the assumption about feature independence is clearly incorrect, classification with a Naïve Bayes classifier can be surprisingly effective.

1.5.4 Neural Network

A Neural Network [29] imitates the human brain (to some extent) by representing a classifier with a network of neurons and some synaptic connections between them. A typical Neural Network consists of three layers: one layer of several input nodes (one node for every input feature), one layer of output neurons (one neuron for every category), and one layer of hidden neurons in between. Each input node is connected to each hidden neuron by a synapse, and each hidden neuron is connected to each output neuron by a synapse, where each synapse comes with a weight with which the following neuron can be excited or inhibited. For classifying a document d_i , its feature weights a_{ij} are assigned to the input nodes; the activation of these units is propagated forward through the network through the synapses, and the value that the output unit(s) take up, as a consequence, determines the categorization decisions. An activation of neuron is determined by summing up the products of the synaptic weights and the activation of the unit before, and an activation function to limit the amplitude of this activation. The learning process of a neural network classifier now boils down to finding a set of appropriate weights for the synapses in the neural network.

A neural network is usually learned with the error back-propagation algorithm. This algorithm basically learns a neural network in two passes

through the different layers: a forward pass and a backwards pass. In the forward pass the feature vector of a training document is applied to the input nodes of the network and is passed through the hidden neurons to the output neurons in order to produce a response of the network. During this forward pass, the synaptic weights of the network are all fixed. During the backward pass, on the other hand, the weights are adjusted in accordance with an error-correction rule. The error of an output neuron can be calculated by subtracting the actual response of the network from its desired (target) response, which is known for the training documents. This error is then propagated backward through the network, against the direction of the synaptic connections, and the synaptic weights are adjusted in such a way that the actual response of the network moves closer to the desired response in a statistical sense.

1.5.5 *K*-Nearest Neighbor

K-Nearest Neighbor (*k*-NN) does not perform any kind of explicit extraction from the documents of the training set, but just stores all examples of the training set along with their category labels. A new document is classified by looking at the k stored examples that are most similar to the new document. The classification is done by weighting the class-labels of these neighbors according to the similarity of each neighbor to the new document. The similarity between two documents is usually calculated by measuring, for example, the cosine or the Euclidean distance between the two feature vector representations of the documents. Because no explicit extraction of information is performed, *k*-NN does not describe a category in any way (except by the set of examples that belong to it), but it has been proven to perform remarkably well in text categorization [49].

1.5.6 Support Vector Machines

Another different way of solving a classification problem is by using Support Vector Machines (SVM) [45]. The SVM method tries to find a decision surface in the feature space that best separates the categories from each other. The best separating surface is the one where the distance to the nearest data point is as large as possible. In figure 1 and 2 this idea of best separation of categories is illustrated for a two-class problem. For sake of simplicity we will only show a two-class case in a two-dimensional feature space with linearly separable data points in this section, but in chapter 3 we will see that the SVM method can be generalized to solve multiclass problems with a high dimensional feature space and with data points that are not linearly separable. The solid lines in figure 1 and 2 show two different separating surfaces for the problem; the dashed lines parallel to these solid lines, show how much the separating surface can be moved before the closest point of

the training set will cause a misclassification. The distance from the separating surface to a dashed line, is called the margin and the data points of the training set that lie on the dashed lines, are called the support vectors of a SVM. The SVM problem is to find the separating surface that maximizes the margin between the data points of the training set. In case of the problem stated in figure 1 and 2, the separating surface of figure 2 which maximizes the margin, will therefore be found by the SVM method. How the SVM method exactly finds such a best separating surface, will be discussed in chapter 3, where we will have a detailed look on the mathematical foundation of SVM.

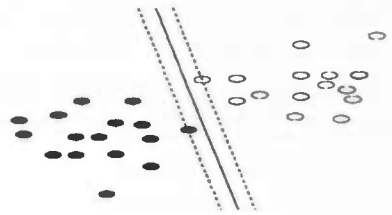

Figure 1: Separating surface (solid) of a SVM with a small margin

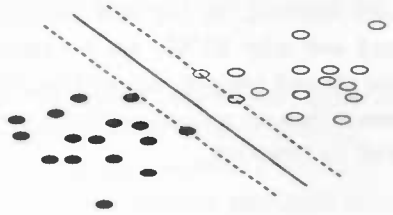

Figure 2: Separating surface (solid) of a SVM with the maximal margin

2 Choice for Support Vector Machines

In the last section of the previous chapter (1.5.6) several machine learning techniques were introduced, with which an automatic text classifier can be built. However, due to the limited amount of available time for this master thesis, not all these machine learning techniques could be researched extensively. For this master thesis we will therefore stick to just one machine learning technique, and do some extensive research on that one, instead of some shallow research on all of them. The question which one of the machine learning techniques this would be, was easily answered: SVM. In the rest of this chapter we will show why SVM was chosen and show that this decision is well-founded. In the first section we will take a look at the theory of SVM, and see why SVMs are so suitable for text categorization problems. In the second part of this section we will show, by means of some other, earlier researches on automatic text categorization, that SVMs indeed perform very well in practice.

2.1 Theory

We will start by describing what makes SVM so unique compared to other machine learning techniques. The important difference between SVM and other machine learning techniques is, that while other machine learning techniques try to optimize the performance on the training set (empirical risk minimization), SVM tries to minimize the probability of misclassifying yet-to-be-seen patterns for a fixed but unknown probability distribution of the data (structural risk minimization). [43][44]

One of most important properties of SVMs is their ability to learn independently of the dimensionality of the feature space. [21] This property makes SVMs extremely useful for automatic text categorization problems, because, like we saw earlier in 1.2, text categorization problems tend to have an extremely large feature space. Besides this property, it is known that SVMs are able to handle real sparse feature vectors very well. [24] This is definitely required for text categorization problems, because text documents are generally represented by a large feature vector of which only a few entries will be nonzero. Another nice property of SVM is that training a SVM is performed by maximizing a hill-shaped function, which means that a SVM will always find the best, unique answer of the problem in polynomial time.

When SVM was introduced in 1974, it was not a very popular machine learning technique, because of its tremendous processing power needs. But now that some extensions to and changes of the original algorithm have been applied, and the processing power of the pc's has increased, nothing stands in the way for SVM to flourish in the automatic text categorization field.

2.2 Practice

In the past decade a lot of research has been done on applying SVMs to text categorization problems. These researches have shown that SVM is one of the best and frequently even the best machine learning technique for automatic text categorization. In the following part of this section we will take a look at some quotes from the reports of these researches, to see what they have to say about applying SVMs to automatic text categorization problems.

- The following quote is a part from the conclusion of the paper from Thorsten Joachims [21] where SVM is introduced to text categorization for the first time.

  > "The experimental results show that SVMs consistently achieve good performance on categorization tasks, outperforming existing methods substantially and significantly. [...] Another advantage of SVMs over the conventional methods is their robustness. SVMs show good performance in all experiments avoiding catastrophic failure like observed for the conventional methods on some tasks."

- Yiming Yang and Xin Liu directly compared five well-known text categorization methods in a controlled study with thorough statistical significance analysis in [49]. They used improved versions of Naïve Bayes and k-nearest neighbours (compared to Joachims work) but still found that the SVM performed at least as well as all other classifiers they tested. They conclude their paper as follows:

  > "For the micro-level performance on pooled category assignments, both a sign test and an error-based proportion test suggest that SVM and kNN significantly outperform the other classifiers, while NB significantly underperforms all the other classifiers.
  >
  > With respect to the macro-level (category-level) performance analysis using F1, all the significance tests we conducted suggest that SVM, kNN and LLSF belong to the same class, significantly outperforming NB and NNet."

- Another part of a paper which reflects promising results for using SVM in text categorization is the following part of [42] from Georges Siölas and Florence d'Alche-Buc:

  > "SVM, with any metric, outperforms kNN and this confirms the supremacy of the induction principle used for SVM."

- Jason Rennie and Ryan Rifkin also reported that SVM significantly outperforms Naïve Bayes in their paper [38]:

  > "The SVM consistently outperforms the Naïve Bayes, but the difference in performance varies by matrix and amount of training data used. [...] We have shown that the Support Vector machine can perform multiclass text classification very effectively when used as part of an ECOC scheme. Its improved ability to perform binary classification gives it much lower error scores than Naïve Bayes."

- Ciya Liao, Shamim Alpha and Paul Dixon reported in their paper [26] that SVM is superior to the linearized neural network:

  > "We find that SVM algorithm is superior to the linearized neural network both in accuracy and training speed"

- Susan Dumais also came to the conclusion in her paper [13] that SVM outperforms every other machine learning technique:

  > "Linear SVMs were the most accurate method, averaging 91.3% for the ten most frequent categories and 85.5% over all 118 categories."

- Last but not least, Susan Dumais, John Platt and David Heckerman state in their paper that training a SVM and classifying new documents with a SVM is fast and also that SVM is one of the most accurate machine learning techniques of all:

  > "Find Similar is the fastest learning method (¡1 CPU sec/category) because there is no explicit error minimization. The linear SVM is the next fastest (< 2 CPU secs/category). [...] Support Vector Machines were the most accurate method, averaging 92% for the 10 most frequent categories and 87% over all 118 categories. [...] Both SVMs and Decision Trees produce very high overall classification accuracy, and are among the best known results for this test collection. [...] SVMs are the most accurate classifier and the fastest to train. [...] Linear SVMs are particularly promising since they are both very accurate and fast."

The promising theoretical foundation of SVMs for text categorization problems given in 2.1 and the encouraging results of the researches reported above, give good reason to restrict this research to SVM. In the following chapter we will extensively discuss every aspect of SVMs.

3 Support Vector Machines

After the quick introduction to Support Vector Machines in section 1.5.6, and the motivation for using SVMs in text categorization problems in chapter 2, it is now time to have a thorough look at this interesting looking SVM technique and discuss every aspect of it in detail. We saw in section 1.5.6 that training a SVM comes down to finding a hyperplane that best separates two classes. In the first section of this chapter we will see how we can find such a best separating hyperplane for simple two-class problems of which the classes can be linearly separated by a hyperplane. But because not all problems are linearly separable, the algorithm introduced in section 3.1 is extended so that linearly non-separable problems can be solved as well. In section 3.2 and 3.3 we will discuss two ways of extending the algorithm to the linearly non-separable case. The first extension is by introducing soft margin hyperplanes that allow, but penalize, examples that fall on the wrong side of the decision boundary (section 3.2). The second one is by mapping the original data vectors nonlinearly into a higher dimensional feature space in which the classes are linearly separable (section 3.3).

In the first three sections we will see the mathematical equation that needs to be solved in order to find a best separating hyperplane. We will also see that this mathematical equation is a quadratic programming problem and that training a SVM comes down to finding the unique solution of this quadratic programming problem. However, solving this quadratic programming problem is non-trivial and computationally expensive, and therefore we will need an efficient algorithm to solve it. In section 3.4 we will take a look at some of these efficient algorithms. We will see that training SVM by using these algorithms is sufficiently fast for practical use. There is only one problem left: most practical problems will not consist of just two categories. The SVM, as described in the first four sections, is just a binary classifier that can distinguish one category from another. In section 3.5 we will take a look at how we can solve multi-class classification problems with SVM.

3.1 Linear machines on linearly separable data

In this section we will start with describing how to find the best separating hyperplane for a simple two-class problem of which the classes are linearly separable (see [5] [33] [35] for more information).

Consider a problem consisting of a set S of n training examples $\vec{x_i} \in \mathbb{R}^d$ with $i = 1, \ldots, n$, where every training example is a feature vector consisting of values for d features. Each training example belongs to either of two categories and is given a label $y_i \in \{-1, +1\}$ to indicate to which one it belongs. Suppose we have some hyperplane that separates these examples

into two categories, leaving all examples with label +1 on one side, and all examples with label −1 on the other. A hyperplane can be given by the equation: $\vec{w} \cdot \vec{x} + b = 0$, where the weight vector $\vec{w} \in \mathbb{R}^d$ is a normal to the hyperplane, and $b \in \mathbb{R}$ is its bias (see figure 3). A separating hyperplane divides the training examples into two categories if there exist a $\vec{w}$ and b such that the following inequalities hold:

$$\begin{aligned} \vec{x_i} \cdot \vec{w} + b &\geq 0 \qquad \text{if } y_i = +1 \\ \vec{x_i} \cdot \vec{w} + b &\leq 0 \qquad \text{if } y_i = -1 \end{aligned}$$

in more compact notation, these two inequalities can be rewritten as:

$$y_i(\vec{x_i} \cdot \vec{w} + b) \geq 0 \tag{2}$$

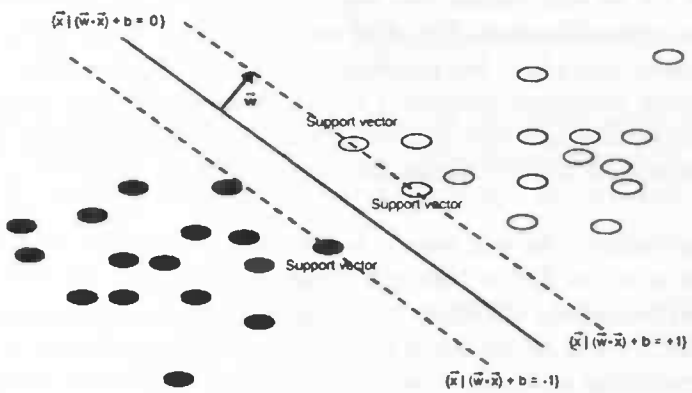

Figure 3: A separable classification problem, that is seperated by a OSH.

The goal of the support vector machine algorithm is to find the separating hyperplane with maximum distance to the closest training examples. The distance between a hyperplane and a training example x_i is given by:

$$d_i = \frac{y_i(\vec{w} \cdot \vec{x_i} + b)}{\|\vec{w}\|} \tag{3}$$

By scaling the variables $\vec{w}$ and b of a hyperplane, a different representation of the same hyperplane can be constructed, since $\{\vec{x} \mid (\vec{w} \cdot \vec{x}) + b = 0\}$ and $\{\vec{x} \mid (c\vec{w} \cdot \vec{x}) + cb = 0\}$ describe the same hyperplane, given that $c \neq 0$. In order to eliminate this scaling freedom and establish a one-to-one correspondence between separating hyperplanes and their parametric representation, we will use the canonical representation of a separating hyperplane [35].

Definition 3.1. Given a separating hyperplane $(\vec{w}, b)$ for the linearly separable set S, the canonical representation of the separating hyperplane is obtained by rescaling the pair $(\vec{w}, b)$ into the pair $(\vec{w'}, b')$ in such a way that the distance to the closest training example equals $\frac{1}{\|\vec{w'}\|}$.

In the rest of this thesis we will assume that a separating hyperplane is always given in its canonical representation and we will write $(\vec{w}, b)$ instead of $(\vec{w'}, b')$. Through this definition 3.1 and equation 3 for calculating the distance between a training example and a hyperplane, we can see that for the closest training example the following holds:

$$\begin{aligned} \frac{y_i(\vec{w} \cdot \vec{x_i} + b)}{\|\vec{w}\|} &= \frac{1}{\|\vec{w}\|} \\ y_i(\vec{w} \cdot \vec{x_i} + b) &= 1 \end{aligned}$$

This yields that equation 2 can be rewritten as:

$$y_i(\vec{x_i} \cdot \vec{w} + b) \geq 1 \tag{4}$$

We are a now in a position to describe the goal of the SVM algorithm more formally, starting by defining the notion of optimal separating hyperplane [35].

Definition 3.2. Given a linearly separable set S, the optimal separating hyperplane (OSH) is the separating hyperplane which maximizes the distance to the closest point of S.

Since the hyperplane is in its canonical representation, the distance to the closest training example that needs to be maximized equals $\frac{1}{\|\vec{w}\|}$. Therefore the OSH can be regarded as the solution of the problem of maximizing $\frac{1}{\|\vec{w}\|}$ subject to constraint 4. Knowing that maximizing $\frac{1}{\|\vec{w}\|}$ is the same as minimizing $\|\vec{w}\|$, minimizing $\|\vec{w}\|^2$ and minimizing $\frac{1}{2}\|\vec{w}\|^2$, the OSH can also be found by solving the problem of minimizing $\frac{1}{2}\|\vec{w}\|^2$. Solving this problem instead of the problem of maximizing $\frac{1}{\|\vec{w}\|}$, will make things much simpler later on.

Problem **P1**

$$\begin{aligned} \text{Minimize} \quad & \frac{1}{2}\|\vec{w}\|^2 \\ \text{Subject to} \quad & y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1, \quad i = 1, \ldots, n \end{aligned}$$

Two comments are in order. First, the parameter b enters in the constraints but not in the function to be minimized. Second, if the pair $(\vec{w}^*, b^*)$ solves **P1**, then for at least one $\vec{x_i} \in S$ we have $y_i(\vec{w}^* \cdot \vec{x_i} + b^*) = 1$, because of the canonical representation of the OSH. An interesting property of SVM is that the decision surface is determined only by the training examples for which hold that $y_i(\vec{w} \cdot \vec{x_i} + b) = 1$. Those points are called the support vectors, and are the only effective examples in the training set. If all examples, except for these support vectors, are removed from the training set, the SVM algorithm will still learn exactly the same decision function.

In order to solve this problem **P1**, we will switch to a Lagrangian formulation of the problem. There are two reasons for doing this. The first is that constraint 4 will be replaced by constraints on the Lagrangian multipliers themselves, which will be much easier to handle. The second is that in this reformulation of the problem, the training data will only appear (in the training and test algorithms) in the form of dot products between vectors. This is a crucial property which will allow us to generalize the procedure to the nonlinear case later on in section 3.3 (see [5]).

We will introduce n nonnegative Lagrange multipliers $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$, one for each of the inequality constraints (4). Finding the solution to problem **P1** now becomes equivalent to determining the *saddle point* of the function:

$$L_p = \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\vec{w} \cdot \vec{x_i} + b) - 1) \tag{5}$$

With $L_p = L_p(\vec{w}, b, \vec{\alpha})$, we can determine the saddle point of this function by determining the point where L_p has a minimum for $\vec{w}$ and b, and a maximum for $\vec{\alpha}$. In this point the derivatives are equal to zero and we can therefore write:

$$\frac{\partial L_p}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{6}$$

and

$$\frac{\partial L_p}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^{n} \alpha_i y_i \vec{x_i} = 0 \tag{7}$$

with

$$\frac{\partial L_p}{\partial \vec{w}} = (\frac{\partial L_p}{\partial w_1}, \ldots, \frac{\partial L_p}{\partial w_i})$$

By substituting equations 6 and 7 into equation 5 we can see that problem **P1** reduces to the maximization of the function (see Appendix A for a full elaboration):

$$L_d = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\vec{x_i} \cdot \vec{x_j}) \tag{8}$$

subject to constraint 6 and $\alpha_i \geq 0$ for $i = 1, \ldots, n$. This new problem is called *dual problem* an can be formulated as (see Appendix B for a full elaboration):

Problem **P2**

$$\begin{aligned} \text{Maximize} \quad & -\frac{1}{2}\vec{\alpha} \cdot D\vec{\alpha} + \sum_{i=1}^{n} \alpha_i \\ \text{Subject to} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0 \\ & \alpha_i \geq 0, \qquad i = 1, \ldots, n \end{aligned}$$

Where D is a $n \times n$ matrix that is solely built from the training examples and their labels, such that each entry $D_{ij} = y_i y_j (\vec{x_i} \cdot \vec{x_j})$.

How this quadratic programming problem **P2** can be solved will be discussed in section 3.4, but, assuming that the outcome of the problem is given by $\vec{\alpha}^*$, we will take a look at how these nonnegative Lagrange multipliers can be used to calculate the $\vec{w}^*$ and b^* of the OSH. From equation 7 it follows that the weight vector $\vec{w}^*$ can simply be determined by:

$$\vec{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \vec{x_i}$$

while b^* can be determined from the Kuhn-Tucker conditions [47]

$$a_i^* (y_i (\vec{w}^* \cdot \vec{x_i} + b^*) - 1) = 0, \qquad i = 1, \ldots, n \tag{9}$$

Note that in order to make the left hand side of equation 9 equal to zero, either α_i^* or $y_i(\vec{w}^* \cdot \vec{x_i} + b^*) - 1$ needs to be zero. Therefore the only α_i^* that can be nonzero, are those for which hold that:

$$\begin{aligned} y_i(\vec{w}^* \cdot \vec{x_i} + b^*) - 1 &= 0 \Rightarrow \\ y_i(\vec{w}^* \cdot \vec{x_i} + b^*) &= 1 \Rightarrow \\ (\vec{w}^* \cdot \vec{x_i} + b^*) &= \frac{1}{y_i} \Rightarrow \qquad (10) \\ (\vec{w}^* \cdot \vec{x_i} + b^*) &= y_i \qquad (11) \end{aligned}$$

Equation 10 is equal to 11 because $y_i \in \{-1, +1\}$. We saw earlier in this section that equation 11 only holds for the training examples of S closest to the OSH and that these training examples are called support vectors (see figure 3). Every training example has a Lagrange multiplier α_i^*, but only those corresponding to the support vectors can thus be nonzero. This emphasizes once again that the support vectors are the only training examples that matter to the SVM algorithm.

Given such a support vector $\vec{x}_{sv}$ and its label y_{sv}, the variable b^* of the OSH can be obtained from the corresponding Kuhn-Tucker condition as:

$$b^* = y_{sv} - \vec{w}^* \cdot \vec{x}_{sv}$$

A new, unseen data point $\vec{x}$ can now simply be classified by computing:

$$sign(\vec{w}^* \cdot \vec{x} + b^*)$$

3.2 Linear machines on linearly non-separable data

Knowing that a lot of problems are not linearly separable, makes searching for an OSH, like described in the previous section, meaningless in such problems, because there will not be a separating hyperplane. An extension to the simple algorithm was therefore required, in order to solve the non separable cases as well. Cortes and Vapnik [11] came up with the idea to allow, but penalize training examples that fall on the wrong side of an OSH, by introducing n nonnegative slack variables $\vec{\xi} = (\xi_1, \ldots, \xi_n)$. (see figure 4)

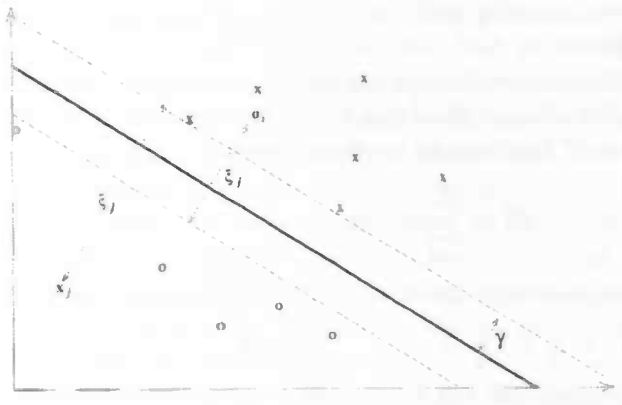

Figure 4: Introducing slack variables $\vec{\xi} = (\xi_1, \ldots, \xi_n)$ to the classification problem

With the use of these slack variables, equation 4 is extended to:

$$y_i(\vec{x_i} \cdot \vec{w} + b) \geq 1 - \xi_i \tag{12}$$

Note that a slack variable ξ_i is equal to zero if its corresponding training example $\vec{x_i}$ falls on the right side of the OSH, whereby inequality 4 is satisfied. This generalized OSH can now be found by solving the extended version of problem **P1**:

Problem **P3**

$$\begin{aligned} \text{Minimize} \quad & \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{Subject to} \quad & y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n \\ & \vec{\xi} \geq 0 \end{aligned}$$

Where the term $C\sum_{i=1}^{n}\xi_i$ is some sort of measure of the amount of misclassification, with C as its regularization parameter. For a large C, the OSH tends to minimize the number of misclassifications, while for a small C the minimum distance to the closest example is maximized. For intermediate values of C the solution of problem **P3** trades errors for a larger

margin. Note that the term $C\sum_{i=1}^{n}\xi_i$, and not the intuitively more appealing quadratic variant $C\sum_{i=1}^{n}\xi_i^2$, is chosen for error measurement. This is done to construct a statistically robust classifier that is not very sensitive to the presence of outliers in the training set.

Similar to the transformation of problem **P1** in the separable case, problem **P3** can be transformed into the *dual*:

Problem **P4**

$$\text{Maximize} \quad -\frac{1}{2}\vec{\alpha}\cdot D\vec{\alpha} + \sum_{i=1}^{n}\alpha_i$$

$$\text{Subject to} \quad \sum_{i=1}^{n} y_i\alpha_i = 0$$

$$0 \le \alpha_i \le C, \qquad i = 1,\ldots,n$$

where D is the same $n \times n$ matrix as in problem **P2** in the separable case with each entry $D_{ij} = y_i y_j(\vec{x_i}\cdot\vec{x_j})$. The weight vector $\vec{w}^*$ can again be determined by:

$$\vec{w}^* = \sum_{i=1}^{n}\alpha_i^* y_i \vec{x_i} \tag{13}$$

while b^* can again be determined from the solution of the dual problem **P4**, $\vec{\alpha}^*$, and the new Kuhn-Tucker conditions:

$$\alpha_i^*(y_i(\vec{w}^*\cdot\vec{x_i}+b^*) - 1 + \xi_i^*) = 0, \qquad i = 1,\ldots,n \tag{14}$$

$$(C-\alpha_i^*)\xi_i^* = 0 \qquad i = 1,\ldots,n \tag{15}$$

where the ξ_i^* are the the values of the ξ_i at the saddle point. Similar to the separable case, the training examples $\vec{x_i}$ for which $\alpha_i^* \ge 0$ are termed support vectors. The difference is that there are two types of support vectors: those for which $\alpha_i^* < C$ and those for which $\alpha_i^* = C$. From 15 it follows that for the support vectors of the first type $\xi_i^* = 0$, and hence from 14 that the distance from the OSH to these points equals $\frac{1}{\|\vec{w}^*\|}$. These support vectors are termed *margin vectors*. The support vectors of the other type, where $\alpha_i^* = C$, are are designated as errors. The value for ξ_i^* determines what kind of error it is. $\xi_i^* > 1$ means that the points are misclassified and $0 < \xi_i^* \le 1$ means that the points are correctly classified, but the distance from the OSH to the points is smaller than $\frac{1}{\|\vec{w}^*\|}$. In some degenerate cases α_i^* can even be zero, what will mean that these points will lie on the margin.

The variable b^* of the OSH can be determined from the Kuhn-Tucker condition 14 in a similar way as in the separable case. Given a margin vector

$\vec{x}_{mv}$ and its label y_{mv}, for which thus holds that $\xi^*_{mv} = 0$ and $0 < \alpha^*_{mv} < C$, variable b^* can be determined as:

$$b^* = y_{mv} - \vec{w}^* \cdot \vec{x}_{mv}$$

A new, unseen data point $\vec{x}$ can still simply be classified by computing:

$$sign(\vec{w}^* \cdot \vec{x} + b^*) \tag{16}$$

3.3 Nonlinear machines

Wanting to separate a set of examples by a linear OSH (with or without using slack variables) will in many cases however be to restricted. A lot of cases can not be separated by a linear OSH, but probably can be separated by, for example a higher polynomial or radial function. Fortunately, the SVM theory can easily be extended to nonlinear separating surfaces by mapping the input data points nonlinearly into some other feature space and looking for a linear OSH in that feature space [11]. All the considerations of the previous sections hold, since we are still doing a linear separation, but in a different space. The mapping can be done by a mapping function $\Phi : \mathbb{R}^d \mapsto \mathcal{H}$, that maps a data point $x \in \mathbb{R}^d$ into some other feature space $\mathcal{H}$ (typically a Hilbert space of finite or infinite dimension [46]) (see figure 5).

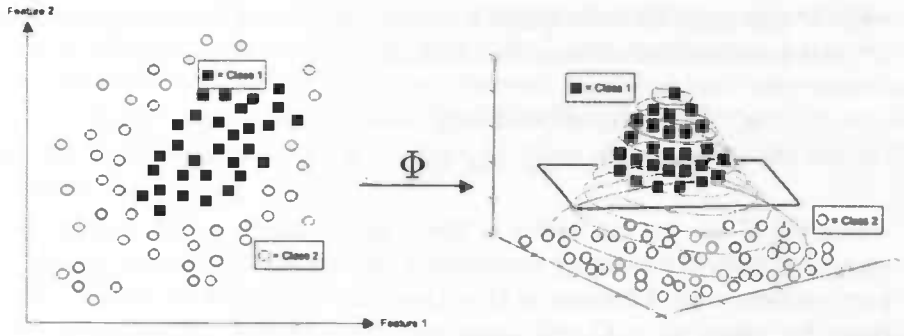

Figure 5: Map the data nonlinearly into a higher-dimensional feature space via Φ and construct a linear seperating hyperplane there.

Such a mapping function Φ can however be very unpractical to work with, due to its (possibly infinitely) large number of components Φ_i. Luckily it follows from the equations that are needed to find an OSH (problem **P4**, equation 13) and classify new data points (equation 16), that the data only appears in the form of dot products between data points:

$$D_{ij} = y_i y_j \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$$

and

$$\vec{w}^* \cdot \Phi(\vec{x}) + b^* = \sum_{i=1}^{n} \alpha_i^* y_i \Phi(\vec{x_i}) \Phi(\vec{x}) + b^*$$

This dot product property makes it possible to make use of the rather old *kernel* trick [2], that will simplify the problem significantly. A kernel function $K(\vec{x_i}, \vec{x_j}) = \Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$ makes full knowledge of Φ unnecessary and can reduce computation time enormously. Take for example the rather simple example where the data points are in $\mathbb{R}^2$ and the polynomial kernel $K(\vec{x_i}, \vec{x_j}) = (1 + \vec{x_i} \cdot \vec{x_j})^2$ is used. A space $\mathcal{H}$, and a mapping Φ from $\mathbb{R}^2$ to $\mathcal{H}$ can then be found such that $(1 + \vec{x_i} \cdot \vec{x_j})^2 = \Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$. Given that $\vec{x} = (x_1, x_2)$ we can write for this example that:

$$\Phi(\vec{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$$

where $\mathcal{H} = \mathbb{R}^6$. It is easy to see that working with this kernel is much easier and faster than working with the mapping function. Imagine the advantage of using the gaussian radial base function $K(\vec{x_i}, \vec{x_j}) = e^{-\gamma \|\vec{x_i} - \vec{x_j}\|^2}$, instead of the dot product notation between mapping functions where $\mathcal{H}$ is infinite dimensional.

Extending the SVM theory to the nonlinear case is now reduced to finding kernels which identify certain families of decision surfaces and can be written as $K(\vec{x_i}, \vec{x_j}) = \Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$. A useful criterion for deciding whether a kernel can be written in such a way, is given by Mercer's theorem [11][12]. Some useful kernel functions are [15]:

Kernel function	**Kernel name**
$K(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j}$	Linear
$K(\vec{x_i}, \vec{x_j}) = (\gamma(\vec{x_i} \cdot \vec{x_j}) + r)^d$	Polynomial
$K(\vec{x_i}, \vec{x_j}) = \tanh(\gamma(\vec{x_i} \cdot \vec{x_j}) + r)$	Sigmoid
$K(\vec{x_i}, \vec{x_j}) = e^{-\gamma \|\vec{x_i} - \vec{x_j}\|^2}$, $\gamma > 0$	Gaussian radial basis function

Finally we can state that the SVM algorithm comes down to finding a separating surface by solving problem **P4** with $D_{ij} = y_i y_j K(\vec{x_i}, \vec{x_j})$. A new, unseen data point $\vec{x}$ can be classified by computing:

$$sign(\sum_{i=1}^{n} \alpha_i^* y_i K(\vec{x_i}, \vec{x}) + b^*) \tag{17}$$

3.4 Solving the quadratic programming problem

In the previous sections, we extensively studied the mathematical foundation of SVM, and came to the conclusion that training a SVM actually comes down to finding the unique solution $\vec{\alpha}^*$ of problem **P4** with

$D_{ij} = y_i y_j K(x_i, x_j)$. In this section we will now take a look at how we can solve this quadratic programming problem efficiently. This problem is called quadratic programming, because the function to be maximized depends on the α_i^* quadratically, while α_i^* only appears linearly in the constraints. Conceptually this problem is about finding the top of a hill-shaped function. The search for this maximum is constrained to lie within a cube and on a plane. The search, though, will normally occur in a high-dimensional space, so that the hill-shaped function will be high-dimensional, the cube will be a hypercube and the plane will be a hyperplane.

Solving this problem looks rather simple, but due to the quadratic form of the matrix D, it can become extremely difficult to compute for large real world problems. Take for example a problem with a training set of 50000 examples. The matrix D of this problem will consist of 2.5 billion elements, and will need a memory of approximately 20 gigabytes to store it in (using a 8-byte floating point representation). A alternative to storing this matrix in memory would be to recompute the elements of D every time it is needed. Either way, an efficient algorithm will be needed to solve such large scale problems within reasonable time.

Vapnik [43] made the observation that the same values for α_i would be found if all training examples, except for the support vectors, would be removed from the training set. If only the support vectors were known beforehand, a matrix D of problems with a small number of support vectors, could be heavily reduced. All rows and columns from D that correspond to a training example with $\alpha_i = 0$ could then be removed, while the ones corresponding to the support vectors are kept. In the following part of this section we will discuss four SVM training algorithms. All these algorithms make use of the observation made by Vapnik.

Chunking

Vapnik introduced an algorithm himself, that is known as *chunking*. This algorithm decomposes the large QP problem into a series of smaller QP problems, with the ultimate goal to identify all of the nonzero Lagrange multipliers and discard all the zero ones. At every step, for some value of M, chunking solves a QP problem that consists of the following examples: every nonzero Lagrange multiplier from the last step, and the M worst examples that violate the Karush-Kuhn-Tucker (KKT) conditions:

$$\begin{aligned} \alpha_i = 0 &\Leftrightarrow y_i u_i \geq 1, \\ 0 < \alpha_i < C &\Leftrightarrow y_i u_i = 1, \\ \alpha_i = C &\Leftrightarrow y_i u_i \leq 1 \end{aligned}$$

where

$$u_i = \sum_{j=1}^{n} \alpha_j^* y_j K(\vec{x_j}, \vec{x_i}) + b^*$$

is the output of the nonlinear SVM for the ith training example. If there are fewer than M examples that violate the KKT conditions at a step, all violating examples are added in. Each QP sub-problem is initialized with the results of the previous sub-problem. At the last step the entire set of nonzero Lagrangian multipliers has been identified, hence the last step solves the large QP problem. Chunking reduces the size of the matrix D from the number of training examples squared, to approximately the number of support vectors squared.

Osuna's algorithm

Chunking still requires a lot of memory when the number of support vectors is large. Osuna, et al. [32] came up with an idea that makes use of a constant size matrix, and will therefore allow training on arbitrarily sized data sets. They proved that a large QP problem can be broken down into a series of smaller QP sub-problems. The suggested method makes use of only a subset of the vectors as a working subset, and optimizes on the corresponding α_i's, while freezing the others.

SVM$^{\text{light}}$

Joachims [22] used this decomposition idea of Osuna, et al. in his SVM$^{\text{light}}$ algorithm, but made it much more efficient and faster. Joachims added an efficient and effective method for selecting the working set, and implemented some computational improvements like caching. In his paper Joachims showed that his SVM$^{\text{light}}$ algorithm is substantially faster than the conventional chunking algorithm.

Sequential Minimal Optimization

A new algorithm was introduced by Platt [34] in 1998. His algorithm, named Sequential Minimal Optimization (SMO), puts the subset selection in Osuna's, et al. algorithm to the extreme, by iteratively selecting a subset of only two training examples at every step. In a specific step these chosen two α_i's are then jointly optimized, and the SVM is updated with its values. SMO can solve for two α_i's analytically, thereby avoiding numerical QP optimization entirely. This makes that a sub-problem can be solved so fast, that even though more optimization sub-problems need to be solved, the overall problem can be solved quickly. Research [22] has shown that SMO is approximately equally fast as SVM$^{\text{light}}$, and therefore outperforms the chunking algorithm substantially as well.

3.5 Multiclass machines

In the previous sections of this chapter we saw how we can construct a SVM that is able to separate two different classes from each other (such SVMs are called binary SVMs). Most real world problems will however, consist of a lot more than just two classes, and will therefore require a method that can make a distinction between a number of classes. Luckily, we can easily extend the SVM approach to these multi-class cases, by combining several of the binary SVMs. In this section we will discuss three of the most used methods that extend the SVM approach to the multi-class case: *one-against-all*, *one-against-one* and *DAGSVM* (see [19]).

One-against-all

In order to solve a k class classification task, the one-against-all method constructs k binary SVMs (one for each class), whereby each binary SVM will be trained to make a distinction between one of the classes and the rest of the classes together. The lth binary SVM will be trained with a set of training examples of class l with positive labels on one side, and the rest of the training examples with negative labels on the other side. A new, unseen data point can then be classified into one of the k classes, by using the *winner-takes-all* scheme, whereby a point is assigned to the class l of the SVM that has the largest value of the decision function:

$$\sum_{i=1}^{n} \alpha_{l_i}^* y_i K(\vec{x_i}, \vec{x}) + b^*$$

This is just equation 17, but without the $sign()$ part to make the binary classification. Point $\vec{x}$ is now assigned to the class of the SVM which is most confident that data point $\vec{x}$ belongs to its class.

One-against-one

Another way of combining several binary SVMs, in order to solve multi-class cases, is by using the one-against-one method. Unlike the one-against-all method, this method constructs not k, but $k(k-1)/2$ binary SVMs (one for each combination of two different classes). Every one of these $k(k-1)/2$ binary SVMs will be trained to make a distinction between two classes, by training them with only the training examples of these two classes. When all these classifiers are constructed, the *max wins* strategy can be used to classify new data points. This *max wins* strategy combines the outcomes of the binary SVMs by counting the number of times each class wins, and assigning a new data point to the class that won most. In case that two or more classes have won an equal amount of times, one of these classes is selected at random. At first it may look like it will take a lot more time

to train a multi-class SVM with this one-against-one method then with the one-against-all method, because of the extra amount of binary SVMs that have to built ($k(k-1)/2$ instead of k), but this does not necessarily have to be true. The one-against-one method can sometimes even be much faster. This follows from the fact that each binary SVM will only have to be trained with a small part of the total amount of training examples. Take for example a multi-class problem that consists of k classes, and has a total of n training examples. Each class will then have an average amount of n/k training examples at its disposal. The one-against-one method will then have to compute $k(k-1)/2$ quadratic programming problems, where each problem has about $2n/k$ variables, while the one-against-all method has to solve k quadratic programming problems with n variables. Which of these two methods will be faster in practice, will depend on the amount of classes and training examples of a specific problem, and the distribution of these examples among the classes.

DAGSVM

The third method for multi-class SVM construction that we will discuss, is the *Directed Acyclic Graph Support Vector Machines*, or DAGSVM for short. DAGSVM has exactly the same training phase as the one-against-one method, wherein the same $k(k-1)/2$ binary SVMs (one for each combination of two different classes) are constructed, but has a different testing phase. Instead of using the *max wins* strategy to classify new data points, DAGSVM uses of a rooted binary directed acyclic graph with $k(k-1)/2$ internal nodes and k leaves. Each node of such a graph represents one of the $k(k-1)/2$ binary SVMs and each leave represents one of the classes. At each node the outcome of the binary SVM determines which way to walk down: left or right. A new document can then be classified by starting at the root of the graph and walk down the graph until one of the leaves is reached. The class of this leaf will then indicate the predicted class of the given document.

The advantage of DAGSVM over one-against-one is that only $k-1$ instead of all the $k(k-1)/2$ binary SVMs will have to be computed to classify a new document. This will make the testing phase of the DAGSVM method much faster than the one-against-one method. The downside of this method is that it will not come up with probability estimates for all the different classes, but will only find the winning class.

4 Research plan

Now that we have extensively discussed the theory of automatic text classification and SVMs in chapter 1 through 3, we are now ready to set up a research with which we can give an answer to the main research question. In the first section of this chapter we will repeat the objective of this research and the research questions as they were given in the introduction of this thesis. In the second section we will take a look at the restrictions of this research. And finally, in section three, we will set up some experiments that we will carry out in this research, in order to give answer to the research questions.

4.1 Research objective and research questions

In the introduction of this thesis, we introduced the objective of this research as follows:

Making a contribution to solving large-scale and complex real world text categorization problems.

In order to make this contribution, I will do a research on the design process of an automatic text classifier for such a large-scale and complex real world text categorization problem, and make clear what choices can best be made during this design process, to maximize the performance of the classifier. This way I will make a guide, how one can best construct an automatic text classifier for a large-scale and complex real world text categorization problem. The main research question of this thesis that will be answered will therefore be:

What choices can best be made in the design process of an automatic text classifier for large-scale and complex real world text categorization problems?

To give answer to this research question we can divide it into a couple of smaller research questions. In chapter 1 through 3 we read all about automatic text categorization and SVMs. Therefore we now know what choices can be made in the design process of an automatic text classifier, and we are ready to carry out the research, to see which of these choices can best be made. The research questions with their possible answers are:

1. Do we want to extract the stopwords?
   - Yes or no?
   - see 1.1.1
2. Do we want to use stemming?

- Yes or no?
- see 1.1.2

3. Do we want to fix class imbalance?
   - Leave imbalanced or make balanced?
   - see 1.1.3

4. What kind of features do we want to represent a document with?
   - BOW, word n-grams, phrasal features or letter n-grams?
   - see 1.2

5. How do we want to represent the importance of a feature for a certain document?
   - Boolean weighting, frequency weighting, tf-idf weighting, tfc weighting, ltc weighting or entropy weighting?
   - see 1.3

6. In what way do we want to reduce the dimensionality of the feature vector?
   - Feature selection: Document frequency, mutual information, information gain, chi-square, correlation coefficient, relevancy score, odds ratio or feature extraction: term clustering, latent semantic indexing?
   - see 1.4

7. What machine learning algorithm do we want to use?
   - Rocchio's algorithm, Linear least squares fit, Naïve Bayes, Neural network, K-Nearest Neighbor, Support Vector Machines?
   - see 1.5

As for the last research question (7), we have showed in chapter 2 that, nowadays, SVM is the best choice for automatic text categorization, so we decided to focus this research only on the SVM machine learning algorithm. Subsequently, we extensively discussed SVMs in chapter 3. This resulted in some more questions in the design process of an automatic text classifier for large-scale and complex real world text categorization problems. Therefore we will add the following research questions to the questions that have to be answered for sake of this research:

8. What kernel do we want to use?
   - Linear, radial, polynomial or sigmoid?
   - see 3.3

9. What value do we want to use for parameter C and for the kernel parameters?
   - see 3.2 and 3.3

10. What SVM solving algorithm do we want to use?
    - Chunking, Osuna's algorithm, SVM^{light} or Sequential Minimal Optimization?
    - see 3.4

11. What multiclass SVM method do we want to use?
    - one-against-all, one-against-one or DAGSVM?
    - see 3.5

Each of these eleven research questions have to be answered, in order to give answer to the main research question, and to reach the objective of this research.

4.2 Restrictions

Due to the limited amount of time that is available for this research, it is unfortunately not feasible to perform a full experiment on each and every one of the research questions. Choices have to be made about which research questions will be fully examined and which will not. Some of the research questions will be answered with a full experiment on all of its choices, while the answers of some other research questions will just be based on assumptions, or researches done by others.

We will not experiment with Research question 2, about stemming the words in a document. Stemming probably will not improve and probably even worsen the performance of the automatic text classifier, because the documents in the 2ehands.nl problem will contain a lot of spelling mistakes and short catchwords. So, we will not use stemming of words in the rest of the research.

We also will not pay any attention to the different features with which documents can be represented (research question 4). We will only use the most used and widely accepted feature representation technique: Bag Of Words. We will only use this technique, because there just is not enough time to experiment with all the other different feature representation techniques, although some of them look interesting. Where the linguistic approach still is not very useful in practice, the word n-grams and especially the letter n-grams look interesting, but have to be examined in further research.

In research question 6 we will only experiment with some of the feature selection techniques and not with the feature extraction techniques, because the feature selection techniques are easier to implement, and will probably have more or less the same effect as the feature extraction techniques. Feature selection will therefore give us the desired insight into the influence of

dimensionality reduction on the performance of a classifier. In this experiment we will examine the following feature selection techniques: document frequency, mutual information, information gain, chi-square and odds ratio.

We also will not experiment with the different SVM learning algorithms, because implementing these algorithms is extremely difficult and takes a lot of time. Instead, we will use a software package that already contains a SVM learning algorithm, to construct our SVMs. We experimented a little with three different software packages: *LIBSVM* [8], Bow [28] and SVMTorch [10], and came to the conclusion that *LIBSVM* is the fastest, and most useful for our research. This is why we will use *LIBSVM* in our research. *LIBSVM* uses a simplification of both SMO and SVM^{light} to construct a SVM.

This leaves us with the following seven research questions: 1, 3, 5, 6, 8, 9 and 11, that we will examine in this research.

Besides these restrictions on the research questions, we will also place a restriction on the number of categories of the problem. We will only use the main categories of the problem of 2ehands.nl, and not also its subcategories, because it is hard enough to classify the documents into one of 144 categories, let alone that we would have to classify them into around 3000 categories.

4.3 Research

In this section we will see how the answers to these research questions, mentioned in the previous section, will be found.

We will set up an experiment for every one of the remaining research questions, except for research question 8 and 9, which will be combined into one experiment. We will therefore set up 6 experiments with which we will give answer to the main research question. The research questions will not be discussed in the order they were presented in, but in a way that is logical for the research. In this research we will first construct a very basic classifier, and subsequently carry out all the experiments one by one, to see what choices can best be made for every research question. Each experiment will begin with the best results of the previous experiments, so that with each experiment the classifier will improve, resulting in the best possible classifier at the final experiment. The first four experiments will involve improving the performance of the classifier while the last two are mainly intended to improve the classification speed of the classifier. This way we can check if we can improve the classification speed of the classifier without reducing its performance.

These experiments will be carried out on AMD Athlon™ XP 2800+ with 512 KB cache and 512 MB of memory. The SVM will be constructed with *LIBSVM* [8] and the performance of a SVM will be measured by the classification rate (number of correctly classified examples divided by the total number of examples) and not by precision and recall, because these are not useful in our case where the classifier comes up with only one category as a winner for each example.

LIBSVM consists of two main programs: *svm-train* for constructing SVMs and *svm-predict* for testing SVMs. *svm-train* constructs a SVM from a given datafile. Such a datafile is the BOW representation of a set of example documents where each line represents one example document and is represented in the following way:

`<class_label_id> <word_id_1>:<value_1> ... <word_id_n>:<value_n>`

where `<class_label_id>` is the id of the class label of the document, `<word_id_i>` is the id of word i in the vocabulary and `<value_i>` is the value that represents the importance of word i for this document. *svm-predict* outputs the class predictions of a SVM on a set of test documents, by giving *svm-predict* the model (constructed with *svm-train*) of the SVM and the datafile of the testset.

Some homemade Perl scripts will first set up the vocabulary of a given set of training documents, and subsequently convert the trainset and the testset into datafiles according to this vocabulary.

We will now describe the experiments that we will carry out in this research, in order to give answer to the research questions.

Experiment 1 - Kernel and parameters (research questions 8 & 9)

Like we read earlier, we will first construct a basic SVM for the text categorization problem of 2ehands.nl. This means that we construct a classifier with the following design choices: no stopword extraction, balanced data, frequency weighting, no dimensionality reduction and the one-against-one multiclass SVM method.

In the first experiment we will figure out which kernel and parameters for the SVM we can used best, in order to solve large-scale and complex real world text categorization problems. Chih-Wei Hsu, Chih-Chung Chang and Chih-Jen Lin suggested in [9] that the radial kernel is a reasonable first choice, because this kernel, unlike the linear kernel, can handle cases where the relation between class labels and features is nonlinear. The radial kernel also has less numerical difficulties because for the radial kernel holds that

$0 < K_{ij} \leq 1$, while for the polynomial kernel these values may go to infinity or zero. Moreover, the sigmoid kernel is not valid (i.e. not the inner product of two vectors) under some parameters. This is why we will use the radial kernel in this experiment, and try to find the best parameters for it. We saw in 3.3 that the radial kernel makes use of one parameter, namely: γ. Besides this parameter we also need to fine-tune the C parameter of the SVM. In this experiment we therefore will have to find the C and γ combination with which the SVM performs best.

This best combination can be determined by means of a so called grid search. Given a set of possible values for the C and γ parameters, a grid search determines the best C and γ combination, by trying out all the given combinations, and selecting the combination that gives the best performance. In order to make the performance of the classifier more reliable, training and testing of a SVM with a C and γ, is usually done by means of n-fold cross-validation. n-fold cross-validation means that, given a set of examples, this set will be divided into n equal sized parts, of which $n-1$ parts will be used as the trainset to train the SVM with. The remaining part will be used as the testset to determine the classification rate of the classifier with. This process will be repeated n times, each time with a different part of the given set of examples as the testset, and the remaining parts as the trainset. The performance of a classifier with a particular combination of C and γ is then calculated as the average over the n classification rates.

The difficulty with this way of determining the best C and γ combination, is the enormous amount of time it can take. Take for example the following scenario: we want to the determine the best C and γ combination out of 12 different C values and 11 different γ values. This comes down to a total of 132 different combinations. When we use 5-fold cross-validation, we need to train and test $5 \times 132 = 660$ different SVMs. When the problem is large, and training a single SVM therefore takes several hours, it would take months to determine the best C and γ combination!

For large problems, like the ones we are dealing with in this thesis, it is therefore not feasible to determine the best C and γ combination, in this way. In this experiment we will therefore figure out if there is some sort of relation between the number of categories, the number of training examples and the parameters C and γ. If there is such a relation, we can easily determine the best C and γ combination for a large problem, by using a smaller subset of this problem. This would drastically reduce the time to determine the best C and γ combination, because the traintime of a SVM is quadratic with the number of training examples

To determine if there exists such a relation, we will carry out the following

experiment: determine the best C and γ combination for several subproblems of the large problem, where each subproblem has a different number of examples and categories.

In this experiment we will use subproblems with a number of 10 and 20 categories and thereby use the following number of examples per category: 50, 100, 200, 400, 800 and 1600. This makes a total of twelve subproblems. In order to determine the best C and γ combinations, we will use the program *grid.py* that comes with *LIBSVM*. Given a range for C and γ, a step value for both parameters and a datafile of all the examples of a problem, *grid.py* trains and tests a SVM, by means of 5-fold cross-validation, one by one for each C and γ combination, and plots a graph of the classification rates of the problem for all the different C and γ combinations. In this experiment we will use 12 different C values, where $C = 2^0, 2^2, \ldots, 2^{20}, 2^{22}$, and 11 different γ values, where $\gamma = 2^{-23}, 2^{-21}, \ldots, 2^{-5}, 2^{-3}$, making a total of 132 combinations for each subproblem.

The outcome of this experiment will show if there is some relation between the number of categories, the number of training examples and the parameters C and γ.

Experiment 2 - Class imbalance (research question 3)

Now that we know which kernel and parameters to use, we will set up an experiment with which we will determine the difference between training a SVM with imbalanced data, and training a SVM with balanced data.

In order to determine this difference, we will set up the following experiment: we will construct a datafile of an imbalanced set of examples, and one of a set of balanced examples, while making sure that the total number of examples in both sets is equal. Besides that, we will use the same examples in both sets as much as possible, so that the difference between the SVM, caused by training with different examples, will be reduced to a minimum. In order to get a better picture of the results, we will carry out this experiment on a number of different sets of examples. These different sets of examples will differ in the total number of examples per set. We will use datasets of the following sizes: 10000, 20000, 40000, 80000 and 160000.

We will no longer test the constructed SVMs by means of n-fold cross-validation, but we will generate a single (verified) testset, with which we will test all the SVMs from now on. The documents in this testset will have to be classified correctly, in order to see if the classifier really classifies these documents into the correct categories. Like we saw earlier in the introduction of this thesis, some of the documents that we use to train the

SVMs with are falsely classified, because people have been mislabeling these documents. When we use these possibly falsely labeled documents to test a SVM with, it is clear that no value can be attached to the classification rate of the SVM, because we can not know if the classifier is right or wrong. Therefore we will need to manually check if every document in this testset is classified correctly, and correct the class labels of the documents that are not. The examples in the testset will have an imbalanced distribution over the categories, in order to reflect the imbalanced distribution of the world where the final classifier is going to be used in.

Experiment 3 - Stopword extraction (research question 1)

In this experiment we will take a look at the influence of stopword extraction on the performance of a classifier.

For this experiment we will simply construct two classifiers by using two different datafiles of the same set of train examples: one with stopword extraction and one without stopword extraction. Generating a datafile with the use of stopword extraction can simply be done by using a list of stopwords to remove certain stopwords from the vocabulary, before generating the datafile. We will remove the stopwords by using a list of stopwords consisting of 99 stopwords. Testing the two classifiers will be done with the same verified testset, we mentioned in experiment 2. We will also use the results from the previous two experiments to construct the best possible classifier.

Experiment 4 - Different feature weighting (research question 5)

Until now we were using the standard frequency weighting method to reflect the importance of a word for a document. We did this by just counting the number of times a word occurred in a document. In this experiment we will take a look at the other feature weighting methods, and see if they improve the performance of a classifier. During this experiment, we will use the same set of documents to train the SVMs, and the same set of documents to test them, while we vary the feature weighting methods to create the datafiles of these sets. The other feature weighting methods that will be examined are: Boolean weighting, tf-idf weighting, tfc weighting, ltc weighting and entropy weighting.

We will make a script for each feature weighting method that will not only construct a vocabulary of the trainset, but will also calculate, and store the constant parts of the feature weighting formulas that can be calculated beforehand, for each word of the vocabulary. For example, we will calculate and store the following part of the tf-idf formula (see 1.3) for each word of the vocabulary: $\log\left(\frac{n}{df_j}\right)$. We will also make a script for each feature weighting

method that will construct a datafile of a set of documents according to its feature weighting method, while using these stored parts. Knowing that in a datafile a document is represented as follows:
`<class_label_id> <word_id_1>:<value_1> ... <word_id_n>:<value_n>`
each script will generate the same datafiles, except for the values `<value_i>`. These values will differ according to the feature weighting formulas as stated in 1.3.

Experiment 5 - Dimensionality reduction (research question 6)

In this experiment we will examine a few different dimensionality reduction techniques to see if we can reduce the dimensionality of the feature vector, without reducing the classification rate of the classifier. The goal of this dimensionality reduction is to improve the classification speed of classifier, that is, the time that it takes to classify new documents. Like we said in the restrictions section of this chapter (4.2), we will only examine feature selection techniques and not feature extraction techniques. The feature selection techniques that we will examine are: document frequency, mutual information, information gain, chi-square and odds ratio.

We will construct a script for each of these techniques, that will remove a certain part of the vocabulary, according to the formulas of the feature selection techniques as stated in 1.4. When the vocabulary is reduced, we will use the best feature weighting script of the previous experiment to convert a set of documents into its datafile, and train the SVMs with these datafiles, to see what difference the dimensionality reduction techniques make.

Experiment 6 - Multiclass SVM algorithm (research question 10)

In the final experiment we will try if the training speed, classification speed or classification rate of a SVM can be improved by using another multiclass SVM method. Until now, we used the standard multiclass SVM method of *LIBSVM*, namely: one-against-one, but in this experiment we will take a look at the other two multiclass SVM methods: one-against-all and DAGSVM. Chih-Wei Hsu and Chih-Jen Lin showed in [19] that one-against-one is the most suitable multiclass SVM method, but we want to examine if this is also true for the problems we are examining.

In order to examine the other two multiclass SVM methods, we will make use of two modifications of *LIBSVM*, which makes *LIBSVM* make use of these other two methods, instead of the standard one-against-one method. These modifications can be found on the tools section of the LIBSVM website [8]. As said, we will use the results of the previous experiments to make the other design questions for constructing the classifiers.

5 Research results

In the last section of the previous chapter (4.3), we described 6 experiments with which we will give answer to the research questions of this thesis. In this chapter we will cover the results of these experiments.

5.1 Experiment 1

In the first experiment we will try if we can find some relation between the number of categories, the number of examples, and the C and γ parameter for the radial kernel. Such a relation would make things a lot easier, because then, we could determine the parameters of a large problem much quicker by doing a grid search on a smaller (and therefore quicker) subproblem.

The following 6 figures show the results of the grid searches on 6 different problems. All the 6 problems consist of 10 classes, but have different number of examples that are used to train and test the SVMs with. The number of examples that are used per problem, range from 50 to 1600 examples per category. Each figure shows the recognition rates of the SVMs that are trained with different C and γ combinations.

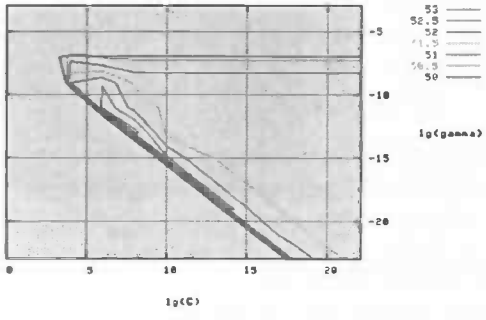

Figure 6: 10 classes, 50 examples per class

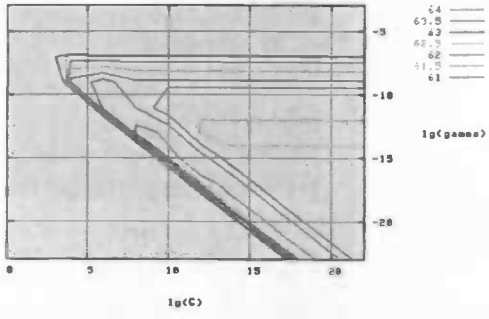

Figure 7: 10 classes, 100 examples per class

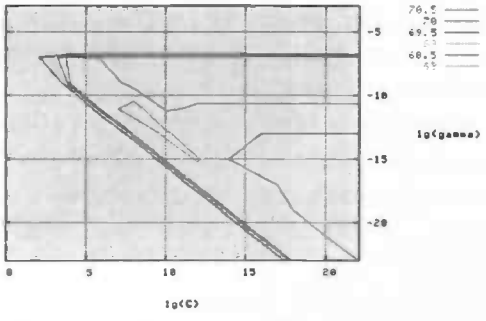

Figure 8: 10 classes, 200 examples per class

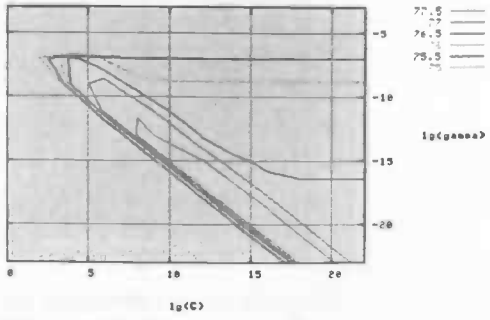

Figure 9: 10 classes, 400 examples per class

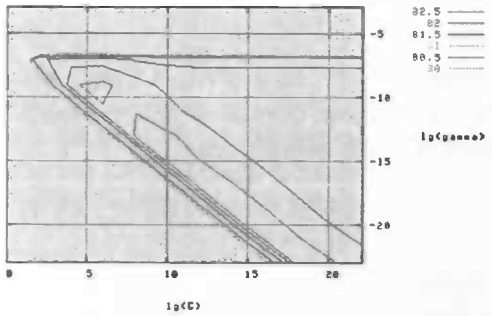

Figure 10: 10 classes, 800 examples per class

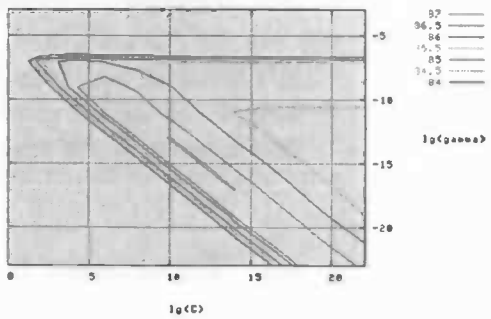

Figure 11: 10 classes, 1600 examples per class

The first thing that catches the eye in the 6 figures above is that the shapes of these figures are more or less the same, or in other words, all the top recognition rates (the colored lines) for each problem can be found in roughly the same area. And in this area, all problems have a best C and γ combination (at least) somewhere near $C = 2^{9}$ and $\gamma = 2^{-13}$. But before we jump to conclusions, we will first take a look at the cases where each problem consists of 20 classes instead of 10.

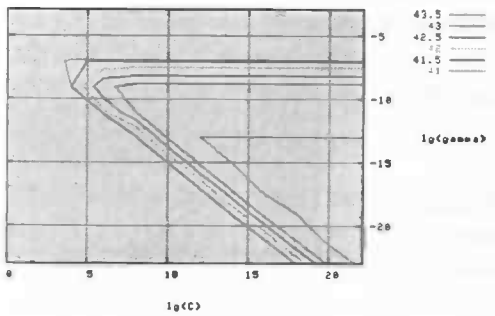

Figure 12: 20 classes, 50 examples per class

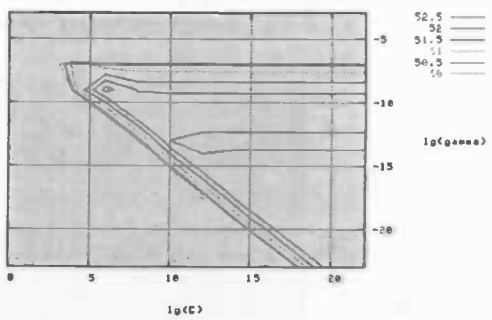

Figure 13: 20 classes, 100 examples per class

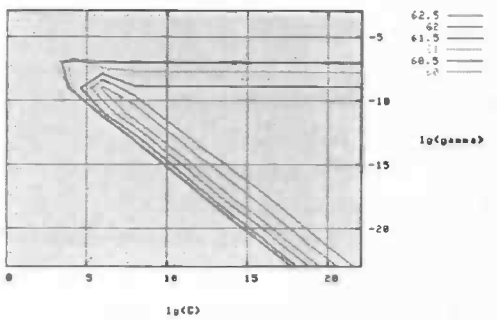

Figure 14: 20 classes, 200 examples per class

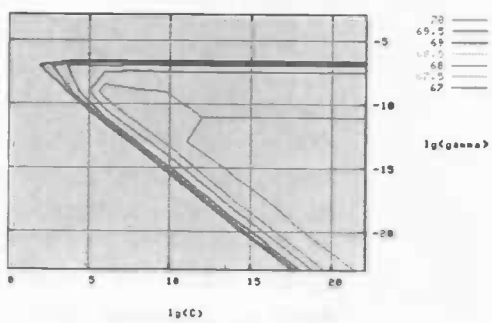

Figure 15: 20 classes, 400 examples per class

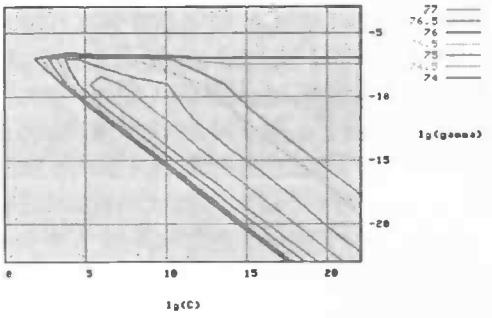

Figure 16: 20 classes, 800 examples per class

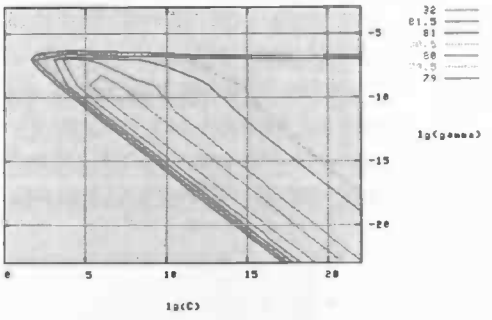

Figure 17: 20 classes, 1600 examples per class

The figures of the 20 class problems again show a more or less same area in which the top recognition rates can be found. But there is something strange about all the results in this experiment. Every problem, no matter if we use 10 or 20 classes, with 50 or 1600 examples per class, has several completely different C and γ combinations that come up with the same recognition rate. There are even an infinite number of totally different C and γ combinations per problem, that lead to the best recognition rate. These best combinations all lie, more or less, on a line that runs from $C = 2^6$ and $\gamma = 2^{-9}$ to $C = 2^{20}$ and $\gamma = 2^{-23}$, and in most cases probably even further. This behavior seems kind of strange, and made us look for some other researches that encountered the same behavior. We found an answer to this behavior in [23]. Sathiya Keerthi and Chih-Jen Lin state there in case 4, where they use $1/\sigma^2 = \gamma$ as the radial kernel parameter, that

> "If $\sigma^2 \to \infty$ and $C = \widetilde{C}\sigma^2$ where $\widetilde{C}$ is fixed then the SVM classifier converges to the Linear classifier with penalty parameter $\widetilde{C}$."

For the problems in this experiment, where we use γ instead of $1/\sigma^2$ as the radial kernel parameter, this means that when $\gamma \to 0$ and $C = \widetilde{C}/\gamma$ where $\widetilde{C}$ is fixed, the SVM converges to the linear classifier with parameter $\widetilde{C}$. This behavior of C and γ is exactly what is going on in the problems of this experiment! Roughly taken, the best recognition rates in each problem can be found one a line between the points $C = 2^{10}$, $\gamma = 2^{-13}$ and $C = 2^{20}$, $\gamma = 2^{-23}$. For each point on this line holds that $\gamma \to 0$ and $C = \widetilde{C}/\gamma$. This would imply that the SVM converges to the linear classifier with parameter $\widetilde{C} = 2^{-3}$! This could be interesting, because a linear classifier is only dependent on the C parameter, and can be calculated much faster than the radial kernel.

We will now try and see if this is really true; can we attain the same recognition rate with the linear kernel as with the radial kernel? In order to find an

answer to this question we will take the 20 class problem with 1600 examples per class, and determine the best classification rate of this problem that can be attained with a linear classifier. We will find the best possible recognition rate by training the linear SVM with a number of different C values that are all around the estimated value of $C = 2^{-3}$. In the following figure we can see the recognition rates that were attained by the linear classifier for a number of different C values.

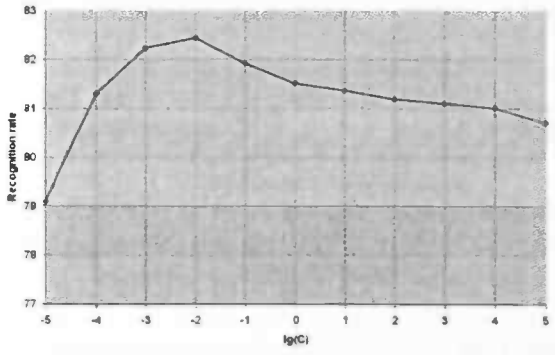

Figure 18: 20 classes, 1600 examples per class, linear kernel

The figure shows us that the best recognition rate of the linear classifier, with 82%, is indeed the same as its radial variant! Even the estimated best value of $C = 2^{-3}$ seems right. Although the best recognition rate is achieved with $C = 2^{-2}$, the estimation seems right, because $C = 2^{-3}$ is second best and is close to $C = 2^{-2}$.

This result is great news, because it will make things a lot easier and faster for the rest of this research. In the remaining experiments we will now use the linear kernel and determine the best recognition rate by trying out just a few different values for C, around $C = 2^{-3}$.

5.2 Experiment 2

In this experiment we will figure out what the difference is between training a SVM with balanced and imbalanced data.

We will set up a balanced and a imbalanced dataset for 5 problems, where each problem consists of a different number of total documents (10000, 20000, 40000, 80000 and 160000). The documents of the imbalanced dataset will be distributed over the classes in the same non-uniform way as in the real world. For the larger problems, the balanced dataset might not be completely balanced, because there just are not enough documents present for the underrepresented classes to make it completely balanced. In these cases we will not use the over-sampling method (see 1.1.3), to get enough

examples for the underrepresented classes and keep the total number of train documents equal to the imbalanced dataset, but we will just use some more documents of the classes that have enough example documents. This will hardly affect the balancedness of the dataset, because only the real small classes do not have enough examples, and besides that, this problem will only occur for the real large datasets. The following figure will show how the documents of the balanced and imbalanced datasets of the 160000 problem will be distributed over the classes.

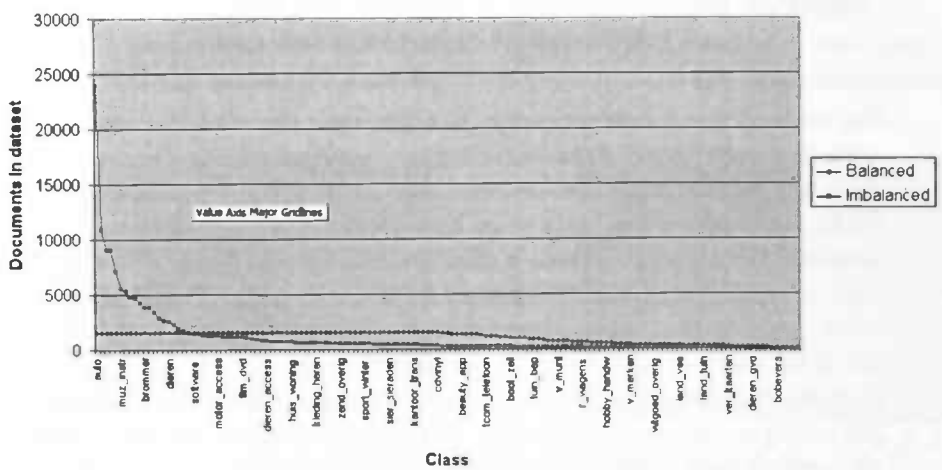

Figure 19: Balanced and imbalanced distribution of 160000 documents.

Besides these trainsets (that will most likely have a number of misclassified documents), we will also need a testset of which all the documents are correctly classified. Therefore we will have to manually verify the classlabels of each document of this testset.

The following figure shows the recognition rates that the classifiers attained on the verified testest, when trained with the imbalanced and balanced trainsets of different sizes.

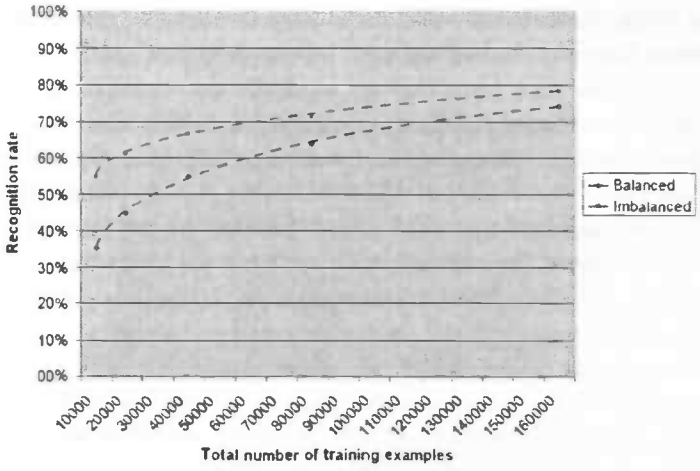

Figure 20: Recognition rates of classifiers that were trained with balanced and imbalanced trainsets of different sizes.

For each of the 5 five problems, it holds that the classifier that is trained with the imbalanced data has a higher recognition rate than the classifier that was trained with the balanced data. It looks like it is better to train a SVM with imbalanced data than with balanced data, but one can question how well the recognition rate is distributed over the classes when imbalanced data is used. It can well be that an imbalanced trainset will make the classifier focus on the overrepresented classes, for which documents occur more frequent. This would result in a classifier that is very good in classifying documents from these classes, but is very poor in classifying documents from the underrepresented classes. In this way, a classifier attains a high overall performance, because the major part of the documents will be classified correctly.

To see if this assumption is true, we will take a look at how well the documents of every class apart are classified, for both the balanced and imbalanced classifier. We will use the 160000 problem, for which the classifier that was trained with the balanced data attained a recognition rate of 74.2%, and the imbalanced variant attained a recognition rate of 78.2%. To see what the difference is between training a SVM with balanced and with imbalanced data, we will do the following for both the balanced and imbalanced classifier:
First we will will make a figure that shows the percentage of correctly classified test documents per class, and order this in descending way, so that the class, of which most documents are classified correctly, will be at the most left side on the x-axis. Underneath this figure we will plot a figure that has the same ordering of classes, and shows how much test documents there are

for each class. Note that the ordering of the classes will be different for the balanced and imbalanced cases, but the same for both figures of the same distribution (balanced or imbalanced).

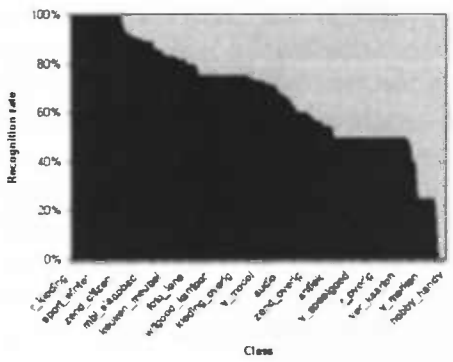

Figure 21: Recognition rate distributon of the **balanced** classifier

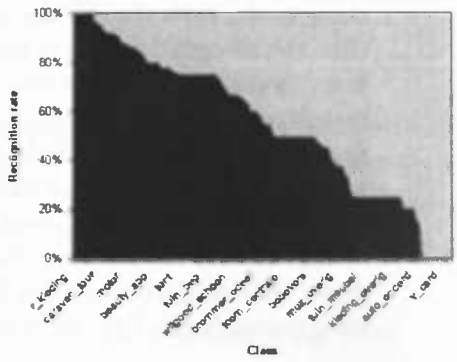

Figure 22: Recognition rate distributon of the **imbalanced** classifier

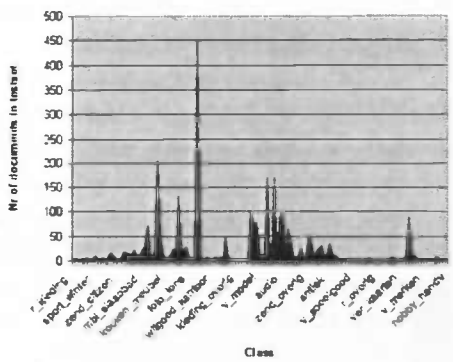

Figure 23: Test document distribution of the **balanced** classifier

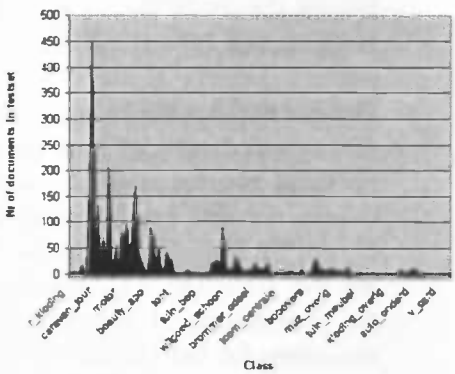

Figure 24: Test document distribution of the **imbalanced** classifier

The first thing that catches the eye is that, although the imbalanced classifier has a higher overall recognition rate than the balanced one (78.2% against 74.2%), the area of figure 21 is much larger than that of figure 22. This means that there are more classes with a high recognition rate with the balanced classifier than with the imbalanced one. When we take a look at the lower figures (figure 23 and 24), we can see that for the imbalanced classifier, most of the overrepresented classes lie on the left side of the figure (and therefore have a relatively high recognition rate), and that these classes are more spread out for the balanced classifier. This confirms our assumption that the imbalanced classifier gets its high recognition rate from trying to correctly classify as much documents of the frequently occurring

classes as possible, while 'ignoring' the documents of the underrepresented classes. This is obviously not the right way to make a useful text classifier.

These results make clear that training with balanced data is the only right thing to do, even though the overall recognition rate of the classifier is lower than the recognition rate of the imbalanced classifier. A positive thing comes from figure 20, where we can see from the two dashed lines that, the more train documents we use, the closer the recognition rate of the balanced classifier comes to that of the imbalanced classifier. In the following experiments, we will therefore use the balanced dataset, consisting of a total number of 160000 documents to train the SVMs with, and the verified imbalanced testset to test them with. (Using more than 160000 documents would slow down the trainproces of the SVMs dramatically, because of the quadratic time complexity of SVMs with the number of train documents)

5.3 Experiment 3

With this third experiment we will check what influence stopword extraction has on the performance of a classifier, by looking at the difference between training and testing a SVM with, and without stopword extraction.

Besides the standard choices for the experiments we have not conducted yet (document frequency and one-against-one), we will use the results of the previous two experiments to construct an as good as possible SVM. This is why we will use balanced data, a linear kernel and a C value around $C = 2^{-3}$. In order to get the best possible recognition rates of the SVMs, we will train the SVMs with a few different values around $C = 2^{-3}$, but for convenience of comparison, we will only report the results of the best classifiers for both problems.

The results of the experiment showed that training a SVM with the use of stopword extraction increases the recognition rate considerably, as opposed to its variant where no stopword extraction is used. In this experiment the recognition rate increased from 74.8171% without using stopword extraction, to 76.4194% with the use of stopword extraction (both recognition rates were obtained with $C = 2^{-3}$). Another, probably even more interesting result of this experiment was, that it took about 27% less time to train a SVM with the use of stopword extraction than without! This result is somewhat startling, because only 99 of the 279997 words were removed from the vocabulary. These stopwords obviously occur so often, that removing them, significantly reduces the size of the datafiles with which the SVMs are trained. These smaller datafiles make the training of a SVM much faster, because of the lesser amount of computations that need to be calculated.

These results convinced us to make use of stopword extraction in the remaining experiments. But besides increasing the recognition rate, and reducing the traintime, it is also more intuitive to use stopword extraction, because stopwords should not have any influence on the choice of a classifier.

5.4 Experiment 4

With this experiment we will find out if we can improve the classification rate of a SVM, by using some other feature weighting method than frequency weighting. We will examine the following feature weighting methods: Boolean weighting, frequency weighting, tf-idf weighting, tfc weighting, ltc weighting and entropy weighting.

In this experiment we will use the standard one-against-one multiclass SVM method and the results of the previous experiments (linear kernel, balanced data and stopword extraction), to construct the SVMs with the different feature weighting methods. We will again experiment a little with the C parameter to get the best performance out of the SVMs. The best value for C will however always be close to the estimated $C = 2^{-3}$. For convenience of comparison, we will only report the results of the best classifiers for each feature weighting method.

The following figure shows the best recognition rates that were attained with every one of the different feature weighting methods.

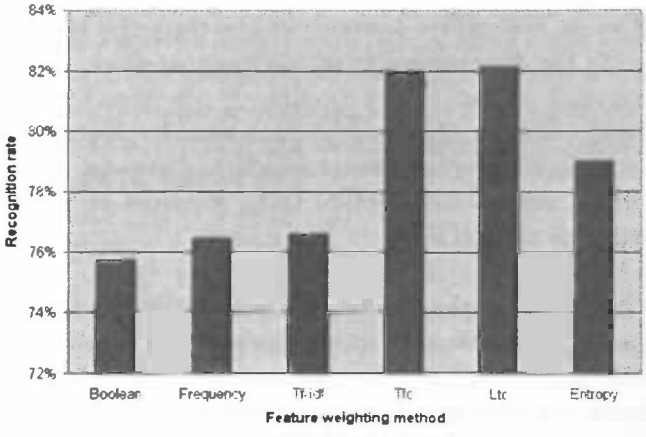

Figure 25: Recognition rates while using different feature weighting methods.

This experiment showed that the recognition rate of a classifier increases enormously when tfc or ltc weighting is used, instead of standard frequency weighting. The recognition rate increased from 76% with frequency weighting, to a respectable 82% with tfc or ltc weighting! The price that we have to pay for this increasing recognition rate, is that it takes a lot more time

to construct a datafile of a set of documents with tfc and ltc weighting, because we have to calculate more complicated computations for assigning weights to the features of a document. It also takes more time to train a SVM with a datafile that is constructed with tfc or ltc weighting (almost 3 hours for frequency weighting and 4 hours for tfc and ltc weighting), but this is a small price for such a gain in the recognition rate. The increase in the training time is partly due to using floats instead of integers, for the weights of the features of a document.

Entropy weighting resulted in a somewhat disappointing increase in the recognition rate, as opposed to frequency weighting. Although it takes an enormous amount of time to calculate the sophisticated feature values of all the documents, there was only a minor increase in the recognition rate of the classifier. Entropy weighting takes that long to calculate, because, in order to determine the feature values, it has to loop through every one of the documents, for each word in the vocabulary.

The results of this experiment strongly suggest to use either tfc or ltc weighting instead of frequency weighting. From now on, we will use ltc weighting, because, besides that ltc weighting came up with the highest recognition rate, it is also more intuitive than tfc weighting, because it reduces the effects of large differences in frequencies (see 1.3).

5.5 Experiment 5

With this fifth experiment, we will try to speed up the train and test time of a classifier, by reducing the dimensionality of the feature vector, under the condition that the recognition rate of the classifier is not (significantly) reduced. In order to realize this dimensionality reduction of the feature vector, we will experiment with the following feature selection methods: document frequency (DOC_FREQ), mutual information (MI), information gain (IG), chi-square (CHI) and odds ratio (OR).

In [17] and [50] we can see that the dimensionality of a feature vector can easily be reduced up to 90%, without harming the performance of the classifier. In this experiment, we will use an aggressivity level of 0.75, which means that we will remove 75% of the words of the vocabulary by means of the feature selection methods. The vocabulary will therefore be reduced from 279898 to 69974 words. We will use this aggressivity level instead of 0.9, because we want to make the gain in train and test speed as high as possible, but we most certainly do not want to harm the recognition rate of the classifier.

In the following figure we will see what effect the different dimensionality

reduction techniques have on the recognition rate of the classifier. The bar at the most left side, represents the recognition rate that is attained without any form of dimensionality reduction, and is meant for comparison.

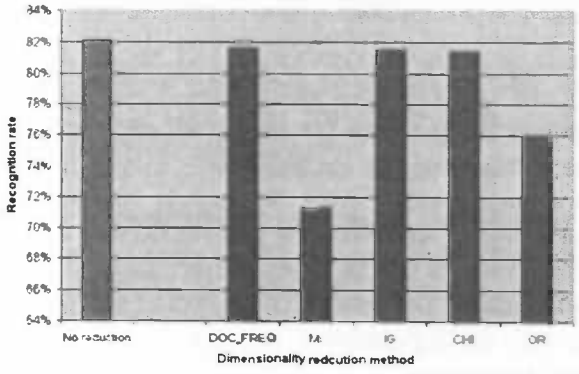

Figure 26: Recognition rates while using different dimensionality reduction methods.

This experiment, first of all, showed that MI (just like in [50]) and OR significantly reduce the recognition rate of the classifier, when an aggressivity level of 0.75 is used, and are therefore not useful for dimensionality reduction. The other three methods (DOC_FREQ, IG and CHI) on the other hand, show hardly any reduction in the recognition rate and can therefore be very useful. It is surprising that a simple method like DOC_FREQ, which simply removes non-frequent words, performs evenly well as the more complicated IG and CHI methods. This, once again, confirms the fact, found by Yang and Pedersen in [50], that DOC_FREQ is not just an ad hoc approach, but is a reliable method for reducing the dimension of the feature vector. Yang and Pedersen showed that there is a strong correlation between IG, CHI and DOC_FREQ, and that therefore DOC_FREQ can be used instead of IG and CHI just as well.

Now that we know that we can safely reduce the dimensionality of the feature vector with 75% by using DOC_FREQ, IG or CHI, we shall now take a look at the gain in train and test speed of this reduction. All the three interesting methods reduced the traintime of the SVM from 4 hours down to about 3 hours. This is a nice result, but a reduction in the testtime would be even nicer, because this would reduce the time that people have to wait when they want their document classified. The reduction in the testtime is however, somewhat disappointing. The testtime only reduced with about 5%, and therefore makes dimensionality reduction not very useful. However, we will use DOC_FREQ, because it does reduce the traintime and does remove a lot of unimportant words from the vocabulary.

5.6 Experiment 6

In this final experiment, we will see what difference training with the other multiclass SVM methods bring about. We will experiment with the following two multiclass SVM methods: one-against-one and DAGSVM.

First we will discuss the results of the DAGSVM method. Training a SVM with DAGSVM is exactly the same as with the standard one-against-one method (see 3.5), and therefore has the same traintime. However, DAGSVM did drastically reduce the testtime of the classifier with about 75%! The downpart of classifying documents with DAGSVM, is that DAGSVM only comes up with the winning category, and does not give probability estimates for all the categories. If this is enough for the problem the classifier is needed for, than DAGSVM is a wise choice, because the recognition rate and traintime is about the same as with one-against-one, but DAGSVM has a much quicker way to classify new documents.

One-against-all on the other hand, seems to be useless for the large problems we are discussing in this research. Training a SVM with one-against-all took more than 10 times as long as with one-against-one and DAGSVM, and took more than a day. This enormous increase in traintime is due to the fact that every one of the SVMs that will be constructed with one-against-all, have to be trained with everyone of the training documents, and the time that it takes to train a SVM is quadratic with the number of examples that it is trained with. The fact that considerably less SVMs have to be constructed with one-against-all as opposed to one-against-one and DAGSVM (144 instead of 10296), does not compensate the increase in traintime per SVM. The testtime of a SVM that is built with one-against-all, also increased with a factor 10, resulting in a SVM that took more than 2 seconds per document to classify. The recognition rate of the classifier was, however, about the same as the classifiers that were trained with one-against-one and DAGSVM.

The results of this experiment make clear that, the one-against-one method was a good choice after all and that, only when the classifier is constructed for a problem where no probability estimates are needed, the DAGSVM is a better choice.

6 Conclusion

In this chapter we will present the conclusions that can be drawn from the results of the researches we have done. These conclusions together will give answer to the main research question of this thesis:

What choices can best be made in the design process of an automatic text classifier for large-scale and complex real world text categorization problems?

We will discuss the conclusions of the research questions one by one, and in the same order as these research questions were presented in 4.1. This way, we will set up a chronological step-by-step plan for the design process of automatic text classifiers for large-scale and complex real world text categorization problems.

Most of the conclusions were already drawn in chapter 5, because we needed them to continue to the next experiment, so we will mainly recapitulate these conclusions in this chapter.

Stopword extraction

The results of experiment 3 (see 5.3), showed that the extraction of stopwords is a great way to improve the automatic text classifier. It not only improves the recognition rate, but also drastically reduces the time that it takes to train the classifier. Besides these positive results, it is also more intuitive to use stopword extraction, because stopwords should not have any influence on the classification process of a classifier. And finally we can state that stopword extraction is also very easy to implement, because we only have to remove the words of the stoplist from the vocabulary.

Stemming

In this research we did not experiment with stemming, because we thought it would not improve the performance of a classifier. Especially in our case, where documents will contain a lot of spelling mistakes and short catchwords, stemming will rather have a negative than a positive effect on the classification rate of the classifier. Our advice is therefore not to use stemming, but it can well be that stemming does improve the classifiers for other text categorization problems.

Class imbalance

With experiment 2 (see 5.2) we extensively investigated the differences between training a classifier with balanced and imbalanced data. The results

of this experiment showed that a classifier that is trained with imbalanced data reaches a higher recognition rate than a classifier that is trained with balanced data, but comes with a major drawback. Training with imbalanced data will make a classifier very good in classifying documents that belong to the overrepresented classes, but extremely bad in classifying documents that belong to the underrepresented classes. In most cases such a phenomenon is not desirable, and therefore training with a balanced dataset is the only wise thing to do.

Features

We did not experiment with different kinds of feature representations of a text document, to train a classifier with, but we chose to use only the Bag Of Words representation. Besides that this way of representing a text document is widely accepted and is the most used way to do this, it seemed to be a very good choice. The recognition rate of over 82% that we attained for the difficult problem of this research, was above our expectations. However, this of course does not mean that the other methods could not have been an even better choice. Letter n-grams for instance, seem very promising for problems with a lot of spelling mistakes, like the one we used in this research. Word n-grams could also be interesting, because good results are accomplished with it (see [16]), but our guess is that it only duplicates the information carried in single word features. On the other hand, the linguistic approach of representing text documents is still not far enough developed, and is not very useful in practice yet.

Feature weighting

Experiment 4 (see 5.4) clearly showed us that tfc and ltc weighting are the best ways to reflect the importance of a feature/word for a document. Ltc weighting was slightly better than tfc weighting, and is also more intuitive than tfc weighting, because it reduces the effects of large differences in frequencies (see 1.3). This is why we suggest the use of ltc weighting, to assign weights to features, for the problems we discuss in this thesis.

Dimensionality reduction

Although we introduced dimensionality reduction as practically essential for large-scale text categorization problems, the results of experiment 5 (see 5.5) showed that this was in fact not completely true. Dimensionality reduction did indeed reduce the time that is needed to train a classifier, and did reduce the time that it takes to classify new documents, but did not have the expected tremendous result that would make dimensionality reduction really essential. Because of this, and because of the fact that the recognition rate

of the classifier did worsen (although slightly) with all dimensionality reduction methods, we suggest not to use any dimensionality reduction method. However, if the reductions in train- and testtime are valuable for a specific problem, we suggest the document frequency method with an aggressivity level around 0.75. The results of experiment 5 showed that the only three dimensionality reduction techniques that hardly decreased the recognition rate of a classifier are: document frequency, information gain and chi-square (of which document frequency came out best). However, document frequency is by far the most easiest method to implement of these three. All you have to do is: keep 25% of the words of the vocabulary that occur most frequent in different training documents, and expunge the rest of the words from the vocabulary.

Machine learning algorithm

The question about which machine learning algorithm we should use, was actually not really a question in this research. We decided to use the Support Vector Machine (SVM) learning algorithm, and we discussed in great detail why we made this choice, in chapter 2. We, of course, can not be sure if this was a wise decision after all, but we can say that the traintime, the testtime and the recognition rate of the classifier exceeded all our expectations.

Kernel

Experiment 1 (chapter 5.1) made clear that the linear kernel is the best choice for large-scale and complex text categorization problems. The performance of the linear kernel is equally good as the radial kernel (because the radial kernel converges to the linear one for such problems), but is much easier to calculate. Besides that, no extra parameter needs to be optimized for the linear kernel, as opposed to the radial kernel.

Parameters

Because of the fact that the linear kernel is the best choice for the problems discussed in this thesis, we only have to deal with the C parameter of the SVM. The best way of finding the best value of this C parameter, is by using a small grid-search around $C = 2^{-3}$, and pick the one where the performance is at its highest. When you notice that the performance of the classifier is still increasing at a boundary of your grid search, you have to continue your search for the best value, in this direction.

SVM solving algorithm

In this research we used the software package *LIBSVM* [8], to construct a SVM, because we did not want to implement the SVM solving algorithms

ourselves. Implementing these algorithms is extremely difficult and takes a lot of time. *LIBSVM* makes use of a simplification of both SMO and SVM^{light}, and worked fine for our problem.

Multiclass method

The results of experiment 6 (see 5.6) showed that the one-against-one method is the best multiclass method for large-scale and complex text categorization problems. Only when the classifier is constructed for a problem where it is enough to let the classifier determine the winning class, and no probability estimates for all the classes are needed, DAGSVM is a better choice. In these cases, DAGSVM has a considerable faster way of classifying new documents than one-against-one.

We will conclude this thesis with a summary of the choices that can best be made in the design process of an automatic text classifier for large-scale and complex real world text categorization problems.

Design question	**Answer**
Stopword extraction	Yes
Stemming	No
Class imbalance	Use balanced traindata
Features	Bag Of Words
Feature weighting	Ltc
Dimensionality reduction	None
Machine learning algorithm	SVM
Kernel	Linear
Parameters	Around $C = 2^{-3}$
SVM solving algorithm	Simplification of both SMO and SVM^{light}
Multiclass method	One-against-one

7 Future work

In this last chapter, we will discuss some issues that were left unanswered in this thesis because of time shortage, but are interesting enough to be investigated in future research.

A first thing that would be interesting to do some further research on, is the use of different features to represent the documents with. In this research we only used the Bag Of Words approach, but word n-grams, and especially letter n-grams look very interesting ways to represent documents. Word n-grams could improve the recognition rate of the classifier somewhat, and letter n-grams seem very promising, because they can cope with spelling mistakes.

It would also be interesting to take a look at the use of a hierarchical structure of classifiers. Such a structure consists of several classifiers that are arranged in a hierarchical, tree-like way. The classifier in the first layer will make the distinction between combinations of several classes, and the second layer of classifiers will classify the documents of specific groups of classes into the classes itself. A possible hierarchy could be to combine all vehicles, all electronics, all house related products, etcetera. The first classifier will then decide to which group of classes a document belongs, and the classifier for that group will actually place the document into its class. The classification process of a document would go as follows: when a car-document is offered to the classifier, the first layer will place it in the vehicle-class, and the vehicle classifier in the second layer will then classify the document into the car-class. The classifier in the first layer, that will have to place a document into a group of classes, will probably have a high recognition rate, because these groups are quite distinct from each other. The classifiers in the second layer on the other hand will have a much more difficult job, because all the documents that belong to a group will have something in common. It would be interesting to see if a hierarchical structure of classifiers will have a better performance than a single classifier.

References

[1] Aas, K., and Eikvil, L. Text categorisation: A survey, 1999. (citeseer.ist.psu.edu/aas99text.html)

[2] Aizerman, M., Braverman, E., and Rozonoer, L. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control 25* (1964), 821 – 837.

[3] Attardi, G., Simi, M., Tanganelli, F., and Tommasi, A. Learning conceptual descriptions of categories. Tech. Rep. TR-99-21, Dipartimento di Informatica, Università di Pisa, November 1999.

[4] Brill, E. A simple rule-based part of speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing* (Trento, IT, 1992), pp. 152–155. (http://citeseer.nj.nec.com/brill92simple.html)

[5] Burges, C. J. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery 2*, 2 (1998), 121–167.

[6] Caropreso, M. F., Matwin, S., and Sebastiani, F. A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization. In *Text Databases and Document Management: Theory and Practice*, A. G. Chin, Ed. Idea Group Publishing, Hershey, US, 2001, pp. 78–102.

[7] Cavnar, W. B., and Trenkle, J. M. n-gram-based text categorization. In *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval (SDAIR-94)* (Las Vegas, Nevada, U.S.A., Apr. 1994), UNLV Publications/Reprographics, pp. 161–175. (http://www.novodynamics.com/trenkle/papers/sdair-94-bc.ps.gz)

[8] Chang, C.-C., and Lin, C.-J. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[9] Chih-Wei Hsu, C.-C. C., and Lin, C.-J. A practical guide to support vector classification, 2003.

[10] Collobert, R., Bengio, S., and Mariéthoz, J. Torch: a modular machine learning software library. Technical Report IDIAP-RR 02-46, IDIAP, 2002.

[11] Cortes, C., and Vapnik, V. Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

[12] COURANT, R., AND HILBERT, D. *Methods of Mathematical Physics*, vol. 1. Interscience Publishers, Inc, New York, 1953.

[13] DUMAIS, S. Using svms for text categorization. In *IEEE Intelligent Systems Magazine, Trends and Controversies, Marti Hearst, ed., 13(4)* (July/August 1998).

[14] DUMAIS, S., PLATT, J., AND HECKERMAN, D. Inductive learning algorithms and representation for text categorization. In *CIKM-98, 7th ACM International Conference on Information and Knowledge Management* (1998), pp. 148–155.

[15] EVGENIOU, T., PONTIL, M., AND POGGIO, T. A unified framework for regularization networks and support vector machines. Tech. Rep. 1654, A.I. Memo, 1999.

[16] FÜRNKRANZ, J. A study using n-gram features for text categorization. In *Technical Report OEFAI-TR-9830, Austrian Institute for Artificial Intelligence.* (1998).

[17] GALAVOTTI, L., SEBASTIANI, F., AND SIMI, M. Experiments on the use of feature selection and negative evidence in automated text categorization. In *Proceedings of ECDL-00, 4th European Conference on Research and Advanced Technology for Digital Libraries* (Lisbon, PT, 2000), J. L. Borbinha and T. Baker, Eds., Springer Verlag, Heidelberg, DE, pp. 59–68. Published in the "Lecture Notes in Computer Science" series, number 1923.

[18] GAUSTAD, T., AND BOUMA, G. Accurate stemming of Dutch for text classification. In *Computational Linguistics in the Netherlands 2001* (Amsterdam, 2002), M. Theune, A. Nijholt, and H. Hondorp, Eds., Rodopi.

[19] HSU, C.-W., AND LIN, C.-J. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks 13*, 2 (2002), 415–425.

[20] JAPKOWICZ, N. Learning from imbalanced data sets: A comparison of various strategies. In *AAAI Workshop on Learning from Imbalanced Data Sets, Technical Report WS-00-05* (Menlo Park, CA, 1980), AAAI Press.

[21] JOACHIMS, T. Text categorization with support vector machines: learning with many relevant features. In *Proc. 10th European Conference on Machine Learning ECML-98* (1998), pp. 137–142. (`http://www-ai.cs.uni-dortmund.de/DOKUMENTE/joachims_98a.ps.gz`)

[22] JOACHIMS, T. Making large-scale SVM learning practical. In *Advances in Kernel Methods — Support Vector Learning* (Cambridge, MA, 1999), B. Schölkopf, C. Burges, and A. Smola, Eds., MIT Press, pp. 169–184.

[23] KEERTHI, S. S., AND LIN, C.-J. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computation 15*, 7 (july 2003), 1667–1689.

[24] KIVINEN, J., WARMUTH, M. K., AND AUER, P. The perceptron algorithm vs. Winnow: linear vs. logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence* (March 1997), 325–343.

[25] LEWIS, D. D. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of SIGIR-92, the 15th Annual International Conference on Research and Development in Information Retrieval* (Copenhagen, Denmark, June 1992), ACM Press, pp. 37–50.

[26] LIAO, C., ALPHA, S., AND DIXON, P. Feature preparation in text categorization.

[27] LUHN, H. The automatic derivation of information retrieval encodements from machine-readable texts. In *Readings in Information Retrieval* (San Francisco, 1997), Morgan Kaufmann Publishers, Inc., pp. 21–24.

[28] MCCALLUM, A. K. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow, 1996.

[29] MCCULLOCH, W., AND PITTS, W. A logical calculus of the idea immanent in nervous activity. *Bulletin of Mathematical Biophysics 5* (1943), 115–153.

[30] MILLER, G. A. Wordnet. A lexical database for the English language. (`http://www.cogsci.princeton.edu/~wn/`)

[31] MLADENIĆ, D., AND GROBELNIK, M. Word sequences as features in text-learning. In *Proceedings of ERK-98, the Seventh Electrotechnical and Computer Science Conference* (Ljubljana, SL, 1998), pp. 145–148.

[32] OSUNA, E., FREUND, R., AND GIROSI, F. An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop* (New York, 1997), J. Principe, L. Gile, N. Morgan, and E. Wilson, Eds., IEEE, pp. 276 – 285.

[33] OSUNA, E. E., FREUND, R., AND GIROSI, F. Support vector machines : Training and applications. Tech. Rep. AI-Memo 1602, M.I.T Artificial Intelligence Laboratory, March 1997.

[34] PLATT, J. C. Sequential minimal optimization: A fast algorithm for training support vector machines. Tech. Rep. MSR-TR-98-14, Microsoft Research, 1998.

[35] PONTIL, M., AND VERRI, A. Properties of support vector machines. *Neural Computation 10* (1997), 955–974.

[36] PORTER, M. An algorithm for suffix stripping. *Program 14(3)* (july 1980), 130–137.

[37] RAKESH MENON, LOH HAN TONG, S. S. A. B. Automated text classification for fast feedback - investigating the effects of document representation. In *Knowledge-Based Intelligent Information and Engineering Systems, 7th International Conference, KES 2003, Oxford, UK, September 3-5, 2003, Proceedings, Part II* (2003), Springer-Verlag, Heidelberg, pp. 1008–1014.

[38] RENNIE, J., AND RIFKIN, R. Improving multiclass text classification with the support vector machine. Tech. rep., MIT - A.I. Memo 2001-026, 2001.

[39] ROCCHIO, J. Relevance feedback in information retrieval. *G. Salton, editor, The Smart Retrieval System: Experiments in Automatic Document Processing* (1971), 313–323.

[40] SALTON, G., AND MCGILL, M. J. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[41] SCOTT, S., AND MATWIN, S. Feature engineering for text classification. In *Proceedings of ICML-99, 16th International Conference on Machine Learning* (Bled, SL, 1999), I. Bratko and S. Dzeroski, Eds., Morgan Kaufmann Publishers, San Francisco, US, pp. 379–388.

[42] SIOLAS, G., AND D'ALCHE BUC, F. Support vector machines based on a semantic kernel for text categorization. In *Proceedings of IJCNN-00, 11th International Joint Conference on Neural Networks* (Los Alamitos, US, 2000), vol. 5, IEEE Computer Society Press, pp. 205–209.

[43] VAPNIK, V. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New York, 1982.

[44] VAPNIK, V. *The nature of statistical learning theory.* Springer-Verlag, New York, 1995.

[45] VAPNIK, V., AND CHERVONENKIS, A. *Theory of Pattern Recognition.* Moscow, 1974.

[46] WEISSTEIN, E. W. Hilbert space. From MathWorld–A Wolfram Web Resource. (`http://mathworld.wolfram.com/HilbertSpace.html`)

[47] WEISSTEIN, E. W. Kuhn-tucker theorem. From MathWorld–A Wolfram Web Resource. (`http://mathworld.wolfram.com/Kuhn-TuckerTheorem.html`)

[48] YANG, Y., AND CHUTE, C. An example-based mapping method for text categorization and retrieval. *ACM Transactions on Information Systems 12*, 3 (1994), 252–277.

[49] YANG, Y., AND LIU, X. A re-examination of text categorization methods. In *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval* (Berkeley, US, 1999), ACM Press, New York, US, pp. 42–49.

[50] YANG, Y., AND PEDERSEN, J. O. A comparative study on feature selection in text categorization. In *Proceedings of ICML-97, 14th International Conference on Machine Learning* (Nashville, US, 1997), D. H. Fisher, Ed., Morgan Kaufmann Publishers, San Francisco, US, pp. 412–420.

Appendices

Appendix A

This appendix is used for calculations in 3.1.

$$
\begin{aligned}
L_p &= \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\vec{w}\cdot\vec{x_i}+b)-1) \\
&= \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i(\vec{w}\cdot\vec{x_i}+b) - \alpha_i \\
&= \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i(\vec{w}\cdot\vec{x_i}) + \alpha_i y_i b - \alpha_i \\
&= \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i(\vec{w}\cdot\vec{x_i}) - \sum_{i=1}^{n} \alpha_i y_i b + \sum_{i=1}^{n} \alpha_i \\
&= \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i(\vec{w}\cdot\vec{x_i}) - b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i \\
&\quad \{\text{Substitute equation 6: } \sum_{i=1}^{n} \alpha_i y_i = 0\} \\
&= \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i(\vec{w}\cdot\vec{x_i}) + \sum_{i=1}^{n} \alpha_i \\
&= \frac{1}{2}\vec{w}\cdot\vec{w} - \sum_{i=1}^{n} \alpha_i y_i(\vec{w}\cdot\vec{x_i}) + \sum_{i=1}^{n} \alpha_i \\
&\quad \{\text{Substitute equation 7: } \vec{w} = \sum_{i=1}^{n} \alpha_i y_i\} \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j - \sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\vec{x_i}\cdot\vec{x_j}) + \sum_{i=1}^{n} \alpha_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\vec{x_i}\cdot\vec{x_j})
\end{aligned}
$$

Appendix B

This appendix is used for calculations in 3.1.

$$
\begin{aligned}
L_d &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\vec{x_i} \cdot \vec{x_j}) \\
&\quad \{\text{Substitute: } D_{ij} = y_i y_j (\vec{x_i} \cdot \vec{x_j})\} \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j D_{ij} \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \alpha_i \sum_{j=1}^{n} \alpha_j D_{ij} \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \alpha_i (D\vec{\alpha})_i \\
&= -\frac{1}{2} \vec{\alpha} \cdot D\vec{\alpha} + \sum_{i=1}^{n} \alpha_i
\end{aligned}
$$