Master's Thesis

Classifier-Based Search
in large document collections

Jelle J.P.C. Schühmacher

– 644 –

17th January 2011

Supervisors:
Prof. C.H.A. Koster
Prof. dr. T.M. Heskes

In this thesis a design for a *classifier-based search* system was developed. The proposed search system enables an user to perform search using a more refined model for a topic or concept when compared to a set of keywords, namely a linear classifier learned from relevant texts.

The CBS systems translates the information contained in the linear classifier into queries suitable for a traditional full-text search engine. This step is divided into two phases to deal with the computational complexity of the problem. It is shown that CBS is closely related to the *subset-sum problem*, which is in the class of NP-complete problems. In order to limit the number of generated queries three heuristics are developed.

The first phase of CBS uses a *high-recall* linear classifier. For this phase *Positive Naïve Bayes* was researched and implemented. PNB is a high-recall classifier, but more importantly it has the added advantage that it needs positive examples and unlabelled documents only to learn a classifier.

The Balanced Winnow algorithm is used for the second phase, the *high-accuracy* phase, in CBS. In this phase the extraneous documents found in the first phase can be filtered out.

The suitability of the classifiers is experimentally determined by means of experiments on the EPO1A and EPO2F data sets. The two-phase design is validated by means of experiments on these data sets. It is shown that classifier-based search works. Using the CBS system a significant fraction of documents does not have to be evaluated.

Contents

1 Introduction

Information retrieval is a hard task to perform accurately and effectively. Currently, the most widespread information retrieval systems are the keyword search engines on the internet. Examples of these are Google, Yahoo, and Bing. They are typically built on the premise that a user is able to formulate his or her *information need* using a limited number of simple keywords, and is then willing to browse through at least the first few result pages to find the needed information. To support the user in this mode of operation, results are often ranked using some likelihood measure indicating that a particular piece of text satisfies the information need conveyed by the provided set of keywords. This way of working is called *ad hoc search*.

However, there are situations where another way of working is needed. In these situations the information retrieval system is typically part of a larger process, where it is required to find *all* results, because they are required for some next step. This next step might entail a manual analysis of the results, because the user is accountable for his findings. The way of working in these situations is called *professional search*. An example of professional search is the use of an information retrieval system in the discovery of prior art as part of a novelty search by a patent attorney. Failing to find a piece of prior art could result in financial damage to the patent attorney and its client. Therefore, professional search is in the first place *high recall* search.

This situation also occurs in the field of *digital forensics*. A sub-field of digital forensics is concerned with finding evidence in large collections of data, or monitoring streams of text for anomalies which point to illegal activities. An example that comes to mind is continuous monitoring of *text streams* or the web with the intent of signalling suspicious events such as disguised communication of extremist groups on blogs and forums [29].

In all of these situations it is the case that besides the relevant results returned by the search application, it also returns irrelevant results. These irrelevant results must be examined manually by the user and thus cost extra money. Therefore, it is necessary to have control over the *ratio of relevant and irrelevant results* a search system returns to keep the costs predictable.

Text classification is the automated labelling of text documents with pre-defined classes learnt from examples. The process of automatically classifying texts traditionally involves collecting a large corpus of documents pre-labelled with one or more classes. A large portion of the collected corpus is used as input to an algorithm that creates abstract models of the categories or classes that need to be recognised. Typically, these models are limited to purely statistical information on the occurrence of certain words in a category. With properly tuned parameters, text classification works surprisingly well in practice. The most widespread example is given by automatic e-mail filters that discriminate between spam and useful e-mail.

After a model has been learned, a text classification system still needs maintenance, but this ought to be far less costly than employing a human workforce for manual document labelling. A large portion of the costs are in collecting labelled documents and setting up the system. Therefore, it would be beneficial to have an automated system limiting the number of documents that need to be labelled in order to expand the training corpus.

1.1 Context & Problem Statement

This thesis describes the design of a *Classifier-Based Search* (CBS) system. In essence it is the combination of a traditional *full-text search engine* with a *text classification system* using *linear classifiers*. The linear classifiers created by a text classification system are used as the description of specific concepts or *topics*. The classifiers are transformed into traditional keyword queries by means of the CBS algorithm so that they become suitable for use with traditional full text search engines.

The main research question answered in this thesis is:

> *How to search effectively for maximally relevant texts on a specific topic by means of a linear classifier in a large closed document collection?*

In order to answer the main question a number of related questions will need to be answered:

1. What is the relation between the precision and recall of the linear classifier and the precision and recall of Classifier-Based search?
2. How can this relation be used to provide confidence on the expected precision and recall of Classifier-Based search?
3. How to limit the number of documents that must be manually judged by a professional?
4. What are the optimal number of terms for the classifiers?

The objective of this thesis is to find a way of using a classifier to search for maximally relevant texts given a full-text index. Due to constraints of time and resources this is limited to search in large closed document collections. However, there is no reason the same techniques cannot be used for *text streams* or open collections such as the internet.

1.2 Related Work

Apart from research into some of the components that make up Classifier-Based search there has been related work on CBS as a whole. These related areas are called *topical crawling* or *focused crawling*.

After the first large search engines became operational in the 1990s it became clear that the concept of a general purpose search engine did not scale well. It seemed impossible to crawl and index the entire World Wide Web. This is still true today, in 2005 Gulli et al. estimated [12] that the largest search engine at that time, Google, only covered 68.2% of the estimated 11.5 billion pages the *indexable* web had. The complete World Wide Web is much larger than that.

Furthermore, it was difficult to get relevant results from a sufficiently large general purpose index of the World Wide Web, because a query term could describe a different concept depending on the *unknown* context of a user's information need. At that time a lot of research was done on how to order the results of search queries to most effectively satisfy users' information needs [].

1.2.1 Topical or Focused Crawling

In 1997 Menczer [24] argued that improving ranking of results only attempts to solve half of the problem. No matter how sophisticated the ranking algorithm is, the results can only be as good as the pages indexed by the search engine. If a page is not in the index, it cannot be found with the search engine. General crawlers use a combination of breadth-first search and depth-first search to crawl the entire web. As a result they visit a lot of pages that are not necessarily relevant to their users, and fail to visit a lot of pages that are.

Because downloading the entire World Wide Web seems impossible, Menczer proposed a method to maximise the portion of relevant pages visited in a crawl. This type of information discovery was later

called *topical crawling* or *focused crawling*. The original algorithm, ARACHNID, described in [24] is a type of genetic algorithm using "local selection" instead of comparing an individual crawlers' fitness with other members of the population. The author envisioned the following basic process:

1. Initialisation step:
   a) A user provides a list of keywords and a list of relevant starting points.
   b) The population of crawlers is initialised by positioning a crawler with an initial reservoir of "energy" and random behaviour at each document in the list of starting points.
2. Repeat while there are live crawlers in the population:
   a) Select a crawler *a*.
   b) Select a link for *a* and fetch the corresponding document.
   c) Update the energy reservoir for *a*. Following a link costs energy, finding a new document yields energy. If a user has judged the visited document it will cost or add energy depending on the documents relevance.
   d) Learn by reinforcement.
   e) Remove the crawler if its energy reservoir is depleted, otherwise add a mutated clone of the crawler to the population.

Menczer used a breadth-first crawl as baseline and showed that his approach performed significantly better in terms of recall. In other words, for a given number of fetched documents the portion of relevant documents was larger for ARACHNID than for the breadth-first crawl.

One of the drawbacks to the described approach is that the user is actively involved in the crawling procedure, he or she has to manually judge new documents to optimise the effectiveness of the reinforcement learning. Otherwise, the crawler can only use the similarity measure with the list of keywords that was initially provided by the user to guide the crawler to a relevant part of the World Wide Web.

1.2.2 Classifier-Based Crawling

Chakrabarti et al. [6] showed similar results in 1999 using an *automatic classifier* to guide a crawler. The process starts with a collection of example documents that the user judged interesting. This collection is compared to a large taxonomy of topics. The system extracts the most relevant topics from the taxonomy based on the examples provided by the user. The example documents combined with the example documents that were part of the taxonomy are used as positive examples for classifier training. All other documents in the taxonomy are used as negative examples. After this initial phase the actual crawler can be started. Every document that the crawler encounters is categorised using the classifier. If the classifier judges a document to be in the positive class, links contained in that document are considered for further exploration. Otherwise, the document and its links are pruned from the crawlers' queue and will not be visited. In accordance with Menczer, Chakrabarti et al. found that a focused crawler will yield a steady stream of mostly relevant documents. This yields a significant reduction in hardware and network requirements when compared to a breadth-first crawler.

The approach taken by Chakrabarti et al. was later coined *classifier-based crawling* in order to distinguish it from the approach taken by Menczer. Recent research seems to favour the classifier-based crawling approach instead of the supervised reinforcement-learning approach employed by Menczer. The method employed by Chakrabarti et al. is far less labour intensive for the end user. All he or she has to do is create a list of relevant examples. The authors use a trick to come by negative examples which are crucial for classifier training. The method requires a large taxonomy with examples for each class. The positive examples are placed in the taxonomy in their best fitting classes. Content from classes without positive examples from the user generated list will be used to provide negative examples.

Recent research includes determining which classifier to use in order to guide a web crawler most efficiently. Pant et al. performed a systematic comparison in [25] of the *Naive Bayes* (NBC), *Neural Network* (NN) and *Support Vector Machine* (SVM) classifier training algorithms. They came to the conclusion that Naive Bayes was a poor choice of classifier to use for classifier-guided web crawling. According to the authors both the Support Vector Machine and the Neural Network performed significantly better in terms of accuracy. The authors suggest that this is the result of systemic errors in the Naive Bayes classifier.

1.3 Contribution Of Our Research

Classifier-based search differs from classifier-based crawling. The latter tries to minimise the discovery of irrelevant documents to a topic, in order to build a specialised search engine at relatively low costs. However, to my knowledge there are no guarantees about what portion of relevant documents classifier-based crawling will eventually find. Results in [25] suggest that classifier accuracy is indicative of the eventual accuracy, but that recall on the target topic is actually pretty low [25]. This is no problem because classifier-based crawling is meant for ad hoc search. Any document answering a particular information need is good enough. However, it cannot be used in professional search where situations occur where users need all relevant results, and are accountable for what they do and do not find.

The approach presented here works differently, Classifier-based search is a sort of topical search that can be “bolted on" any search engine that provides an interface which understands queries with boolean modifiers. This method has several advantages over classifier-based crawling.

Firstly, in practice most search engines provide such a query interface. Therefore, the method could turn any existing general search engine into a topical search engine. Of course the user has to provide many training examples. Furthermore, it will only work if no measures are taken to block non-human use of the search engine interface. Unfortunately, search engines operators tend to block non-human use of their services quickly, so it may be necessary to contact a friendly search engine to get permission.

Secondly, classifier-based search is more suitable for use in a professional search situation where all relevant results are needed. A human operator has full control over the balance between *precision* and *recall* by adjusting a score threshold value. More on this can be found in Chapter 4.

Finally, finding the right result set of documents is more interactive compared to the classifier-based crawling method. In the approach presented here, the time consuming work of crawling has been done beforehand. Therefore, the human operator gets his results immediately after a classifier has been created from his example documents as opposed to getting the results after a new crawl of the documents has been performed. What actually happens is that CBS use a linear classifier to generate search queries that produce maximally relevant documents.

1.4 Outline

Chapter 2 shows a high-level architecture of a generic classifier-based search system to be used as a guide through the various components that will be discussed throughout the thesis.

Section 2.2 gives some background on the theory behind document categorisation using linear classifiers. Furthermore, some concrete algorithms that were implemented over the course of this thesis as well as some that were already present in the *Linguistic Classification System* (LCS) are discussed in some detail.

In Chapter 3 our classifier-based search algorithm is described in more detail. Arguments are given as to why the approach is feasible for text categorisation in the first place, and it is shown that it probably is not a realistic approach to non-text categorisation problems. A two-phase search protocol is introduced which must alleviate the effects of the inherent computational complexity somewhat.

In Chapter 4 the performance of the described system is explored using different classification algorithms. Furthermore, the relation between classifier complexity and classification performance is explored. In particular, the two-phase search protocol is analysed experimentally.

Finally, Chapter 5 contains the conclusion to this work. The research questions are reiterated and answers are provided. That chapter also contains pointers for further research to issues that were encountered over the course of creating this work but could not be researched due to time constraints.

2 Architecture

A coarse architecture of a traditional full-text search engine consists of three major components: a *text collector*, an *indexer*, and a *query engine*.

Additionally, a query generator based on a linear classifier is placed on the query interface, called the *classifier-based search* algorithm. This component was newly developed, it is not normally part of a search engine. A detailed explanation can be found in Chapter 3. An overview of the entire process is shown in Figure 2.

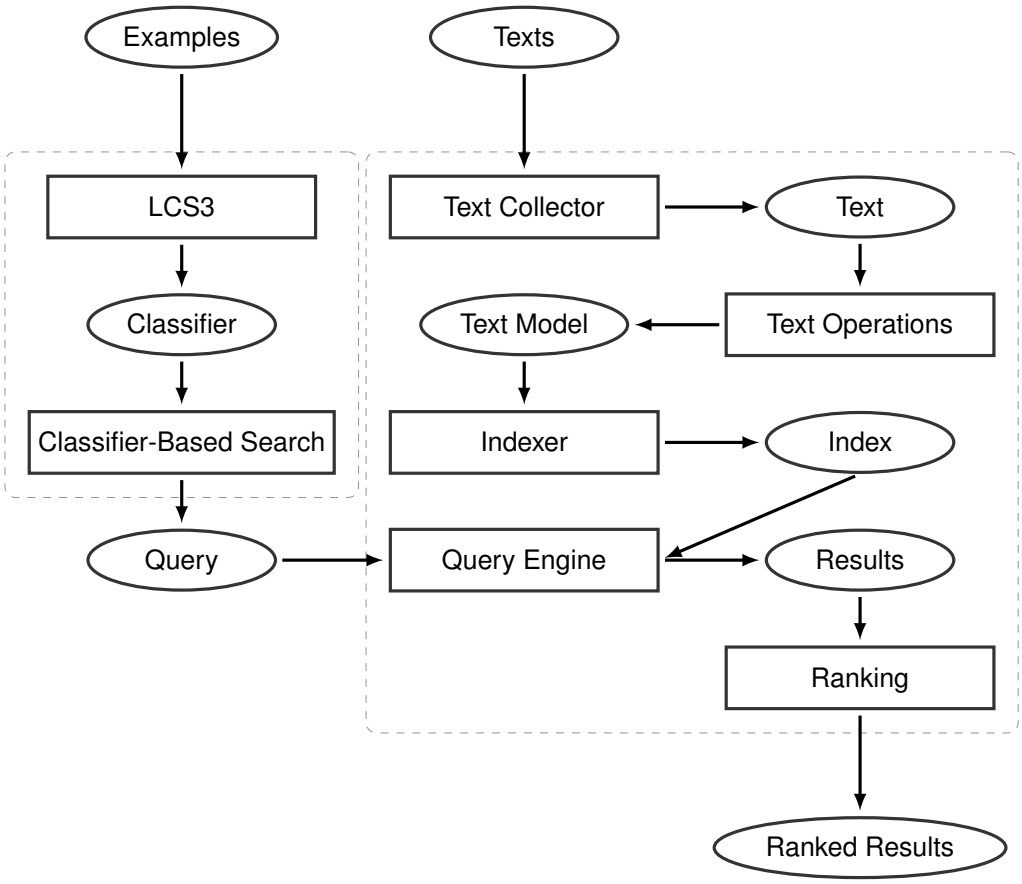

Figure 2.1: CBS attached to a generic full-text search engine.

The text collector runs until no more new data can be found. In case of the World Wide Web it is a web crawler that collects new web pages by following hyperlinks. For the course of the thesis, the text collector will be considered as a black box process, and no detailed explanation will be given. It is only named here for completeness sake.

Newly discovered texts are fed into an indexer. The indexer first preprocesses the data, and will then transform it to the internal data representation for storage in the *index*, according to its model of text. More details on the document model and storage is given in Section 2.1.

Data stored in a full-text index can be retrieved using the query interface. In general, queries will be pre-processed and will be converted to the internal data representation. Using some measure of similarity

between the text in the index and the query, results will be retrieved. Similarity measures are highly dependent on the text model. A small overview of different text models in use are given in Section 2.1.1 with a more detailed explanation of the *Vector Space Model* and the bag-of-words representation of a document.

Chapter 3 gives an elaboration of the final part of the diagram, classifier-based search. It is the main contribution in this thesis.

2.1 Storing and Retrieving Documents

A major component of classifier-based search is document retrieval from a *full-text index*. The following section gives some details and background information on modern full-text indices and their role in current information retrieval [2].

2.1.1 Modelling documents

In order to build a successful document retrieval system it is necessary to have a model of what constitutes a document. For document retrieval the most important part of the model is that it has to convey to a certain extent what the content of the document is *about*. The most simple model of aboutness can be defined as:

> If the word x occurs in the document then the document is about x [4].

Even though examples spring to mind that immediately invalidate this model, it is an often used notion of aboutness. Much research has focused on developing more realistic and more formal models of aboutness [4, 5], but these models offer no substantial advantage for information retrieval yet.

The aboutness model above gives rise to the *bag-of-words* (BOW) model for texts. The aboutness model does not require knowledge of the relations between words present in a document, but merely that if a word is present then the document is about that word. Therefore, a reasonable model for documents, according to this definition of aboutness, is to simply take the set or bag of all words present in the document as its model.

This type of document model becomes cumbersome when creating models for large document collections. Therefore, it is more convenient to explicitly split the document model into a *vocabulary* $\mathscr{V}$, and a vector $\mathbf{d} = (d_1, ..., d_n)$ per document, indicating the presence of words in the document:

$$d_i = \begin{cases} 1 \text{ if } v_i \in \mathscr{V} \text{ is present in document } d \\ 0 \text{ otherwise.} \end{cases} \tag{2.1}$$

A document collection can then be modelled as a separate vocabulary and a large sparse boolean matrix.

Accompanying the bag-of-words model is the Boolean retrieval model. The Boolean model is a simple retrieval model based on set theory and Boolean logic. A user translates his or her information need into a logical expression composed of vocabulary terms and logical operators. In the Boolean model, all documents that that satisfy the query expression are returned to the user.

The Boolean model caters perfectly for professional search, but it has some major disadvantages for users performing ad-hoc search. Users need to provide a relatively precise translation of their information need to the retrieval system, otherwise they might get either too many documents or too few documents. This is a problem when the user does not have a clear understanding of his information need, and consequently, cannot describe it clearly. The model prohibits partial matches, these are not possible.

One way of allowing partial matches is to amend the previous model of aboutness with:

> If the word x occurs more in document a than in document b, then a is more about x than b is about x.

This definition leaves room for ordering results for a query according to some degree of similarity.

This definition also leads to a change in the document model from bag-of-words to *bag-of-weighted-words*, where a weight in its simplest form is the term frequency in a document:

$$d_i = \begin{cases} w_i \text{ if } v_i \in \mathcal{V} \text{ is present in document } d \\ 0 \text{ otherwise.} \end{cases} \tag{2.2}$$

A user can specify his information need by simply giving a list of words that may be present in relevant documents. The degree of similarity between a query and a document is determined by the cosine of the angle between the corresponding numerical vectors:

$$sim(\mathbf{b}, \mathbf{q}) = \frac{\sum_{i=1}^{n} d_i \cdot q_i}{\sqrt{\sum_{i=1}^{n} d_i^2} \cdot \sqrt{\sum_{i=1}^{n} q_i^2}} \tag{2.3}$$

Currently, this model, called the *Vector Space Model*, is the most commonly used information retrieval model. There are various variations on the weighting schemes used for the document vectors and for the query vectors. However, for classifier-based search only the Boolean model is needed, the VSM is merely described in the interest of completeness, therefore, its various extensions and weighting schemes will not be described here.

2.1.2 Inverted File Indexing

It is inefficient to directly use the sparse matrix representation for the document models to answer a search query, because in general they are too large to fit into main memory. Furthermore, using this data structure would require a complete scan of the matrix to retrieve all hits.

Therefore, it is desirable to be able to use query terms to index the data structure. This is called an *inverted-file index*. Instead of creating a list of documents, in which each document number in turn points to a list of occurring terms: the data structure is inverted. A list of terms is created, in which each term

Doc. nr.	**Terms**
0	`example, information, retrieval`
1	`classifier, data, full, index, text`
2	`data, example, text`

points to a list of documents numbers from documents that contain the term:

Term	**Doc. nrs.**
`classifier`	1
`data`	1, 2
`example`	0, 2
`full`	1
`index`	1
`information`	0
`retrieval`	0
`text`	1, 2

This is only a simplified example of an inverted-file index, in general much more information is stored in the index, such as the locations of the terms in the documents, and the global frequencies of the terms. Furthermore, creating such an index for very large text collections is not a trivial task. An extensive overview of more than 40 years of research into inverted-file indexing can be found in [33].

Because classifier-based search is supposed to be an add-on to existing full-text search engines in the first place, the choice was made to not develop a purpose-built indexing system. Instead, the inverted-file indexing system of *Apache Lucene* version 3 [1] was used. Apache Lucene is a high-performance, full-featured text search engine library suitable for nearly any application that requires full-text search.

2.2 Classifying Documents

This chapter gives some details and background information on the supervised learning methodology which is at the basis of document classification, also called *document categorisation*.

2.2.1 Supervised Learning

Some tasks are very expensive or downright impossible to solve using a traditional programming approach. For example, how to tell if a certain person is present in a picture or if a certain document is about a topic. Solving these problems using a traditional approach will entail giving precise specifications on what it means for a document to be about a specific topic, or a precise specification on what it means for a person to be present in a certain picture. In order to solve these kinds of problems it is possible to use an alternative approach, what if the computer can learn the desired input/output behaviour from pre-labelled examples? Then a detailed specification is no longer necessary as long as there are enough labelled examples. This is called the *supervised learning* approach.

For types of problems where supervised learning could be effective there typically exists a relationship mapping the inputs to the outputs in a consistent way, however this relationship may be somewhat obfuscated by noise in the examples. This mapping is called the *target function*. By looking at the provided examples a *learning algorithm* tries to find a mapping that fits, an approximation of the target function. When used for classification this approximation is often called the *decision function*. The accuracy of the decision function is mostly dependent on the amount of training data available, the amount of noise that is present in the examples and the type of learning algorithm that is used.

Not all learning algorithms are created equal, when shown the same examples some algorithms produce a decision function that is closer to the target function than others. These algorithms are said to *generalise* better from the training data. Intuitively, this means that the decision functions at which the algorithms arrive can correctly yield decisions on data that was not part of the training examples.

Generalisation is important because usually the input part of the training examples consist of many *features*, therefore the examples can cover only a small portion of all possible combinations without the training examples becoming prohibitively many or expensive to collect. The data is said to be *sparse* in this case.

2.2.2 Classification

The decision function resulting from a learning algorithm can be used to decide on a label for unseen data. Typically, classification tasks are divided in *binary classification* and *multi-class classification*, however, in this thesis I will describe only a specific subset, which is binary classification using a *linear classifier*. It is common practice to use multiple binary classifiers to get the same effect as using one multi-class learning algorithm.

One way to look at binary classification is that we can use a decision function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ to separate the inputs into two classes. Every input $\mathbf{x} \in X$ will be labelled as positive if $f(\mathbf{x}) \geq 0$, and if $f(\mathbf{x}) < 0$ the input will receive a negative label. When $f(\mathbf{x})$ is a linear function of $\mathbf{x} \in X$ it is called a *linear classifier* with the decision rule $sign(f(\mathbf{x}))$.

[1] `http://lucene.apache.org/`

The decision function of a linear classifier can be written as:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - \theta \tag{2.4}$$

$$= \sum_{i=1}^{n} w_i x_i - \theta, \tag{2.5}$$

where $\mathbf{w} \in \mathbb{R}^n$ are called the weights, and $\theta \in \mathbb{R}$ is called the threshold. Together the weights and the threshold can be used to define a hyperplane splitting X in half-spaces using the equation $f(\mathbf{x}) = 0$, this is shown in Figure 2.2.2. The weights and the threshold can be approximated from the training examples, one algorithm that does this is described in the next section.

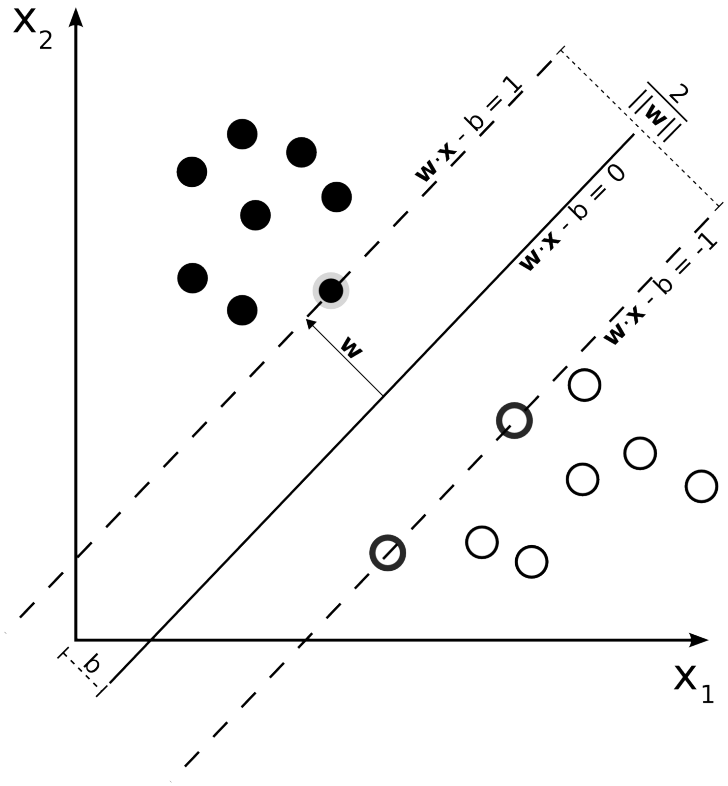

Figure 2.2: The weights and threshold (b) form a separating hyperplane[2].

2.2.3 Finding the weights

One of the earliest methods algorithms for finding a separating hyperplane was the perceptron algorithm developed by Rosenblatt in the fifties [28]. This algorithm is guaranteed to terminate if a separating hyperplane exists but might fail to do so if there is no separating hyperplane. In the following sections three algorithms for finding a separator are discussed in some detail.

These algorithms are selected because they are present in the *Linguistic Classification System* [3] (LCS) that is used for experiments in this thesis. The LCS is a basic component for all applications involving document classification. It has a proven track record [17] in the classification of patent documents.

2.2.4 Balanced Winnow

The family of Winnow algorithms were invented by Littlestone [21, 22] as a method to learn boolean functions from examples. The algorithm itself is surprisingly simple as shown in Algorithm 2.2.1.

[3] `http://www.phasar.cs.ru.nl/LCS`

Algorithm 2.2.1 Pseudocode for balanced Winnow

procedure BALANCEDWINNOW($\{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_m, t_m)\}, \alpha, \beta, \theta^-, \theta^+$)
  $\mathbf{w}^+ \leftarrow (2\theta^+ - \theta^-)/d$
  $\mathbf{w}^- \leftarrow (\theta^+ - \theta^-)/d$
  for $l \leftarrow 1, k$ **do**
    for $i \leftarrow 1, m$ **do**
      $s \leftarrow \sum_{j=1}^{n} (w_j^+ - w_j^-) \cdot x_{ij}$
      if $s \leq \theta^+$ **and** $t_i = 1$ **then**
        $\mathbf{w}^+ \leftarrow \alpha \cdot \mathbf{w}^+$
        $\mathbf{w}^- \leftarrow \beta \cdot \mathbf{w}^-$
      else if $s \geq \theta^-$ **and** $t_i = 0$ **then**
        $\mathbf{w}^+ \leftarrow \beta \cdot \mathbf{w}^+$
        $\mathbf{w}^- \leftarrow \alpha \cdot \mathbf{w}^-$
      end if
    end for
  end for
  return $\mathbf{w}^+, \mathbf{w}^-$
end procedure

Littlestone provided in his paper a proof that Winnow is guaranteed to find a hyperplane that separates the input examples perfectly if such a hyperplane exists. Furthermore, he also showed that the algorithm is robust to noise and irrelevant attributes in the input examples. This robustness is Winnows key strength, Littlestone proved that the upper bound on the number of mistakes shows linear growth in the number of *relevant* attributes and only a logarithmic growth in the total number of attributes. In combination with the fact that the Winnow algorithm makes no assumptions with respect to the (in)dependence between attributes, these properties make it a good candidate for use on text classification.

2.2.5 Naïve Bayes

Currently, the de-facto algorithm used in text classification is *Naïve Bayes*. It is a probabilistic classifier based on the application of Bayes' theorem:

$$P(C = c|D = d) = \frac{P(D = d|C = c) \cdot P(C = c)}{P(D = d)}.$$

The main idea is to calculate the probability that a certain document belongs to a certain class, to find $P(C = c|D = d)$. A document d is then said to belong to class $c \in C$ when the probability of this event is maximal with respect to other classes. More formally this can be written as:

$$\arg\max_{c \in C} P(C = c|D = d) = \arg\max_{c \in C} \frac{P(D = d|C = c) \cdot \ P(C = c)}{P(D = d)}.$$

From this definition it is clear that $P(D = d)$ is irrelevant to the decision, because it is constant when deciding between classes. Thus it can be left out without changing the final classification, yielding the following simpler equation:

$$\arg\max_{c \in C} P(C = c|D = d) = \arg\max_{c \in C} P(D = d|C = c) \cdot P(C = c).$$

Naturally, the true probability distributions are unknown, but they can be reasonably approximated from a set of labelled documents if the probability distribution of these documents resemble the true distribution in the intended application.

The approximation depends on how a document is modelled. In line with the rest of this thesis a document is modelled using the *bag-of-words* method. Therefore, the probability of a document d given a class c equals the joint probability of all terms in the vocabulary V, i.e. $P(D = d|C = c) = P(t_1 \wedge \ldots \wedge t_n|C = c)$. Incorporating this document model into the classification equation yields:

$$\arg\max_{c \in C} P(C = c|t_1 \wedge \ldots \wedge t_n) = \arg\max_{c \in C} P(t_1 \wedge \ldots \wedge t_n|C = c) \cdot P(C = c). \tag{2.6}$$

Thus all that is needed are approximations of $P(C = c)$ and $P(t_1 \wedge \ldots \wedge t_n|C = c)$. The prior probability of any document belong to class $C = c$ can be estimated by counting the training documents that are labelled as belonging to c, so that:

$$P(C = c) = \frac{N(C = c)}{\sum_{c' \in C} N(C = c')}, \tag{2.7}$$

under the assumption that the labelled collection of documents is a random sample from the real probability distribution.

The other approximation is more difficult to come by. By using Bayes' rule repeatedly it is possible to rewrite that into the following:

$$\begin{aligned} P(t_1 \wedge \ldots \wedge t_n|C = c) &= P(t_1|t_2 \wedge \ldots \wedge\ t_n, C = c) \cdot P(t_2 \wedge \ldots \wedge t_n|C = c) && (2.8)\\ &= P(t_1|t_2 \wedge \ldots \wedge t_n, C = c) \cdot P(t_2|t_3\ \wedge \ldots \wedge t_n, C = c) \cdot P(t_3 \wedge \ldots \wedge\ t_n|C = c) && (2.9)\\ &= P(t_1|t_2 \wedge \ldots \wedge t_n, C = c) \cdot \ldots \cdot\ P(t_{n-1}|t_n, C = c) \cdot P(t_n|C = c), && (2.10) \end{aligned}$$

which is expensive to compute. This can be alleviated by the assumption that word occurrence given a class is independent. This assumption is of course false, but it greatly simplifies the approximation, because:

$$P(t_1|t_2 \wedge \ldots \wedge t_n, C = c) = P(t_1|C = c)$$

when $t_1, \ldots, t_n$ are independent. Applying this to Equation 2.10 yields the more convenient:

$$P(t_1 \wedge \ldots \wedge t_n|C = c) = \prod_{i=1}^{n} P(t_i|C = c), \tag{2.11}$$

where $P(t_i|C = c)$ can be estimated with:

$$P(t_i|C = c) = \frac{N(t_i, C = c)}{\sum_{j=1}^{n} N(t_j, C = c)}. \tag{2.12}$$

In this equation $N(t_i, c)$ is a function that yields the number of tokens that occur for type t_i in documents labelled as belonging to class c.

Usually, natural logarithms are used instead of the raw probabilities to counter the numerical errors that occur due to the probabilities being small. In case the class labels are binary it can be shown that Naïve Bayes is a linear classifier too. It is possible to write the Naïve Bayes decision rule in Equation 2.11 in an equivalent manner as Equation 2.5. This can be done by writing the decision rule as:

$$\ln \frac{P(C = c|D = d)}{P(C = \neg c|D = d)} = \ln \frac{P(C = c)}{P(C = \neg c)} + \sum_{i=1}^{n} \ln \frac{P(t_i|C = c)}{P(t_i|C = \neg c)}. \tag{2.13}$$

where $w_i = \ln P(t_i|C = c)/P(t_i|C = \neg c)$, x_i equals the occurrence of a term in a document, and $\theta = -\ln P(C = c)/P(C = \neg c)$.

Surprisingly, despite the false independence assumption Naïve Bayes works well in practice. This counterintuitive empirical result is the subject of much research [27, 19]. Recently, it has been shown that in certain cases, such as text classification, it is the optimal classification algorithm [32], because the results of the false independence assumptions are expected to cancel each other out.

2.2.6 Positive (Negative) Naïve Bayes (PNNB)

Traditional Naïve Bayes requires both positive and negative example documents to compute a classifier. Recently, a new variant of the Naïve Bayes classifier was proposed by Denis et al. [10, 9] which is relevant to Classifier Based Search. The authors adapted the original algorithm so that it could approximate the conditional probabilities from positive training examples and unlabelled examples. This is expected to result in less manual work, because no negative examples have to be labelled by hand. This variant differs in the estimation of the probability distribution of the negative documents, because these are not labelled. However, the decision rule remains the same as shown in Equation 2.13.

The main idea behind this classifier is that in a strictly binary classification it must be the case that the probability of a document being a negative example of a class together with the probability of it being a positive example sums to 1, i.e. $P(C = \neg c) = 1 - P(C = c)$.

The estimation of $P(t_i|C = \neg c)$ is problematic, because there are no negative examples. Therefore, these probabilities have to be estimated from the positive and unlabelled examples only. Denis et al. derived the following equation:

$$P(t_i|C = \neg c) = \frac{N(t_i, \varepsilon) - P(t_i|C = c) \cdot P(C = c) \cdot N(C = \varepsilon)}{P(C = \neg c) \cdot N(C = \varepsilon)}, \tag{2.14}$$

from the relations:

$$P(t_i) = P(t_i|C = \neg c) \cdot P(C = \neg c) + P(t_i|C = c) \cdot P(C = c),$$

and

$$P(t_i) = \frac{N(t_i, C = \varepsilon)}{N(C = \varepsilon)}.$$

After using PNB for classification of unlabelled documents negative examples can be found automatically. However, apart from examining them by hand there is no way to be sure about these labels. Denis et al. describe a modification to their PNB algorithm that lets the algorithm use such negative examples as well. They combine the PNB estimate $P_{PNB}(t_i|C = \neg c)$ with the traditional Naive Bayes estimate $P_{NB}(t_i|C = \neg c)$:

$$P(t_i|C = \neg c) = (1 - \alpha) \cdot P_{PNB}(t_i|C = \neg c) + \alpha \cdot P_{NB}(t_i|C = \neg c), \tag{2.15}$$

where $\alpha = 1/2 \cdot N(C = \neg c)/N(C = c) \cdot P(C = c)/P(C = \neg c)$.

Both variants of Naïve Bayes are sensitive to missing examples in the training data. When the labelled examples for class c do not contain a certain word any real world document of class c that does contain the word will have a probability of 0. This situation occurs due to the multiplication of conditional probabilities in Equation 2.11. To mitigate this effect a constant prior probability is simulated by applying *smoothing*. Usually, *Laplace smoothing* also known as *+1 smoothing* is used for the conditional probabilities of a word given a class label, for example adding the small prior to Equation 2.12 gives:

$$P(t_i|C = c) = \frac{1 + N(t_i, C = c)}{n + \sum_{j=1}^{n} N(t_j, C = c)}. \tag{2.16}$$

3 Classifier-Based Search

Classifier-based search is closely related to document categorisation, in essence it is an attempt to use a linear classifier in reverse. When an organisation has collected a large collection of labelled documents over time it could use this collection to train classifiers to aid in the classification of future unlabelled documents. If a classifier is accurate and is able to label unseen documents, it must somehow be a description of the essence of the class. Therefore, it might be possible to use this description to retrieve all documents that the classifier would label with the class it describes. The end result is no different than simply downloading all documents and using the classifier to label each document, but if only the relevant documents are downloaded and labelled this would save resources.

We start out with a given procedure $query(\{t_1, \ldots, t_n\})$ which returns a list of all documents that are relevant to its parameter. In this case *query* will retrieve its results from a local index, but this could be changed at some point to also work on remote search engines, therefore we prefer to hold on to this abstraction. This procedure will be the basis upon we build an algorithm to retrieve documents relevant to a given linear classifier.

3.1 A solution for small collections

As is often the case, we are looking at a tradeoff between time and space complexities. We can generate exactly n queries of one term only which each would yield a large result set. These sets will later have to be merged into one final set by taking their union. From the final set we can prune all documents that have a score less than the given threshold, θ. In a worst case scenario, the list of hits for a single-termed query comprises the complete universe of documents, something like this would for instance happen when one of the terms in the class profile is a common word such as “the" or “a". If we have an upper limit L to the number of results we can expect this amounts to a worst case space complexity of $O(cL)$, for a small value of c. As long as this fits into memory we could use Algorithm 3.1.1 to locate all relevant documents in the collection.

However, for sufficiently large collections of documents this is not a particularly useful solution. During its runtime the algorithm maintains scores for all documents it has found, regardless of their relevance. For document collections of a more realistic size this may quickly become a problem.

3.2 Towards larger collections

To limit the amount of space that is consumed we can alternatively perform many queries over multiple terms yielding smaller result sets. Each such query is only performed if the terms composing it comprise a *feasible* subset. A feasible subset has the property that the weights of all terms will sum up to at least the threshold. The result sets of the feasible queries can then be merged by taking their union. Furthermore, we will download only a small fraction of irrelevant documents since the accumulated documents are all the result of queries that, at least according to the classifier, may produce relevant results.

Formally, given a linear classifier of the form presented in Equation 2.5 this idea could be modelled as

Algorithm 3.1.1 A solution for small collections

```
1:  procedure FINDDOCUMENTS(w, θ)
2:      n ← Count(w)
3:      H ← Map()                                   ▷ H is a map of docid to score
4:      for i ← 1, n do
5:          H_i ← Query(t_i)                        ▷ H_i is a set of docids for t_i
6:          for j ← 1, count(H_i) do
7:              score ← H.Get(H_t[j])
8:              if score ≠ NULL then
9:                  H.Put(H_t[j], score + w[i])
10:             else
11:                 H.Put(H_t[j], w[i])
12:             end if
13:         end for
14:     end for
15:     Prune(H, θ)                                 ▷ remove items with score <= θ
16:     return H
17: end procedure
```

follows:

$$\text{find all } \mathbf{q} \text{ with } \mathbf{q} \in \{0,1\}^n, \tag{3.1}$$

$$\text{subject to } \sum_{i=1}^{n} w_i q_i > \theta. \tag{3.2}$$

Algorithm 3.2.1 shows one way to get all the relevant queries. In this algorithm only the maximum sum of weights for each document is maintained. The procedure $Bin(i)$ generates a binary representation of its argument. Using the binary representation it is possible to generate a boolean query by interpreting all ones as positive terms and all zeros as negative terms. This is what the procedure *GenQ* does. Clearly, in

Algorithm 3.2.1 Producing all queries

```
1:  procedure FINDDOCUMENTS(w, θ)
2:      n ← Count(w)
3:      H ← Map()                                   ▷ H is a map of docid to score
4:      for i ← 1, 2^n do
5:          q ← GenQ(Bin(i))                        ▷ Bin(i): binary representation of i
6:          nscore ← Dot(w, q)
7:          if nscore > θ then
8:              H_i ← Query(q)
9:              for j ← 1, count(H) do
10:                 H.Put(H_i[j], nscore)
11:             end for
12:         end if
13:     end for
14:     return H
15: end procedure
```

its present form this algorithm is neither optimal nor useful, because it will have to produce 2^n queries. This exponential behaviour quickly becomes too much work for any realistically sized class profile.

However, it does away with the overlap between the result sets by using the full set of terms in each query, if $\mathbf{q}_i = 1$ the term is added to the query otherwise the negation of the term is added to the query. Ultimately, the result is that H contains exactly the relevant results together with the sums of their weights.

3.3 Improvements to this approach

The relation in Equation 3.3 can be exploited to limit the number of queries produced.

$$query(\{t_1,\ldots,t_{n-1},t_n\}) \subseteq query(\{t_1,\ldots,t_{n-1}\}). \tag{3.3}$$

We only care about the result sets of the queries, therefore we can look for queries over the smallest number of terms, that barely sum up to over the threshold when their weights are summed. Due to the inclusion relation any document retrieved by a long query will also be retrieved by a shorter one. It is possible to model "barely" by adding a constraint to the model that sets a maximum value to the sum of weights:

$$\text{find all } \mathbf{q}, \text{ with } \mathbf{q} \in \{0,1\}^n, \tag{3.4}$$

$$\text{subject to } \sum_{i=1}^{n} w_i q_i > \theta, \tag{3.5}$$

$$\neg\exists \mathbf{q}' : \sum_{i=1}^{n} w_i q'_i > \ \theta \wedge \mathbf{q}' \subset \mathbf{q}. \tag{3.6}$$

where $\mathbf{q}' \subset \mathbf{q}$ is taken to mean that $\mathbf{q}'$ is a prefix of $\mathbf{q}$.

When the weights corresponding to the terms in a query sum to over the threshold, the added constraint makes sure that there is no shorter version of the query that also sums to over the threshold. Algorithm 3.3.1 is an implementation of this model. By using this model the ability to list the sum

Algorithm 3.3.1 Producing smallest queries

```
1:  procedure FINDDOCUMENTS(i, n, w, q, θ)
2:      H ← Set()                                   ▷ H is a set of docids
3:      score ← Σ_{i=1}^{n} w_i · q_i
4:      if score > θ then
5:          H ← Query(q)
6:          return H
7:      else if i ≥ n then
8:          return H
9:      else
10:         H ← FindDocuments(i + 1, n, w, q ∪ q_i, θ)
11:         H ← H ⋃ FindDocuments(i + 1, n, w, q, θ)
12:     end if
13:     return H
14: end procedure
```

of weights for each document is lost. These scores are needed later, therefore we need to add a post-processing step in order to calculate the scores for all the retrieved documents.

3.4 Why CBS is hard

Unfortunately, by adding the above constraint the amount of work needed to find the optimal queries is not necessarily reduced. Moreover, even without this extra constraint the problem shows similarities

with a known hard problem, the 0-1 knapsack problem. The 0-1 knapsack problem is defined as follows: given a knapsack with capacity W, a set of n items each with a corresponding weight w_i and a profit p_i, calculate which set of items that will fit in the knapsack will yield a maximal profit. Formally, this can be written as shown in Equation 3.8:

$$\text{maximise} \sum_{i=1}^{n} p_i q_i, \tag{3.7}$$

$$\text{subject to} \sum_{i=1}^{n} w_i q_i \leq W, \tag{3.8}$$

where $p_i, w_i \in \mathbb{R}$ and $q_i \in 0, 1$.

In 1972 Richard Karp [13] proved that this problem along with 20 others is in the class of *NP-complete* problems. This class contains all problems for which solutions can be verified in polynomial time with respect to the input length and which can only be solved in polynomial time on a nondeterministic Turing machine.

The difference between the 0-1 knapsack problem and CBS is that in the latter problem the weights and the profits are equal to each other, $p_i = w_i$. Unfortunately, this does not mean that it is easier to solve this problem, this special case of the 0-1 knapsack problem is known as the *subset-sum problem* and is also in the class of NP-complete problems [14]:

$$\text{maximise} \sum_{i=1}^{n} w_i q_i, \tag{3.9}$$

$$\text{subject to} \sum_{i=1}^{n} w_i q_i \leq W. \tag{3.10}$$

Taking this into account it is unlikely that a linear time or a polynomial time algorithm can be constructed that solves the CBS problem exactly.

When presented with a NP-complete problem it is sometimes enough to find a solution that comes to within a predefined ε of the most optimal solution. In other words it is enough to find an approximation with a predictable error bound. By allowing an approximation it is often possible to construct an algorithm that is far less computationally expensive than the exact solution.

In this case an enumeration of the set of feasible solutions is needed instead of selecting the most optimal one from the set of solutions. There are two possible outcomes for an approximation scheme. Firstly, if solutions are found with a score that is too low we will retrieve too many irrelevant documents. It is not clear if it is possible to say how many extra documents will be downloaded with respect to the error bound. Secondly, if solutions are found with a score that is too high too many queries will be generated. Adding an error bound may even worsen the situation, because it is no longer clear if the result sets will be disjoint.

3.5 2-Phase Classification with Heuristics

There is already a need for a post-processing phase because of the need to have the document scores. If a shorter high-recall classifier is used as the input to CBS and a full fledged classifier is used as the second phase it might obtain a high accuracy as well. Algorithm 3.3.1 can be used as the basis for the high-recall first phase. So we will investigate *two-phase classification* and try to find heuristics to obtain a feasible solution.

3.5.1 A Light Tail

This algorithm can be improved by adding branch-and-bound heuristics to stop descending into branches as early as possible. A first heuristic could be to stop extending a query when the sum of the weights

of the terms that have not been considered by the algorithm yet are not enough to score higher than the threshold value.

This heuristic is implemented in Algorithm 3.5.1 by the *rs* variable. In the initial call of the procedure, *rs* will contain the full sum of weights. For every subsequent call, *rs* will decrease until it is clear that the current score combined with the remaining weights is not enough to score higher than the threshold value.

Algorithm 3.5.1 Heuristics

```
1:  procedure FINDDOCUMENTS(i, n, w, q, θ, rs)
2:      H ← Set()                                   ▷ H is a set of docids
3:      score ← Σ_{i=1}^{n} w_i · q_i
4:      if score > θ then
5:          H ← Query(q)
6:          return H
7:      else if i ≥ n then
8:          return H
9:      else if score + rs ≤ θ then                 ▷ rs is the sum of the remaining w_i
10:         return H
11:     else
12:         H ← FindDocuments(i + 1, n, w, q ∪ q_i, θ, rs − w_i)
13:         H ← H ⋃ FindDocuments(i + 1, n, w, q, θ, rs − w_i)
14:     end if
15:     return H
16: end procedure
```

The effect of this heuristic is dependent on the distribution of weights over the query terms. Typically, only a small fraction of terms are given a significant weight by a linear classifier, these are the *most important terms*. Certain terms achieve a score over the threshold by themselves, such as the first few terms in Figure 3.1.

After this high peak, the graph of term weights typically has a midsection of terms that are more or less equally important. The midsection is the collection of terms that is responsible for most of the computational complexity in the branch-and-bound algorithm.

The final part of the graph typically is a large tail of terms with a very low weight. These terms provide very little retrieval precision in the final result set, but typically produce a high recall. For example, the 20 lowest scoring terms from the graph above are shown in Figure 3.5.1. Most of these terms are words typically found in a stop list, because on their own they do not describe any particular concept or subject.

Luckily, only a small number of terms are actually needed. Experience with term selection [26, 15, 31] has shown that in document classification 99% of all terms can be discarded without loss of classification accuracy.

3.5.2 The Oracle

Figure 3.1 shows another opportunity. Assume that the horizontal line shows the class threshold, and the terms in the profile are sorted so that $w_t \geq w_{t+1}$. If we generate a query with as first positive term t, i.e all terms up to t are given as negative terms, then the line $1/w_i$ denotes the absolute minimum number of positive terms that need to be included in the query from $t+1 \cdot n$ in order for the total weight to be larger than the threshold.

Due to the weight distribution of a typical class profile, the number of terms needed quickly becomes large. Furthermore, it is also known that the probability of a collection of terms co-occurring in a single document decreases when the number of terms in the collection increases.

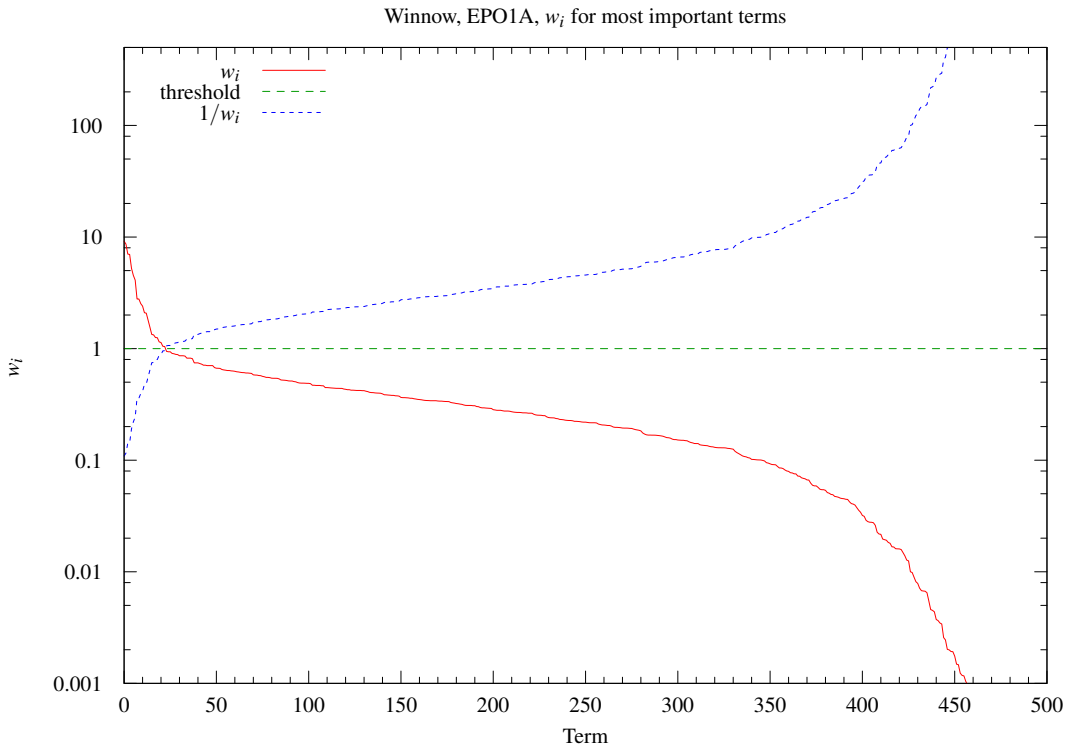

Figure 3.1: Class profile weights for the most important terms

`top, used, along, also, apparatus, being, when, within, two, through, between, be, as, at, are, which, with, an, to, and`

Figure 3.2: The 20 least important terms

This can be exploited by peeking in the index from time to time to see if certain combinations of terms occur together at all. If any of these combinations return an empty result set we no longer have to check any queries containing those combinations.

This is shown in Algorithm 3.5.2, the procedure *Peek* checks whether the positive terms in (q) occur simultaneously in documents anywhere. Only a single example has to be found, and calls can be memoised, therefore on average the call returns quickly.

Algorithm 3.5.2 Heuristics

```
1:  procedure FINDDOCUMENTS(i, n, w, q, θ, rs)
2:      H ← Set()                                   ▷ H is a set of docids
3:      score ← Σ_{i=1}^{n} w_i · q_i
4:      if score > θ then
5:          H ← Query(q)
6:          return H
7:      else if i ≥ n then
8:          return H
9:      else if score + rs ≤ θ then                 ▷ rs is the sum of the remaining w_i
10:         return H
11:     else
12:         if ¬isEmpty(Peek(q)) then
13:             H ← FindDocuments(i + 1, n, w, q ∪ q_i, θ, rs − w_i)
14:             H ← H ⋃ FindDocuments(i + 1, n, w, q, θ, rs − w_i)
15:         end if
16:     end if
17:     return H
18: end procedure
```

4 Experimentation and Evaluation

In order to evaluate the design of classifier-based search, several measurements and experiments were performed. For these experiments several extensions to the Linguistic Classification System were implemented.

Firstly, the multinomial Naïve Bayes Classifier, such as the one described in Section 2.2.5, was implemented in Java for LCS3. In order to make use of the LCS data structures created for the Balanced Winnow decision rule:

$$y(x) = (\mathbf{w}^+ - \mathbf{w}^-) \cdot \mathbf{x} - (\theta^+ - \theta^-) \tag{4.1}$$

the Naive Bayes decision rule in Equation 2.13 was rewritten to fit. This is possible due to the logarithms used in Naïve Bayes.

Secondly, the Positive Naïve Bayes algorithm described in Section 2.2.6 was implemented in Java. The ability to use negative examples was removed after some initial experiments [1], because these initial results showed the same deterioration as obtained in Section 4.2.4.

Finally, a prototype of the classifier-based search algorithm was implemented in Java on top of LCS3. To simulate a search engine the Apache Lucene library was used. Initial experiments using the Google search engine were performed, but it quickly became clear that this was not feasible without the cooperation of Google. After 1000 queries Google would block new requests and show the user a "captcha" each time.

Classifier-based search has been evaluated using two datasets with different characteristics in order to answer the main research question:

> *How to search effectively for maximally relevant texts on a specific topic by means of a linear classifier in a large closed document collection?*

The answers to the supporting questions:

1. What is the relation between the precision and recall of the linear classifier and the precision and recall of Classifier-Based search?

2. How can this relation be used to provide confidence on the expected precision and recall of Classifier-Based search?

3. How to limit the number of documents that must be manually judged by a professional.

4. What is the optimal number of terms for the classifiers?

will be determined experimentally is this chapter.

4.1 Dataset Description

EPO1A is a dataset created by the European Patent Office containing 16000 patent application abstracts. All patent abstracts are written in English. Each document belongs in exactly one class out of 16 possible classes, it is a *mono-classification*. For each class there are exactly 1000 positive examples. Some

Total number of documents	16 000
Number of documents per class	1 000
Total nr. of terms (tokens)	2 004 009
Nr. of unique terms (types)	21 918
Mean nr. of terms / document	143

Table 4.1: Characteristics of EPO1A

pre-processing was performed on the abstracts. Firstly, all terms were converted to lowercase. Secondly, all non-alphabetic terms were discarded. This results in the characteristics shown in Table 4.1.

EPO2F is a collection of full-text patent applications in English. This collection was also created by the European Patent Office. For each of the documents, one up to seven class labels are provided out of a total of 44 labels. This collection is a *multi-classification* where each document is labelled on the average for 1.8 different classes. For each class there is minimum of 2000 positive examples; sometimes there are more examples because documents can have multiple class labels.

Total number of documents	68 418
Total number of examples	88 000
Number of examples per class	2 000
Total nr. of terms (tokens)	387 245 505
Nr. of unique terms (types)	557 788
Mean nr. of terms / document	59 528

Table 4.2: Characteristics of EPO2F

When ranking all terms in the collections according to their term frequencies, it is clear that they adhere approximately to *Zipf's law*. Zipf's law is an empirical law that states that the frequency of each word in a corpus is inversely proportional to its rank in the frequency table.

4.2 Effects of classifier complexity and number of training examples

We need to measure the effect of classifier complexity on classifier performance in order to determine whether it is feasible to do document retrieval using a short classifier.

Furthermore, we need to measure the effect of the number of training examples, both positive and negative, on classifier performance in order to estimate how many documents need to be judged manually by a human operator while still having reasonable classifier performance.

4.2.1 Quality Measures

To measure the performance of a classifier, the standard information retrieval measures of *precision*, *recall* and F_1 will be used. The precision of a classifier is given as:

$$\text{precision} = \frac{|\text{relevant documents}| \cap |\text{retrieved documents}|}{|\text{retrieved documents}|}. \tag{4.2}$$

In other words, precision is the fraction of retrieved documents that are relevant.

Recall is the fraction of relevant documents that were retrieved during a search. Formally, this can be written as:

$$\text{recall} = \frac{|\text{relevant documents}| \cap |\text{retrieved documents}|}{|\text{relevant documents}|}. \tag{4.3}$$

[1] not reported here

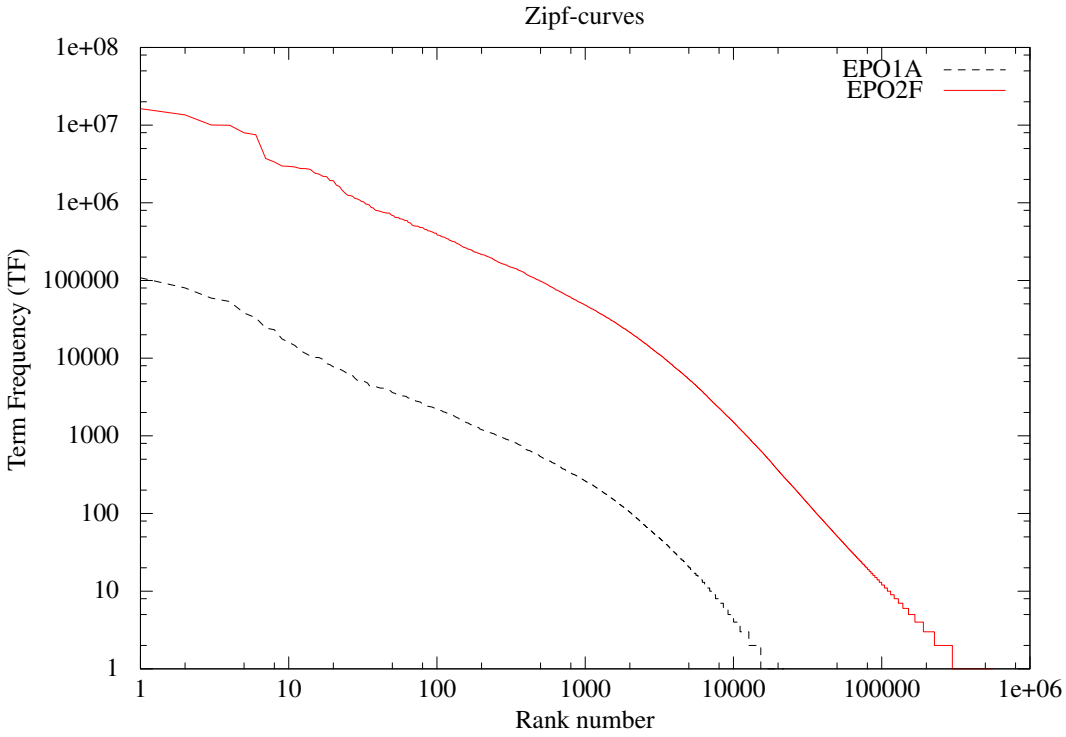

Figure 4.1: Zipf-curves for EPO1A and EPO2F

On its own, recall is not a good measure of quality, because 100% recall rate is trivial to achieve; one simply has to return the entire document collection.

Therefore, it is necessary to have a quality measure that balances precision and recall. This role is typically fulfilled by the F_1 measure. The F_1 measure is a harmonic mean between precision and recall. It can be written as:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{4.4}$$

4.2.2 Baseline with Positive and Negative Examples

We will start by measuring the effect of the number of training examples and the number of profile terms using the default LCS3 classifier, Balanced Winnow. This experiment will be a baseline to compare PNB against. For Balanced Winnow, LTC term weighting and cosine document length normalisation were enabled.

We use the EPO1A data set, split into two. We use 80% of the data set to draw training examples from, and we use the remaining 20% as a validation set. For each category c, the documents not labelled for c are considered an counter example. Unlabelled documents are ignored during training.

4.2.3 Positive Naive Bayes (PNB)

Using the EPO1A dataset a linear classifier is trained using the Positive Naive Bayes algorithm for each class that is present in the dataset. As described in Section 2.2.6 and the implementation notes in Section 4 the algorithm does not need counterexamples. The classifiers assigned either one or zero classes to each document that they were presented with.

After shuffling the dataset it was split in two separate parts, 80% of the corpus was used for training the classifiers, leaving the remaining 20% as test data. This process was repeated to get a 5-fold cross

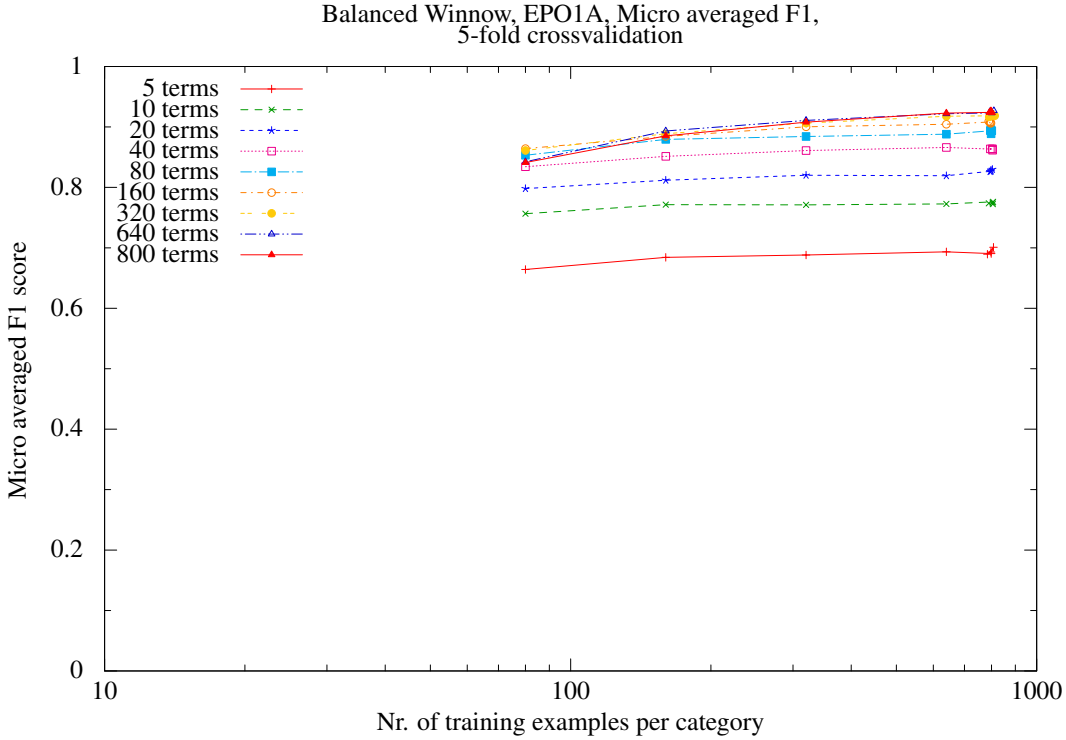

Figure 4.2: The effect of number of training examples and number of profile terms on F1 score

validation of the results. The training data was pre-processed such that only the desired fixed number of labelled examples remained, the other examples were input to the LCS unlabelled.

For the experiment no document length normalisation was performed, and no term-weighting was done, because these have negative effects on Positive Naïve Bayes performance. The positive class probability parameter was derived from the data set and was set to $\frac{1}{16}$. Furthermore, during the experiment Laplace smoothing was applied.

The effect of the number of documents on retrieval performance is shown in Figure 4.3. This figure shows the feasibility of reasonably accurate document classification using positive examples and unlabelled examples only. Furthermore, it shows that Positive Naïve Bayes is able to overcome the noise of known positive examples in the unlabelled training data, which is used to estimate the probability of a document not being in a class.

Notice that the graph for PNB grows much more slowly than seen in the graph for Winnow in the baseline experiment. That is the price paid for not having labelled negative examples. But precisely the absence of negative weights makes PNB suitable for classifier-based search.

4.2.4 Balanced Winnow

The same experiment using only positive training examples was repeated using the Balanced Winnow algorithm. In this case cosine document length normalisation was performed, because it helps Balanced Winnow gain retrieval performance. Furthermore, LTC term-weighting was applied because that also helps Balanced Winnow. Table 4.3 shows the values of the learning parameters given to Balanced Winnow during the experiment.

The result of this experiment is shown in Figure 4.4.

The implementation of the Balanced Winnow algorithm in the LCS uses all documents not labelled for category c as negative examples for the category c, but a fraction of the unlabelled documents are actually positive examples. This resulted in severely deteriorated classification performance, despite

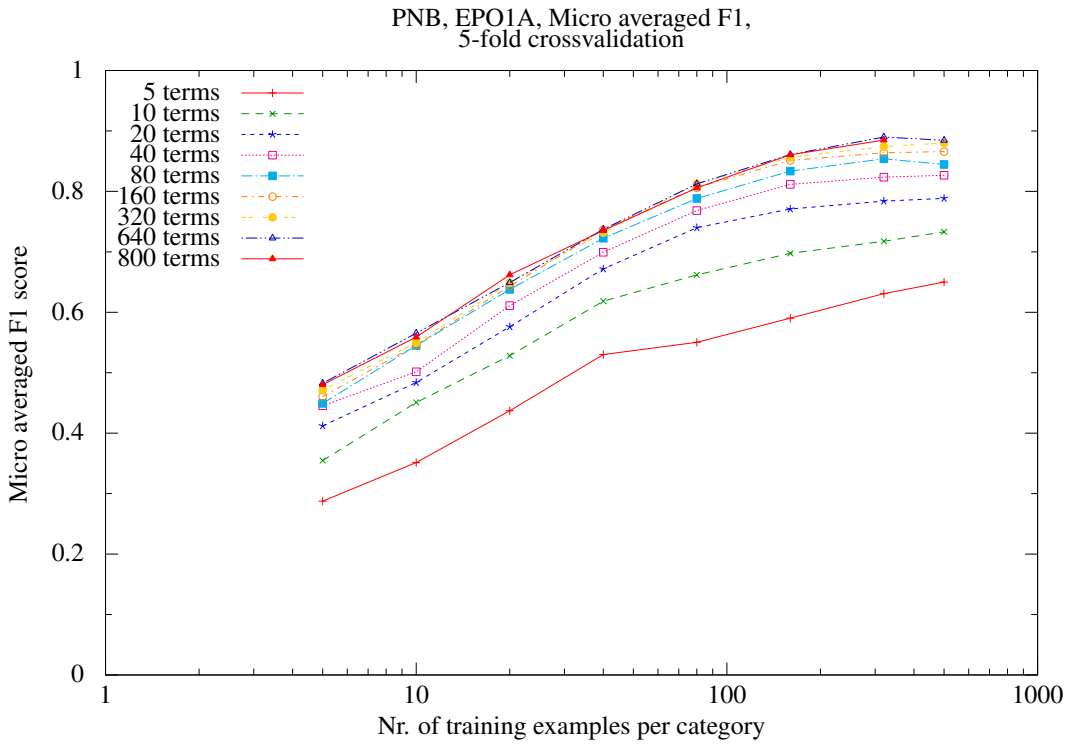

Figure 4.3: The effect of number of training examples and number of profile terms on F1 score

Parameter	Value
Θ^+	2.5
Θ^-	0.5
α	1.05
β	0.95
iterations	3
threshold	1.0

Table 4.3: Values of Winnow's parameters

the fact that at the least 93.75% of the unlabelled examples were negative examples. The classification performance becomes better after a enough documents are trained to minimise the number of wrongly labelled positive examples.

The sudden steep ascend of the graph when enough examples are added is surprising. One possible explanation is that a limited number of positive documents is present. After adding enough examples for training, the fraction of positive examples used as negative examples decreases. Another possibility is that certain positive examples are vital for correct classifier training, and those documents were used as negative examples. Unfortunately, these hypotheses could not be tested in time.

Naturally, this is not a fair testament to the performance that Balanced Winnow normally gives. The need for negative examples makes Winnow unsuitable for classifier-based search. That leads us to the idea of two-phase classification. In the first phase a high-recall classifier is used, which is trained with PNB. The second phase performs a high-accuracy classification with Winnow.

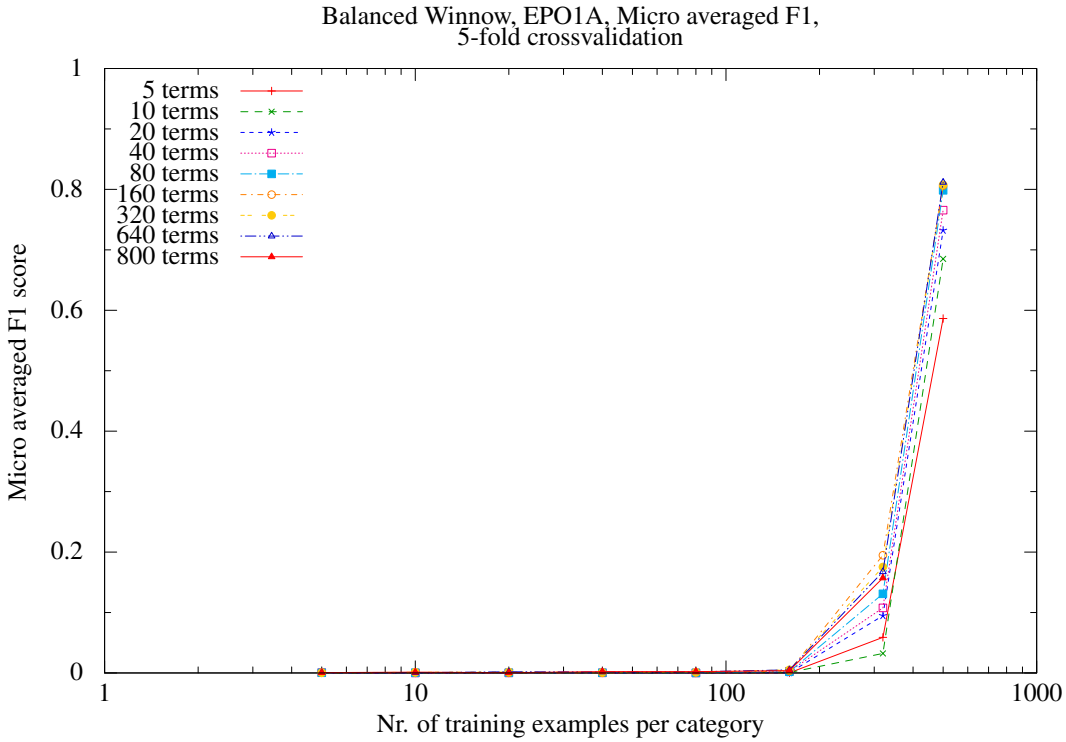

Figure 4.4: The effect of nr. of training examples and nr. of profile terms on F1 score

4.3 Evaluating the two-phase design

In this section some experiments are presented that validate the two-phase design of classifier-based search.

4.3.1 Feasibility of classifier-based search

We have divided the EPO1A collection into a test set and a train set using ten percent of the documents for training and the remaining 90 percent for testing. This is a non-standard method of splitting a data set, the reason is that a human workforce has to label the documents manually in a real-world scenario. Therefore, we have opted to minimise the manual labour.

We used the *Positive Naïve Bayes* algorithm to learn a first phase classifier for each of the 16 classes using the 10/90 split. Exactly one class had to be assigned to each document, i.e. a *mono-classification*. χ^2 *term-selection* was used to fixate model complexity on the predefined values. This type of term-selection is one of the many features of LCS3 and is considered a good term-selection algorithm [26, 31]. It will be considered as a black-box process during these experiments.

The 10% training data was reused to learn the second phase classifier with the *Balanced Winnow* algorithm, using the same parameters as shown in Table 4.3. In this case no term selection was performed, resulting in a classifier length of 4283 terms on average.

Table 4.3.1 shows the results of different sized classifiers, ranging from 10 to 640 selected terms. For each size classifier four rows of results are shown.

1. *CBS (phase 1)* corresponds to the first phase of classifier-based search, using only the short classifier created with PNB.

2. *CBS (phase 2)* corresponds to using both phases of classifier-based search, i.e. the full-text of the documents found during phase one is retrieved and classified using the full-length classifier created using Balanced Winnow.

3. *LCS (short)* corresponds to the performance of the first phase short classifier alone on the full-text of all test documents.

4. *LCS (long)* corresponds to the performance of the second phase long classifier alone on the full-text of all test documents.

	Precision	Recall	F1 measure	Nr. of docs
10 terms				
CBS (phase 1)	0.5193	0.7786	0.6231	1523
CBS (phase 2)	0.9039	0.7531	**0.8216**	1523
LCS (short)	0.5193	0.7786	0.6231	14400
LCS (long)	0.9007	0.8376	**0.8680**	14400
20 terms				
CBS (phase 1)	0.5984	0.8473	0.7015	1708
CBS (phase 2)	0.9360	0.8086	**0.8677**	1708
LCS (short)	0.5984	0.8473	0.7015	14400
LCS (long)	0.9382	0.8396	**0.8862**	14400
40 terms				
CBS (phase 1)	0.5267	0.8631	0.6542	2894
CBS (phase 2)	0.9246	0.8126	**0.8650**	2894
LCS (short)	0.5267	0.8631	0.6542	14400
LCS (long)	0.9251	0.8182	**0.8684**	14400
80 terms				
CBS (phase 1)	0.5451	0.9020	0.6795	4219
CBS (phase 2)	0.9381	0.8263	**0.8786**	4219
LCS (short)	0.5451	0.9020	0.6795	14400
LCS (long)	0.9381	0.8263	**0.8786**	14400
160 terms				
CBS (phase 1)	0.5666	0.9050	0.6969	4598
CBS (phase 2)	0.9629	0.7937	**0.8701**	4598
LCS (short)	0.5666	0.9050	0.6969	14400
LCS (long)	0.9629	0.7937	**0.8701**	14400
320 terms				
CBS (phase 1)	0.6301	0.9188	0.7475	5955
CBS (phase 2)	0.9488	0.8042	**0.8706**	5955
LCS (short)	0.6301	0.9188	0.7475	14400
LCS (long)	0.9488	0.8042	**0.8706**	14400
640 terms				
CBS (phase 1)	0.5837	0.9137	0.7124	7543
CBS (phase 2)	0.9356	0.8031	**0.8643**	7543
LCS (short)	0.5837	0.9137	0.7124	14400
LCS (long)	0.9356	0.8031	**0.8643**	14400

Table 4.4: Classifier-based search on class01 of EPO1A.

The difference in F_1 score of the least complex 2-phase classifier-based search and the most complex 2-phase classifier-based search is remarkably small, at only 4.27%. This little increase in quality comes at the cost of retrieving 4.95 times as many documents. Furthermore, when comparing the least complex 2-phase classifier-based search to using a full length classifier on all documents, CBS has to score about 9.5 times less documents at the cost of 4.27% F_1 score. The end result is that row 2 and 4 only differ significantly in the number of documents classified!

Using a similar 10/90 shuffle for training and testing examples, the same experiment was repeated with the EPO2F data set. This experiment simulates a different environment, it is a multi-classification with much longer texts. The first phase of CBS again is created using the Positive Naïve Bayes algorithm. These short classifiers range from 10 to 640 terms in length.

The second phase classifier was created using the Balanced Winnow algorithm, again using the parameters in Table 4.3. The length of the classifiers was 63920 terms on average. The results are shown in Table 4.3.1.

	Precision	**Recall**	**F1 measure**	**Nr. of docs**
10 terms				
CBS (phase 1)	0.1520	0.5948	0.2421	20883
CBS (phase 2)	0.6359	0.5948	**0.6147**	20883
LCS (short)	0.1085	0.6369	0.1854	61576
LCS (long)	0.6351	0.6358	**0.6354**	61576
20 terms				
CBS (phase 1)	0.2078	0.7473	0.3252	24494
CBS (phase 2)	0.5995	0.7088	**0.6496**	24494
LCS (short)	0.1746	0.7513	0.2833	61577
LCS (long)	0.5992	0.7110	**0.6503**	61577
40 terms				
CBS (phase 1)	0.2623	0.7996	0.3950	29769
CBS (phase 2)	0.6948	0.5598	**0.6201**	29769
LCS (short)	0.2558	0.7996	0.3876	61576
LCS (long)	0.6948	0.5598	**0.6201**	61576
80 terms				
CBS (phase 1)	0.2779	0.8347	0.4170	31738
CBS (phase 2)	0.6609	0.5674	**0.6106**	31738
LCS (short)	0.2742	0.8353	0.4129	61576
LCS (long)	0.6609	0.5674	**0.6106**	61576
160 terms				
CBS (phase 1)	0.2776	0.8790	0.4220	36929
CBS (phase 2)	0.6258	0.6724	**0.6483**	36929
LCS (short)	0.2771	0.8790	0.4213	61576
LCS (long)	0.6258	0.6724	**0.6483**	61576
320 terms				
CBS (phase 1)	0.3053	0.9026	0.4563	41418
CBS (phase 2)	0.5301	0.8188	**0.6435**	41418
LCS (short)	0.3053	0.9026	0.4563	61576
LCS (long)	0.5301	0.8188	**0.6435**	61576
640 terms				
CBS (phase 1)	0.3117	0.9158	0.4651	44220
CBS (phase 2)	0.6469	0.6096	**0.6277**	44220
LCS (short)	0.3116	0.9158	0.4650	61576
LCS (long)	0.6469	0.6096	**0.6277**	61576

Table 4.5: Classifier-based search on dir01 of EPO2F.

This table shows that the 2-phase design also works on for multi-classification problems. Although, the difference between the number of documents scored by classifier-based search and the direct application of the long classifier is not as large, it is still a marked improvement. In a multi-classification it is more difficult to separate two classes, because they might overlap. This might be the reason why the difference between CBS and a direct application of the long classifier is not as large.

The results sometimes show small differences between *CBS (phase 2)* and *LCS (long)*, these are to be expected when using a 2-phase design. The first and second phase classifiers of CBS do not make the same classification mistakes. Two situations change the results from using the long classifier directly:

1. A false positive of the classifier in the second phase might disappear, because the document was not retrieved by the first phase.

2. A true positive of the classifier in the second phase might disappear, because it is a false negative

by the classifier in the first phase.

The second situation decreases overall classification quality, but the first situation increases classification quality. These empirical results suggest that these two situations do not change the results much compared to the direct application of the long classifier.

4.3.2 Influence of the Threshold in the First Phase

The results in Table 4.3.1 were created using the threshold that PNB calculated, namely $\theta = -\ln P(C = c)/P(C = \neg c)$. However, there is also a possibility to vary this threshold to get a different balance between precision and recall. Changes to the threshold have a direct effect on the number of documents that will be retrieved during the first phase of CBS for further inspection.

In order to demonstrate this effect, an experiment was done using the 10, 20, 40 and 80 term classifiers created during the experiment for Table 4.3.1. The CBS algorithm was run again with varying threshold values for the first phase classifier. The effect of the threshold value, θ, on the number of documents that are scored during classifier-based search is shown in Figure 4.3.2. The vertical bar is denotes the threshold

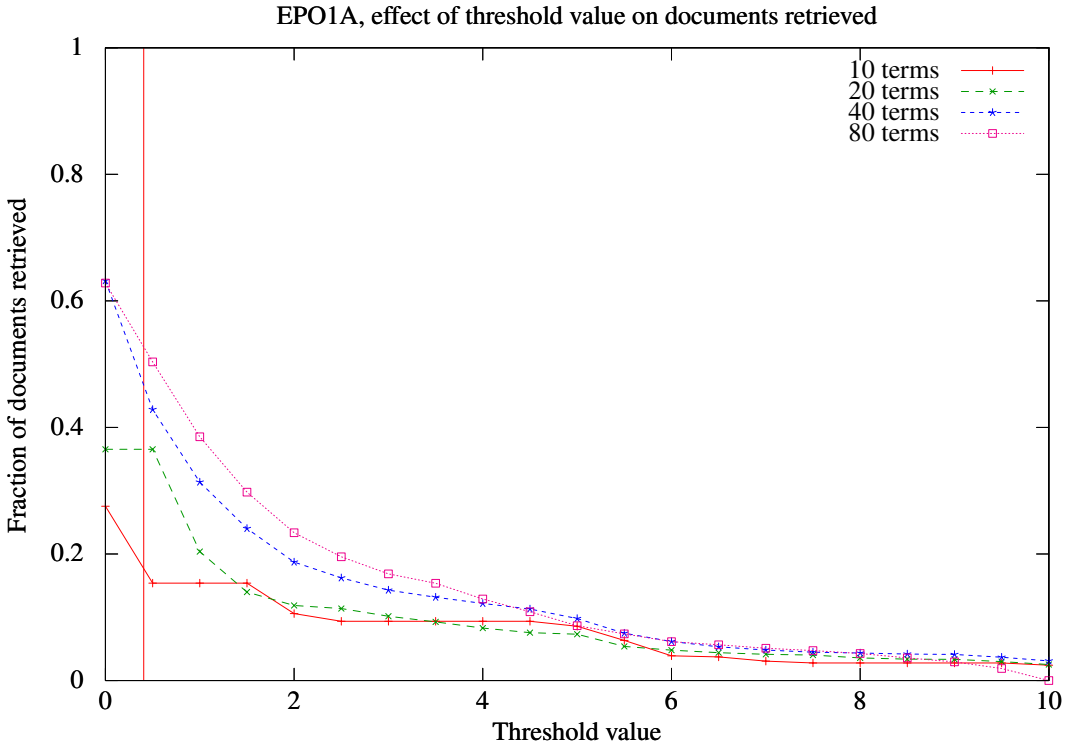

Figure 4.5: The effect of θ on nr. of documents scored.

that LCS calculated itself. In the case of PNB, the threshold is equal to $\theta = -\ln P(C = c)/P(C = \neg c)$.

The effect of the threshold value, θ, on the F_1 score for the 2-phase classifier-based search process is is shown in Figure 4.3.2.

Figure 4.3.2 and Figure 4.3.2 show that increasing the threshold value decreases the number of documents that will be evaluated significantly at a relatively little penalty in quality. This is further reinforced by the view presented in Figure 4.3.2. It shows the effect of threshold value on precision and recall separately for the classifier with 80 terms. Here it is clearly visible what is happening; precision increases so much at the cost of recall that eventually no more documents are retrieved and both precision and recall drop to zero.

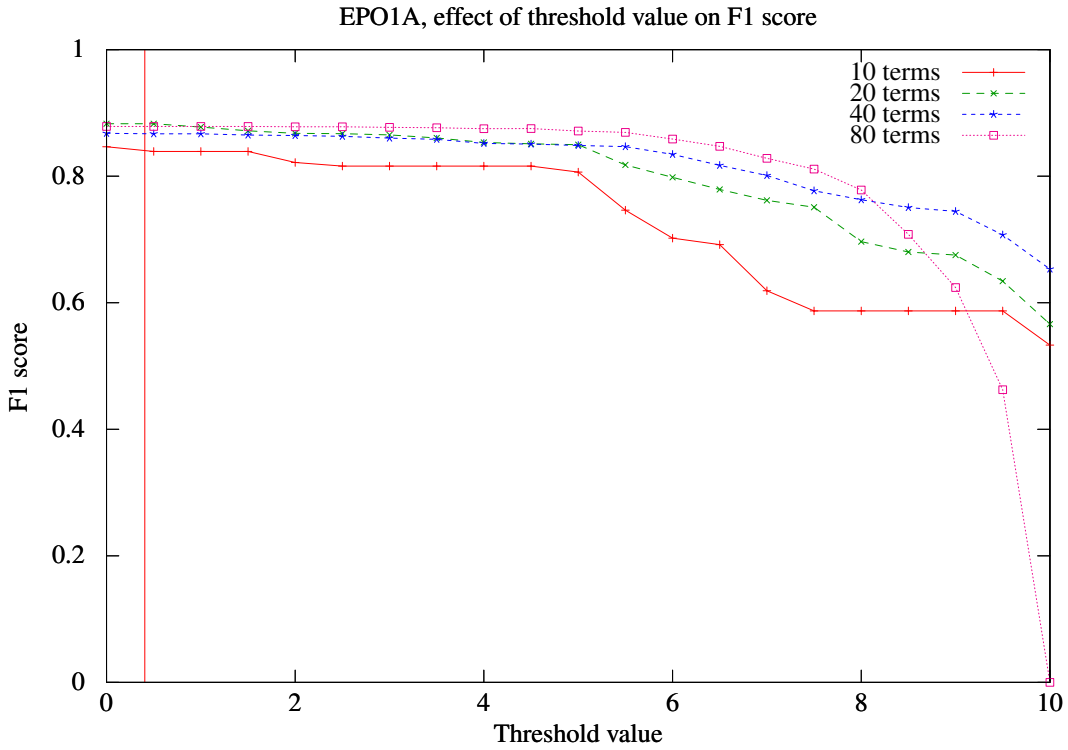

Figure 4.6: The effect of θ on F_1

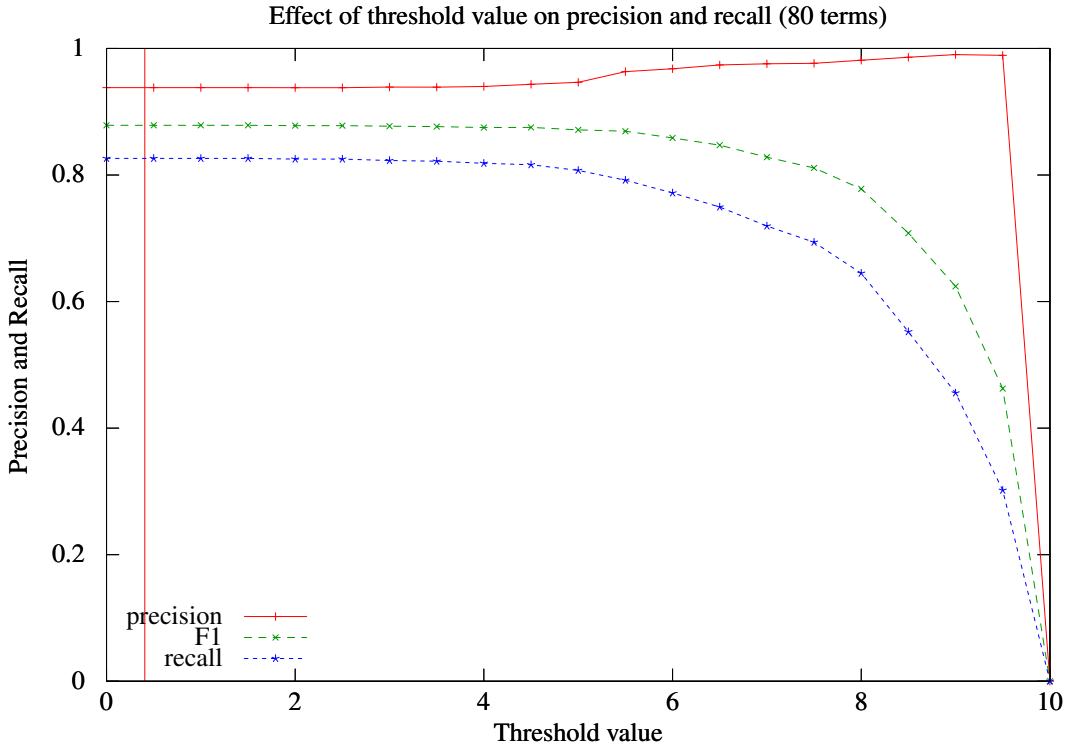

Figure 4.7: The effect of θ on precision and recall

5 Conclusion

In this thesis a design for a *classifier-based search* system was developed. Compared to the state-of-the-art, such a search system enables a user to perform a search using a more refined model for a topic. In turn, this translates to a less labour intensive process in a *professional search* situation.

It was shown that the CBS problem is closely related to the *subset-sum* problem, therefore it is prohibitively difficult to solve directly for a realistically complex linear classifier. The final design utilises a *two-step approach* in order to sidestep this problem. In the first phase, a short *high-recall* classifier is used to find documents that have a high likelihood of being relevant, but with a low number of false negatives. The second phase uses a full-fledged linear classifier with a high accuracy, it will filter out most non-relevant documents. The resulting list of hits can be further processed in a professional search pipeline.

In order to handle linear classifiers of a useful length with reasonable time and space limitations, several heuristics were developed and implemented for the first phase of classifier-based search. These heuristics are mostly dependent on the distribution of the term weights in a classifier, which in turn is dependant on the distribution of terms in typical texts. This makes it likely that these heuristics are specific to the text mining domain.

Furthermore, as a first-phase classifier, *Positive Naïve Bayes* was implemented for use in the CBS system. Besides a large body of unlabelled texts, PNB needs only a small set of labelled relevant texts. This makes the classifier a prime candidate for *bootstrapping* and *active learning* of new sets of labelled documents.

A first prototype of the system was implemented in the LCS, using a high-recall step with PNB followed by a high-accuracy step with Winnow. This was used to validate the claims by means of experimentation on two data sets, EPO1A and EPO2F. The experiments showed that the heuristics worked well enough to be able to handle classifiers of a reasonable length. For example the 640-termed classifier could potentially generate approximately $4.56 \cdot 10^{192}$ queries, but generated merely 12167 queries for EPO1A and 10198 queries for EPO2F.

The search space is limited by the threshold of the linear classifier, higher values of the threshold will translate into more queries evaluated during the search. Consequently, recall will be lower and precision will be higher. Therefore, a human operator has, by means of the threshold, control over the runtime of the system, and over the classifier quality.

Further experimental results showed that the classification performance of a two-phase classifier based search system is closely related to classification performance of the second stage classifier when applied directly on the test set. This property may be used by a human operator to gauge the classifier quality on a small test collection before using the classifier-based search system on a large index. Moreover, it may be used to estimate costs in case of a professional search situation.

The experimental results also showed that recall keeps increasing with a larger number of terms for the first phase classifier. This means that the optimal number of terms for the first-phase classifier must be determined separately for each situation. Judging by the limited results in this thesis, a good starting point is between 80 and 160 terms.

The classifier-based search system could be integrated into an *active learning* system such as the *forensic monitoring* application described in [29]. With CBS at its basis, such a system could be used to rapidly learn new concepts to monitor for.

6 Further Research

Some questions which remained unanswered in this thesis. The results in Section 4.2.4 showed that a small portion of positive examples, wrongly labelled as negatives, would impact the classification quality of Balanced Winnow. It is expected that this will be true for a lot of learning algorithms. In order to build a semi-automated active learning system, it is necessary to have a high precision classifier that does not need negative examples. Another option is to find a method to reliably prune all positive examples from the unlabelled ones with PNB.

Would classifier-based search be any easier if a purpose built index could be used? A preliminary survey suggests that this is not the case, classifier-based search is very closely related to a problem from computational geometry called *half-space range reporting*. The half space range reporting problem [1] can be given as: preprocess a set of n points so that the points inside a query half-space can be reported efficiently. The decision function of a linear classifier defines two half-spaces. Efficient solutions exist for the half-space range reporting problem up to and including three dimensions [1]. However, for typical dimensions encountered in text categorisation there are no efficient solutions, but maybe an approximation [11] can be used.

Bibliography

[1] Peyman Afshani and Timothy M. Chan. 'Optimal halfspace range reporting in three dimensions'. In: *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '09. Society for Industrial and Applied Mathematics, 2009, pp. 180–186.

[2] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN: 020139829X.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 1st Oct. 2007. ISBN: 0387310732.

[4] P. D. Bruza and T. W. C. Huibers. 'A study of aboutness in information retrieval'. In: *Artificial Intelligence Review* 10 (5 1996), pp. 381–407. ISSN: 0269-2821. DOI: 10.1007/BF00130692.

[5] Peter Bruza, Dawei Song and Kam-Fai Wong. 'Fundamental properties of aboutness'. In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1999, pp. 277–278. ISBN: 1-58113-096-1. DOI: 10.1145/312624.312696.

[6] Soumen Chakrabarti, Martin Berg and Byron Dom. 'Focused crawling: a new approach to topic-specific Web resource discovery'. In: *The International Journal of Computer and Telecommunications Networking* 31 (11-16 1999), pp. 1623–1640. ISSN: 1389-1286. DOI: 10.1016/S1389-1286(99)00052-3.

[7] Nello Cristianini and John Shawe-Taylor. *An introduction to Support Vector Machines: and other kernel-based learning methods*. 1st ed. Cambridge University Press, Mar. 2000. ISBN: 0521780195.

[8] Ido Dagan, Yael Karov and Dan Roth. 'Mistake-driven learning in text categorization'. In: *Empirical Methods in Natural Language Processing*. Aug. 1997, pp. 55–63.

[9] Francois Denis, Anne Laurent, Rémi Gilleron and Marc Tommasi. 'Text Classification and Co-training from Positive and Unlabeled Examples'. In: *Proceedings of the ICML 2003 Workshop: The Continuum from Labeled to Unlabeled Data*. 2003, pp. 80–87.

[10] François Denis, Rémi Gilleron and Marc Tommasi. 'Text Classification from Positive and Unlabeled Examples'. In: *Proceedings of the 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. 2002, pp. 1927–1934.

[11] Guilherme D. Fonseca and David M. Mount. 'Approximate range searching: The absolute model'. In: *Computational Geometry: Theory and Applications* 43 (4 2010), pp. 434–444. ISSN: 0925-7721. DOI: 10.1016/j.comgeo.2008.09.009.

[12] Antonio Gulli and Alessio Signorini. 'The indexable web is more than 11.5 billion pages'. In: *Special interest tracks and posters of the 14th international conference on World Wide Web*. ACM, 2005, pp. 902–903. ISBN: 1-59593-051-5. DOI: 10.1145/1062745.1062789.

[13] Richard M. Karp. 'Reducibility Among Combinatorial Problems'. In: *Complexity of Computer Computations* (1972), pp. 85–103.

[14] Hans Kellerer, Ulrich Pferschy and David Pisinger. *Knapsack Problems*. 1st ed. Springer, Oct. 2004. ISBN: 3540402861.

[15] Cornelis H.A. Koster and Jean Beney. 'On the Importance of Parameter Tuning in Text Categorization'. In: *Perspectives of Systems Informatics*. Vol. 4378. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, pp. 270–283. DOI: 10.1007/978-3-540-70881-0_24.

[16] Cornelis H.A. Koster, Nelleke H.J. Oostdijk, Suzan Verberne and Eva K.L. D'hondt. 'Challenges in Professional Search with PHASAR'. In: *Proceedings of DIR 2009*. 2009, pp. 101–102.

[17] Cornelis H.A. Koster, Marc Seutter and Jean Beney. 'Multi-classification of Patent Applications with Winnow'. In: *Perspectives of System Informatics*. Vol. 2890. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, pp. 111–125. DOI: 10.1007/978-3-540-39866-0_53.

[18] David D. Lewis. 'A sequential algorithm for training text classifiers: corrigendum and additional data'. In: *SIGIR Forum* 29.2 (1995), pp. 13–19. ISSN: 0163-5840. DOI: 10.1145/219587.219592.

[19] David D. Lewis. 'Naive (Bayes) at forty: The independence assumption in information retrieval'. In: *Machine Learning: ECML-98*. Vol. 1398. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1998, pp. 4–15. DOI: 10.1007/BFb0026666.

[20] David D. Lewis and William A. Gale. 'A sequential algorithm for training text classifiers'. In: *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, 1994, pp. 3–12. ISBN: 038719889X.

[21] Nick Littlestone. 'Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm'. In: *Machine Learning* 2.4 (1988), pp. 285–318. DOI: 10.1023/A:1022869011914.

[22] Nick Littlestone. 'Mistake bounds and logarithmic linear-threshold learning algorithms'. PhD thesis. Santa Cruz, CA, USA, 1989.

[23] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN: 978-0521865715.

[24] Filippo Menczer. 'ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery'. In: *Proceedings of the 14th International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 227–235. URL: http://informatics.indiana.edu/fil/Papers/ICML.ps.

[25] Gautam Pant and Padmini Srinivasan. 'Learning to crawl: Comparing classification schemes'. In: *ACM Transactions on Information Systems* 23 (4 2005), pp. 430–462. ISSN: 1046-8188. DOI: 10.1145/1095872.1095875.

[26] Charles Peters and Cornelis H.A. Koster. 'Uncertainty-Based Noise Reduction and Term Selection in Text Categorization'. In: *Advances in Information Retrieval*. Vol. 2291. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2002, pp. 248–267. DOI: 10.1007/3-540-45886-7_17.

[27] Jason D. M. Rennie, Jaime Teevan and David R. Karger. 'Tackling the Poor Assumptions of Naive Bayes Text Classifiers'. In: *In Proceedings of the Twentieth International Conference on Machine Learning*. Jan. 2003, pp. 616–623.

[28] Frank Rosenblatt. 'The perceptron: a probabilistic model for information storage and organization in the brain'. In: *Psychological review* 65.6 (Nov. 1958), pp. 386–408. ISSN: 0033-295X.

[29] Jelle J. P. C. Schühmacher and Cornelis H. A. Koster. 'Signalling Events in Text Streams'. In: *User Centric Media*. Vol. 40. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2010, pp. 335–339. ISBN: 978-3-642-12630-7. DOI: 10.1007/978-3-642-12630-7_42.

[30] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, 12th May 2005. ISBN: 0321321367.

[31] Yiming Yang and Jan O. Pedersen. 'A Comparative Study on Feature Selection in Text Categorization'. In: *Proceedings of the Fourteenth International Conference on Machine Learning*. ICML '97. Morgan Kaufmann Publishers Inc., 1997, pp. 412–420. ISBN: 1-55860-486-3.

[32] Harry Zhang. 'The Optimality of Naive Bayes'. In: *FLAIRS Conference*. AAAI Press, 2004. URL: http://www.aaai.org/Papers/FLAIRS/2004/Flairs04-097.pdf.

[33] Justin Zobel and Alistair Moffat. 'Inverted files for text search engines'. In: *ACM Computing Surveys* 38.2 (2006), p. 56. ISSN: 0360-0300. DOI: 10.1145/1132956.1132959.