Effective and Efficient Classification on a Search-Engine Model

Aris Anagnostopoulos[*]
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089
aris@cs.brown.edu

Andrei Z. Broder[*]
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089
broder@yahoo-inc.com

Kunal Punera[*]
Dept. of Electrical and
Computer Engineering
University of Texas at Austin
Austin, TX 78751
kunal @ ece.utexas.edu

ABSTRACT

Traditional document classification frameworks, which apply the learned classifier to each document in a corpus one by one, are infeasible for extremely large document corpora, like the Web or large corporate intranets. We consider the classification problem on a corpus that has been processed primarily for the purpose of searching, and thus our access to documents is solely through the inverted index of a large scale search engine. Our main goal is to build the "best" short query that characterizes a document class using operators normally available within large engines. We show that surprisingly good classification accuracy can be achieved on average over multiple classes by queries with as few as 10 terms. Moreover, we show that optimizing the efficiency of query execution by careful selection of these terms can further reduce the query costs. More precisely, we show that on our set-up the best 10 terms query can achieve 90% of the accuracy of the best SVM classifier (14000 terms), and if we are willing to tolerate a reduction to 86% of the best SVM, we can build a 10 terms query that can be executed more than twice as fast as the best 10 terms query.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Measurements

Keywords

Text Classification, Search Engine, Feature Selection, Query Efficiency, WAND

[*]This work was done while the author was at IBM Research.

1. INTRODUCTION

Automatic document classification is one of the fundamental problems of Information Retrieval and has been amply explored in the literature [13, 16, 17, 18, 23]. Applications include the classification of emails as spam or not, or into different mail folders, automatic topic classification of news articles, and so on. In most existing classification frameworks, including the ones just mentioned, the trained classifier is applied to instances being classified one by one, predicting a class label for each. This method of deployment is useful when these documents are not all available at the same time and works well when only a small number of documents are to be classified. However, consider the scenario where we want to use a classifier to find a relatively small set of on-topic documents from a corpus of a *billion* documents. Here, the standard method of applying the classifier one by one to each document in order to determine whether it is on topic is impractical, and especially so when new classification problems appear frequently.

In this paper we show how to efficiently and effectively classify documents in a large-scale corpus that has been processed primarily for the purpose of searching. Thus, our methods access documents solely through the inverted index of a large-scale search engine. Here, our main goal is to build the "best" query that characterizes a document class (topic), which can then be used to retrieve the documents in the positive set from the search engine. This index query runs in time proportional to the size of the result set ("output-sensitive"), and independent of the corpus size.

Apart from the efficiency considerations mentioned above, there are also several other reasons for taking this approach:

- Most previous classification frameworks assume that the documents are pre-processed (scanned, parsed, reduced to a bag of words, etc.) and represented (as term vectors, within a low rank approximation, etc.) for the express purpose of classification. For web search engines as well as large corporate search engines maintaining separate representations (one each for search and classification) is difficult and expensive.

- New classification problems might appear at any moment driven by regulatory or business needs. (Find all "internet pharmacy" sites, find all car opinion sites, find all pages that discuss bypassing a digital rights management scheme.) Such problems are naturally handled by our approach. All we need is a training set after which we can use fairly standard interfaces.

- Once a class has been characterized via a query, one can rerun the query whenever the corpus changes. In fact, the major Web search engines (Google, Microsoft, Yahoo!) provide such "alert" service that notify users of changes in the result set of a standing query. However the query is fixed and designed by a user; here we envisage a query produced by a learning system that can capture a concept.

- Several search engines such as Vivisimo, Ask, Kosmix, etc. cluster the top-k search results before displaying them to the user. As a further enhancement, the search engine could let the user select one or many clusters as the appropriate context for her query, and then expand these selected clusters to retrieve further results. Our approach can be used to convert clusters into queries to directly address this scenario.

There are, of course, also trade-offs: in order to build the "best" query that characterizes a class, we need to balance *effectiveness* (what precision/recall is achievable) against *efficiency* (how fast can this query be executed) and *implementation complexity* (what capabilities are required from the underlying search engine: term weights, negated terms, etc.). Our classification approach is inherently limited to the features normally indexed by search engines and requires a modest amount of access to the internals of the underlying search engine (basically, we need to be able to explicitly set weights on query terms, and for further optimization we need to know the length of posting lists). However, as we show below, with respect to this feature set and these constraints, one can perform very well.

1.1 Contributions

In this paper we discuss various ways to create queries to retrieve documents that belong to a concept. We investigate several classification methodologies in order to create a representative query that can be easily implemented within a search engine framework. In particular, we consider formulae-based approaches such as Naive Bayes classification and techniques from relevance feedback in information retrieval. We compare our approach with more advanced machine-learning methods (Support Vector Machines), and we show that our approach is very effective. We select the terms for inclusion in the query by applying several feature-selection techniques. To further increase the efficiency, we investigate the factors that determine the running time and we incorporate them into our query-creation framework.

We show that surprisingly good classification accuracy can be achieved on average over multiple classes by queries with as few as 10 terms. Moreover, we show that optimizing the efficiency of query execution by careful selection of these terms can further reduce the query costs. More precisely, we show that on our data set, with our method, the best 10-terms query can achieve 90% of the accuracy of the best SVM classifier (14000 terms), and if we are willing to tolerate a reduction to 86% of the best SVM, we can build a 10-terms query that runs more than twice as fast as the best 10-terms query.

In summary, we make the following contributions:

- We introduce a framework for performing classification by making use of the existing search-engine technology.

- We show that by selecting only a few terms (fewer than 10) we can achieve classification accuracy comparable to the best possible.

- We perform a study to estimate the execution time of a query as a function of the terms included. This study might be of independent interest. We adapt our query construction to use this information to achieve a good trade-off between effectiveness and efficiency.

In Section 2 we present some past work on similar problems and we examine how our work is related. We define the basic problem and our solution to it in Section 3. In Section 4 we show how we apply these ideas and present an empirical evaluation. Finally, we conclude the paper in Section 5.

2. RELATED WORK

In this section we review some related work in the areas of machine learning for automatic text classification, and relevance feedback and query modification in information retrieval. Later in the section we provide a comparison of our work with the prior art.

2.1 Automatic Text Classification

Automatic text classification is a heavily studied area in machine learning and numerous techniques have been proposed for the problem [16, 17]. Of these, Support Vector Machines (SVMs) have been shown to be especially effective [13, 18]. Multinomial Naive Bayes (NB) [15], although simple and efficient, has also been observed to be effective. Our approach in this paper relies on the use of both SVM and NB classifiers and we describe them in brief in Section 3.4. Moreover, we use SVMs with linear kernels to compute the best text classification performance achievable with the full set of features (see Section 4.3).

2.2 Relevance Feedback and Query Modification

The idea of modifying a user query in order to increase precision and recall is a very old one in the field of information retrieval. One of the most popular strategies is *relevance feedback*, where the user identifies the relevant results returned by a query. The system then uses this information to create a new query that will potentially retrieve more documents that are related to those that the user has marked as relevant and, therefore, might be of interest to him. The main two modifications to a query are the *query expansion*, in which terms from the relevant documents are added to the query, and *term re-weighting*, in which the weights of the terms in the original query are modified based on the user's selections. In Section 3, we review *Rocchio's algorithm* [21] and *RTFIDF* that are popular relevance feedback mechanisms for the *vector model* and *probabilistic model* [20] respectively. Both these query weighting schemes are used in our approach in this paper. For details about modeling in information retrieval and about relevance feedback, we refer readers to the book of Baeza-Yates and Ribeiro-Neto [1].

Chang and Hsu [5] combine document clustering and relevance feedback ideas to address the problem of providing coarse-grained feedback. They build a meta search-engine, which operates as follows. First it accepts an initial query from the user which it submits to several commercial search engines. The ensemble of the collected documents is clustered and presented to the user, who specifies the clusters that are relevant to him. Finally, the original query is expanded and re-weighted by applying a method similar to Rocchio's algorithm.

The same motivation is behind the work of Glover et al. [10]. The authors use a Support Vector Machine (SVM) classifier with a Gaussian kernel to create a query that can fetch documents of a specific type (e.g., user homepages). The goal is to build a meta search-engine that creates queries for different search engines and combines the results. This work is extended in [8], where further techniques are employed on top of SVMs to increase recall.

2.3 Comparison with Our Work

There are many differences in the motivation, problem setting, and approach of our work from prior art. The primary difference from existing work on text classification is our equal emphasis on efficiency as well as effectiveness of classification. We discuss methods for construction of queries that are not only accurate but can also be executed very fast. Another important difference is that in our problem setting the classifier does not have just black-box access to the search engine, but instead can access and make use of internal information such as sizes of posting lists. Moreover, we use the **WAND** primitive, described in Section 3.2.1, which allows the use of weights on query terms. As shown in Section 4, the availability of these two additional capabilities helps us design queries that are extremely efficient and accurate. Finally, our primary goal in this paper is to show that a search engine model can be used to perform classification on very large corpora in "output-sensitive" time with extremely little loss in accuracy.

3. APPROACH

In this section we formally define the problem and describe our approach towards tackling it.

3.1 Problem Definition

In our model a document can be viewed as a vector $\vec{d} \in D$ of features, where D is the feature space in which every document belongs. These features could be words, bi-grams, and other information that the search engine might index about the page, such as elements from its HTML structure. For concreteness, we also assume that the feature-to-document mapping is organized as an inverted-index, which is the standard way most search engines in the web store their information. However, our approach can be applied to different models. Every document is also assigned a class label which, without loss of generality, we call *positive* ($\oplus$) or *negative* ($\ominus$). We are given a train set $D_{\text{train}} \subset D$ and a disjoint test set $D_{\text{test}} \subset D$. For every document $\vec{d_i}$ in the train set D_{train}, we are given its label (class) $\ell_i \in \{\oplus, \ominus\}$. Our goal is to obtain a query $\mathcal{Q} : D \to \mathbb{R}$ to the reverse index (of documents in D_{test}) which maximizes the number of documents of the positive class that are ranked higher than the documents of the negative class.

This problem is reminiscent of statistical text classification in which we learn a classification function $\mathcal{F} : D \to \{\oplus, \ominus\}$ that assigns a class label to each document. This function is used to classify the test set of documents D_{test}. The objective of learning is to obtain a classification function $\mathcal{F}$ which maximizes the number of positive (negative) testing instances that are assigned positive (negative) labels.

In the current scenario we have to operate under three potential constraints.

1. Since we are using the search-engine model, we would like to limit the size of the queries and this leads to the problem of selecting query terms.

2. We would like to optimize the selection of terms for query execution time as well as accuracy.

3. In certain applications (such as query refinement via concept learning) the query has to be constructed in real time. In such cases we might prefer *one-shot* (formulae-based) learning approaches as opposed to iterative and more expensive procedures.

The goal of this paper is to study these and other issues associated with the use of a search-engine model for classification.

3.2 Classification via Querying

As we mentioned, we want to make use of the existing technology in text search in order to classify documents that have been indexed by a search engine. In our approach we obtain the text search query by first learning a classifier for the positive concept and then converting the classifier into a search-engine query.

In this paper we consider search engines based on posting lists and document-at-a-time evaluation. To our knowledge, all popular Web search engines, for example, Google, Yahoo!, Inktomi, AltaVista, and AllTheWeb, belong to this class. For most commercial search engines, the query format is a boolean expression involving words as terms, but might also include additional operations, such as containment of phrases, proximity operations, and so on.

In order to obtain a query from a classifier we have to make some choices. The first decision involves whether the query should have weights for the terms or whether it should be boolean. Since we are going to use the query to retrieve documents in the positive class we expect a query with term weights to better approximate the decision boundary. Later in this section we show a primitive called **WAND** [2], which can be used to efficiently execute a weighted query over a traditional search engine.

The second decision that we need to make is whether to limit ourselves to the use of linear classifiers. In general, linear classifiers are faster to train and more interpretable than non-linear ones. This is true especially in the context of document classification, where the features used in linear classifiers correspond well to the terms used in the queries. Moreover, in high-dimensional domains such as text classification linear features suffice to find an effective discriminating boundary and provide high-quality results [18, 23]. For these reasons we limit this study to linear classifiers. Note that using **WAND** and the two-level retrieval process outlined in [2] allows us to efficiently implement nonlinear classifiers as well. However, we do not provide any more details in this paper.

3.2.1 *The WAND Operator*

Here we briefly describe the **WAND** operator that was introduced in [2] as a means to optimize the speed of weighted search queries. **WAND** stands for Weak **AND**, or Weighted **AND**. The **WAND** operator takes as arguments a list of

Boolean variables $X_1, X_2, \ldots, X_k$, a list of associated positive *weights*, $w_1, w_2, \ldots, w_k$, and a threshold θ. By definition, $\mathbf{WAND}(X_1, w_1, \ldots X_k, w_k, \theta)$ is true iff

$$\sum_{1 \le i \le k} x_i w_i \ge \theta, \tag{1}$$

where x_i is the indicator variable for X_i, that is

$$x_i = \begin{cases} 1, & \text{if } X_i \text{ is true} \\ 0, & \text{otherwise.} \end{cases}$$

With X_i indicating the presence of query term t_i in document d, the **WAND** operator is used to return the set of documents satisfying Equation (1).

This is the original version of **WAND** that appeared in [2], and making use of it has as a result the feature space being $\{0,1\}^m$ (m is the total number of features (terms)), which only exploits information of whether a word exists in a document or not. Since we want to work on the richer feature space $\mathbb{R}^m$, which contains frequencies of words in documents, we set

$$x_i = \text{frequency of term } t_i.$$

A second generalization of **WAND** in this work is that we allow the weights of the terms to be negative; this increases the power of our query since now we can use terms that are negatively correlated with the target user concept to avoid retrieving off-topic documents.

3.3 Selecting the Query Terms

Having presented the mechanism with which we will execute our queries, we now present the mechanism for creating them. First, we have to answer the question: *How many terms should a query contain*? In general, a query with more terms retrieves more accurate results. The downside is that the cost of the query increases with its size. There is an inherent trade-off between query cost and retrieval accuracy, and the above question can be reframed as negotiating this trade-off.

Given that we cannot execute a query with all available terms and weights, the second question that we need to answer is: *Which terms should we choose*? In this section we suggest approaches to tackle both these questions. We describe ways to measure the quality of terms, and discuss further factors that might affect the query cost versus retrieval accuracy trade-off.

Note that the problem of selecting terms for the query is essentially the same as the problem of feature selection, which has been widely studied (see [4, 24]) in the context of text classification. The key difference lies in the fact that traditional feature-selection techniques emphasize the preservation of the accuracy of classification with the fewest possible features. In our work, we jointly want to optimize the processing time of a query along with its accuracy. In the rest of this section, we present our approach for achieving this goal.

3.3.1 Term Selection for Query Accuracy

In this section we present measures that score each term in the vocabulary on their impact on retrieval accuracy. Based on this scoring, the top-k terms are greedily selected to form the query. The weights of these selected terms in the **WAND** query are then calculated by the chosen classifier or formula. Below we present three ways to compute the *term-accuracy-scores*.

1. **Information Gain:** In the past, *information gain* (IG) has been used as a measure of saliency of features in text classification tasks [17, 24]. IG is the number of bits of information obtained for predicting the class of a document based on the presence or absence of a term:
$$\text{IG} = H(\ell) - H(\ell \mid t)$$
where $H(\ell)$ is the entropy of the class label, and $H(\ell \mid t)$ is the entropy of the label conditioned on the presence or absence of the term t.

2. **Fisher Index:** Chakrabarti et al. [4] introduced a measure called *Fisher Index* (FI) score which calculates how well a term separates the mean vectors of the positive and negative document classes. The FI score of a term is the ratio of the between-class to within-class scatter:
$$\text{FI}(t_i) = \frac{(\mu(\oplus, t_i) - \mu(\ominus, t_i))^2}{\sum_{c \in \{\oplus, \ominus\}} \frac{1}{|C_c|} \left(tf(c, t_i) - \mu(c, t_i)\right)^2},$$
where $\mu(c, t_i) = \frac{1}{|C_c|} \sum_{\vec{d} \in C_c} tf(\vec{d}, t_i)$, $tf(\vec{d}, t_i)$ is the frequency of term t_i in document $\vec{d}$, $tf(c, t_i)$ the frequency of term t_i in class c, and C_c the set of documents belonging to class c.

3. **Coefficient in the Linear Boundary:** An alternative way to construct the query is to select terms that have high absolute coefficient values in the linear decision boundary obtained by the classifiers in Section 3.4. These are the terms that have the most influence on deciding which class a document containing them belongs to. Also, these terms are good candidates as they are often contained in a large fraction of the positive documents in the inverted index, which has a positive effect on the recall of the set of positive documents.

3.3.2 Term Selection for Query Efficiency

The processing time to execute a query depends on the way the search engine stores the documents and processes the queries, being roughly proportional to the number of disk accesses performed while evaluating the query. While the details are specific to each search engine, the time generally depends on the number of terms and their individual characteristics. We confirm this fact in Section 4.6 where we report on experiments on estimating a formula that relates the query processing time in our search engine as a function of the terms included. Our experiments show that the query processing time is essentially proportional to the number of terms and the lengths of their posting lists.

As mentioned earlier, there exists a trade-off between the accuracy achievable with a query and its processing time. Hence, while constructing the query we would like to include terms that have both a high term-accuracy-score and a small postings list. In order to achieve this we normalize the term-accuracy-score by the size of the postings list. Specifically, we select terms in the decreasing order of the *term-selection-score* value

$$\textit{term-selection-score} = \frac{\textit{term-accuracy-score}}{(\textit{postings-list-size})^{\alpha}} \tag{2}$$

where α is used to trade-off between accuracy and efficiency. A higher value of α will prefer terms that have small postings list over terms that are deemed very important by the term accuracy measures, thus sacrificing retrieval effectiveness for efficiency. On the flip-side, using $\alpha = 0$ results in selection terms based only on their accuracy scores. In our approach we select a fixed number of terms in this fashion, but one can easily imagine also varying the number of terms so as to fit into a budget of total sum of postings list lengths. We chose this approach, since many search engines impose limits on the number of terms they permit (e.g., as of writing Google imposes a limit of 32 terms).

At this point we must note that the specific term selection formula might be different for a different search engine implementation, but one can apply the same methodology that we present in this paper, and instead attempt to optimize a different function.

3.4 Weighting the Query Terms

In the previous section we saw that a **WAND** query acts as a linear discriminant boundary that ranks documents based on how much towards the positive side of the boundary they lie. Hence, the weights w_i in **WAND** queries can be computed by learning a linear classifier over the provided training instances. In this section we describe the various learning methods we use and the reasons we select them.

3.4.1 Support Vector Machines

Support Vector Machines (SVMs) were introduced by Vapnik [22] as a learning approach for two class problems. These techniques try to find a decision surface that maximizes the "margin" between the data points in different classes. Intuitively, in the linearly separable case, the SVM problem is to compute a vector $\vec{w}$ (and a threshold b) such that

$$\begin{aligned} \vec{w} \cdot \vec{d_i} - b &\geq +1 & \text{for } \ell_i &= \oplus \\ \vec{w} \cdot \vec{d_i} - b &\leq -1 & \text{for } \ell_i &= \ominus, \end{aligned}$$

while minimizing a norm of $\vec{w}$. We will use the coefficients in this vector $\vec{w}$ to weigh the terms in our **WAND** query. With the use of slack variables, a similar program can be used to solve the non-separable case too. Furthermore, kernels can be used to learn nonlinear decision boundaries in the data space using SVMs.

SVMs were applied to text categorization by Joachims [13] and were found to outperform all other classification methods. Moreover, though primarily a binary classifier, SVMs have also been employed for multi-class classification. Rifkin [19] showed that the one-vs-all strategy performed reasonably well. For these reason we use linear SVMs in an one-vs-all ensemble to indicate the "best" performance that is achievable at the classification task. While SVMs achieve very high classification accuracy, they suffer from the drawback that they are extremely computationally expensive. Hence, we consider other "formula-based" classifiers (mentioned below) that are a function of some small number of data statistics (such as frequencies of terms, etc.).

3.4.2 Naive Bayes Classifier

Naive Bayes (NB) classifiers make the assumption that the occurrences of words in a document are conditionally independent given the class labels [15, 16]. However, despite this seemingly "naive" assumption, NB has been found to be competitive for text classification problems, and even optimal under certain conditions [6, 9]. The multinomial NB classifiers model the distribution of words in a document as a multinomial. Using the Bayes rule, the decision boundary in a two class problem can be characterized as the following equation: $\log \mathbf{Pr}(\oplus) + \sum_i x_i \log \mathbf{Pr}(t_i \mid \oplus) = \log \mathbf{Pr}(\ominus) + \sum_i x_i \log \mathbf{Pr}(t_i \mid \ominus)$, where x_i is the number of occurrences of term t_i in the document being classified, $\mathbf{Pr}(\odot)$ is the prior probability of a document belonging to class $\odot$, and $\mathbf{Pr}(t_i \mid \odot)$ is the probability of term t_i appearing in a document of class $\odot$. (All these probabilities are estimated from the frequencies of the terms in the training set.) As one can see, the prior probabilities of the classes do not affect the weights of the terms and only define the threshold of the linear classifier. Therefore we set the weight for term t_i in the **WAND** query as

$$w_i = \log \mathbf{Pr}(t_i \mid \oplus) - \log \mathbf{Pr}(t_i \mid \ominus).$$

As we can see, learning the weights only involves a single pass over the data to estimate the statistics—even this can be avoided in some cases if the search engine stores precomputed statistics—and can be done very fast.

3.4.3 Rocchio's Formula

Relevance feedback is one of the most successful approaches for query reformulation. One of the first, and now a standard, approach proposed to create queries is *Rocchio's algorithm* [21], which is based on the vector information retrieval model (documents and queries are represented as vectors of weights of individual terms). The weights of the terms of the new query are given by

$$\vec{q}_m = \alpha \vec{q} + \frac{\beta}{|C_\oplus|} \sum_{\vec{d_j} \in C_\oplus} \vec{d_j} - \frac{\gamma}{|C_\ominus|} \sum_{\vec{d_j} \in C_\ominus} \vec{d_j},$$

where $\vec{q}$ and $\vec{q}_m$ are the vectors corresponding to the old and the modified query, respectively, $\vec{d_j}$ is the vector corresponding to document j, $C_\oplus$ ($C_\ominus$) is the set of documents among the retrieved documents belonging to the positive (negative) class, and α, β, and γ are constants (e.g., $\alpha = \beta = \gamma = 1$). Often this formula is iterated several times (setting $\vec{q} = \vec{q}_m$) to produce the final query.

In our work we use the Rocchio's formula as a classifier to learn the weights for the **WAND** query. The query so generated is the difference vector of the centroid vectors of the positive class and the negative class. This query is optimal in the sense that it maximizes the difference between the average score of documents in the positive and negative classes. One should note that since we are not implementing relevance feedback, we do not have an initial query (so $\alpha = 0$ and $C_\odot$ is the set of documents belonging to class $\odot$ in the training set) and we only iterate once. We also set $\beta = \gamma = 1$.

3.4.4 RTFIDF Formula

A weighting mechanism similar to Rocchio's that we apply is given by considering the document frequencies (number of documents that contain a given term). Specifically, we consider the *rtf* × *idf* formula for weighting terms:

$$w_i = \frac{1}{|C_\oplus|} \sum_{\vec{d_j} \in C_\oplus} tf(\vec{d_j}, t_i) \cdot \log \frac{N}{n_i} - \frac{1}{|C_\ominus|} \sum_{\vec{d_j} \in C_\ominus} tf(\vec{d_j}, t_i) \cdot \log \frac{N}{n_i},$$

where w_i is the weight given to term t_i, $tf(\vec{d_j}, t_i)$ is the frequency of term t_i in document $\vec{d_j}$, N is the total number of documents, and n_i is the number of documents containing term t_i. Haines and Croft [11] give an interpretation of this approach based on an inference network probabilistic model.

There are various other term weighting approaches that we can use, for example OKAPI-BM25 [12]. However, a detailed analysis of the best term weighting scheme is beyond the scope of this paper. Our goal is to show that efficient and effective classification can be performed on a search engine model by careful construction of queries. For this purpose we experiment with the four term weighting schemes mentioned above.

4. EXPERIMENTS

In this section we present an experimental study that evaluates our approach for classification with a search-engine model. Furthermore, we present an extensive exploration of the issues presented in Section 3.

4.1 Datasets and Experimental Setup

We used two datasets for our evaluation:

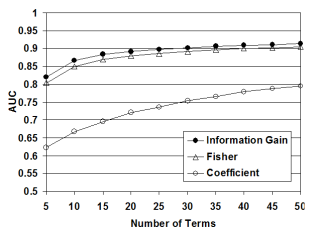
- **20 Newsgroups Dataset:** The 20 Newsgroups dataset has been extensively used for evaluation of text categorization techniques (see, for example, [18, 19]). It contains a total of 18,828 documents that correspond to English-language posts to 20 different newsgroups, with a little less than a 1000 documents in each. This is a challenging dataset for classification as some of the categories (newsgroups) are very similar to each other with many documents cross-posted among them (e.g., alt.atheism and talk.religion.misc). This dataset is relatively small, so we were able to perform extensive experiments with various parameterizations.

- **RCV1-v2 (Reuters) Dataset:** The larger dataset that we used in our experiments is the Reuters Corpus Volume 1 (RCV1-v2) data set [14]. It contains 804,414 newswire English-language stories written in 1996-1997. The documents have been categorized (by a combination of automatic and manual techniques) into 103 categories, such as "Domestic Markets," "Advertising," and so on. The dataset is split into a training set of 23,149 documents, and a test set of 781,265 documents—this is called the LYRL2004 split [14].

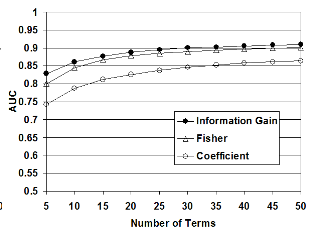
The query-creation part of our approach, including the feature selection and the classifier training, was implemented and executed in a Matlab environment. Furthermore, we indexed both datasets using the Juru Search Engine developed by IBM [3]. Juru was used to execute the queries and the classification accuracy of the results returned are reported in the following sections of the paper. All reported results were averaged over 5 random runs of the query building and execution. This ensures that the results are robust against biases in the training set selection. Finally, for all experiments other than those in Section 4.7, we do not use posting-list sizes for selection terms. This is done by setting $\alpha = 0$ in Equation (2).

4.1.1 *Evaluation Measures*

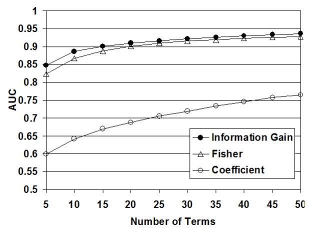
The query execution step of our approach returns a ranked list of documents that match the query. One of the standard



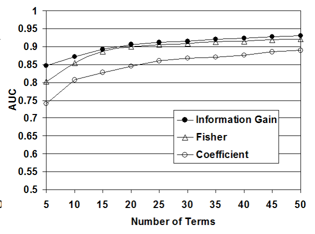
(a) Naive Bayes (50 docs)



(b) SVM (50 docs)



(c) Naive Bayes (200 docs)



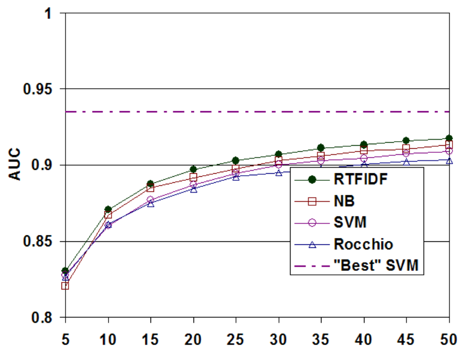
(d) SVM (200 docs)

Figure 1: Plots depict AUC of results of queries with terms chosen by different selection measures vs. number of terms in the query, in the 20 Newsgroups dataset. Plots (a) and (b) correspond to training set sizes of 50 docs per class, while (c) and (d) correspond to training with larger datasets (200 docs per class).

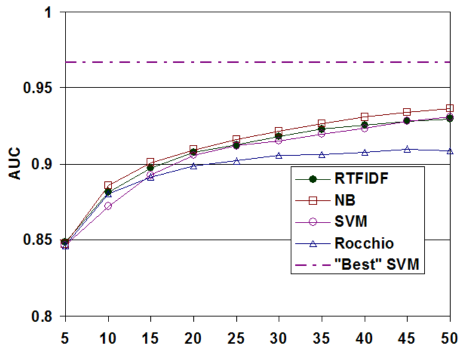
ways in information retrieval to plot the performance of such a classifier is through *precision-recall* graphs. *Precision* is the fraction of retrieved documents that are in the positive set, while *recall* is the fraction of documents of the positive set that were retrieved. In this paper we report the harmonic mean of these quantities, which is also known as the *F–measure*.

Receiver Operating Characteristics (ROC) graphs convey similar information as the precision-recall graphs. ROC graphs are two-dimensional graphs in which the *true positive rate* (the fraction of the documents belonging to the positive class that were correctly classified) is plotted on the Y-axis and the *false positive rate* (the fraction of the documents belonging to the negative class that were classified as positive) is plotted on the X-axis. One of the characteristics of the ROC graphs that makes them attractive for collections of documents that are being continually modified (such as the set of documents in the World Wide Web) is the fact that the graphs are insensitive to the ratio of the sizes of the positive and negative classes. A detailed description of the ROC graphs can be found in [7].

The Juru search engine returns the documents with score higher than the threshold specified by the user. Increasing the threshold results in an increase in precision and a reduction in recall. Selecting the appropriate threshold is an interesting problem and it is application specific, depending on the desired precision and recall. In our experiments, since we wanted to reproduce the entire precision-recall and ROC graphs, and not a particular point, we set the threshold to 0 (returning, as a result all the documents containing at least one positive term) and then observed the scores of the documents returned (Juru provides this information in its output) to create the desired graphs.



(a) IG (50 docs)



(b) IG (200 docs)

Figure 2: AUC of queries with terms weighed by different classifiers vs. number of terms in the query for the 20 Newsgroups dataset. The x-axis on both plots is the number of terms in the query. Plot (a) corresponds to training set sizes of 50 docs per class, while plot (b) corresponds to training with larger datasets (200 docs per class).

Since we want to compare different classifications, and often across different parameterizations, we need a quantitative measure of a given ROC graph. One that is often employed is the *area under the curve* (AUC) score of the ROC graph. A classifier that randomly classifies documents has an expected AUC score of 0.5, while a perfect classifier has an AUC score equal to 1.

4.2 Comparing Term Accuracy Scores

In these experiments, we compared the term-accuracy-scores on their ability to create queries with good classification accuracy. In all resultant plots, the AUC of the retrieved results is plotted against the number of terms in the query for each term-accuracy measure. For robustness, we present results for queries with terms weighted by two different classifiers. Also, we performed experiments with small and large training datasets. The plots are shown in Figure 1.

As we can see, for both the term weighting schemes Information Gain (IG)–based term selection is slightly better than Fisher-Index (FI)–based term selection. Both these methods in turn are much better than the Coefficient-based term selection. These two observations hold for all query sizes, however the difference between the performances of the measures becomes smaller as queries get larger. These observations also hold for the other term weighting schemes (RTFIDF and Rocchio) as well. A point to note is that the relative accuracies of the three term selection measures are not impacted much by changes in the training set size. In other words, even when there is plenty of data to weight the terms in the query appropriately, careful selection of terms is extremely important. Another key observation is that while selecting terms based on coefficients in the weight vector, the SVM based weighting outperforms the Naive Bayes based one. Interestingly, this difference disappears with other term selection mechanisms. This is further evidence to show that careful term selection is critical for creating effective queries.

4.3 Comparing Term Weighting Schemes

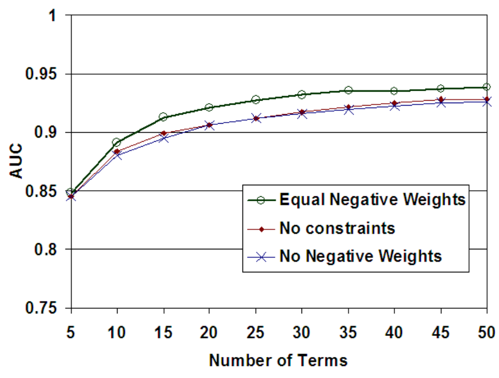
In this section, term weighting schemes are compared in terms of the accuracy of the queries they generate. For robustness, we used two term selection mechanisms with two different training data sizes. The results for the IG term selection measure are shown in Figure 2 (results for the Fisher measure are similar but not shown because of paucity of space). All resultant plots have AUC of results plotted against the number of terms in the query. The “Best” SVM curve plots the performance of a linear SVM classifier with the full set of terms ($\approx 14,000$). This curve represents the best performance achievable if there were no constraints on the number of terms in the query.

The performance of all four weighting schemes is comparable with no clear winner. These results hold for all term-selection mechanisms. As expected, the accuracies of queries increase as the number of terms in the queries increase, almost approaching the accuracy of the “best” possible SVM classifier. In fact, the accuracy of the query with as few as 5-10 terms is 90% of the “best” SVM accuracy. Hence, with careful selection and weighting of terms, we can achieve, using very few terms, classification accuracies comparable with the use of the full feature set.

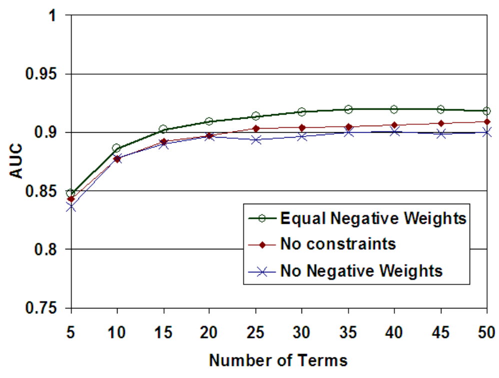
4.4 Impact of Negative Terms on Accuracy

For the purposes of this paper we extended the **WAND** primitive to support negative weights on terms in the query. In this section we evaluate whether this additional expressive power results in more accurate queries. Here, by negative (positive) terms we mean terms that have negative (positive) weights in the **WAND** query.

First we want to note that both the IG and FI based term selection measures tend to select very few negative terms. More concretely, for the 20 Newsgroups dataset, on average only 2 out of the top-50 features as ranked by the IG term-accuracy-score are negatively correlated to the class label. The corresponding value for the FI measure is 7 out of the top-50. For the top-100 features, these values are 7 and 18 for the IG and FI based measures respectively. The reason behind this is the one-vs-all approach that we employ for selecting terms and learning their weights. In this approach, while the positive class is a cohesive set of documents, the negative class comprises documents on all other topics. Hence, potential negative terms are only present in a small fraction of documents of the negative class, thereby scoring low with the IG based measure (which measures correlation to the class label). Similarly, as the negative class is very diverse, these terms typically have large intra-class variance and hence score low with the FI based measure too



(a) IG + RTFIDF (200 docs)



(b) IG + Rocchio (200 docs)

Figure 3: "No constraints" corresponds to accuracy of queries with terms selected by the IG measure. From the terms scored by IG, we selected equal number of positive and negative weighted terms (plotted as "Equal Negative Weights"). We also plotted accuracy of queries with "No Negative Weights".

(FI score is inversely proportional to intra-class variance of the term).

However, including negative terms in queries is useful for distinguishing between documents of closely related classes. Consider for example, learning a query for the class *talk.religion* in the 20 Newsgroup dataset, which also contains the class *alt.atheism*. While these two classes share a significant common vocabulary, the term "Atheist" should serve as a useful negative term. However, IG and FI scores for this term would be low because it would not be found in documents of most topics (other than *alt.atheism*) in the negative class.

In order to empirically evaluate the impact of negative terms in queries, we measured the change in classification accuracy after artificially replacing a few positive terms with negative terms (that scored lower on the term-accuracy-score). We depict some results in Figure 3. The bottom line shows the accuracy achieved if we select only positively weighted terms, as we would have done if the search engine did not have the added functionality of **WAND** to handle negative terms. As expected, in this case the classification accuracy drops only marginally (recall that there were very few negative terms to begin with). In the top line we see

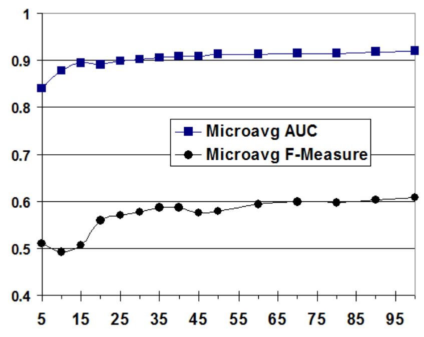


Figure 4: AUC and F-measure of the query results vs the number of query terms.

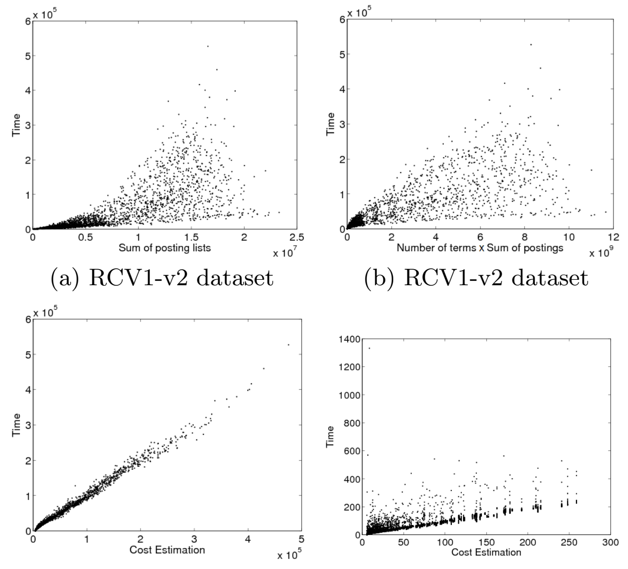
the accuracy when we ensure that the number of positive and negative terms in the query is the same. Notice that in this case the accuracy is increased over the case when we do not impose any constraints (middle line). Interestingly, this happens even when the positive terms being replaced are ranked higher by IG than the negative terms replacing them. The choice of 0.5 as the ratio of the number of positive terms over the total number of terms here is arbitrary; in general we expect the optimal ratio of positive to all the terms to be corpus and query dependent (between 0.5 and 1). Estimating this ratio before executing a query (e.g., as a function of the corpus) is an interesting problem for future research.

In summary, we see that the ability to perform search queries with negative weights improves the effectiveness of our classification. Furthermore, there is evidence that while learning queries with very few terms in a corpus with a number of diverse classes, selecting terms solely based on IG and FI gives inferior results.

4.5 Accuracy on the RCV1-v2 Dataset

In our experiments with the RCV1-v2 dataset we created queries using the IG term-accuracy-score and the Naive Bayes term weighting scheme. Figure 4 shows the increase in micro-averaged values of AUC and F-measure of the query results as the number of query terms is increased. To compute the F-measure, the threshold for the classifier was set using a held-out validation set of 5% of the test set documents in each class. As can be seen from the plot, the F-measure increases from 0.5 to 0.6 over 100 terms, and the AUC increases from 0.85 to 0.92.

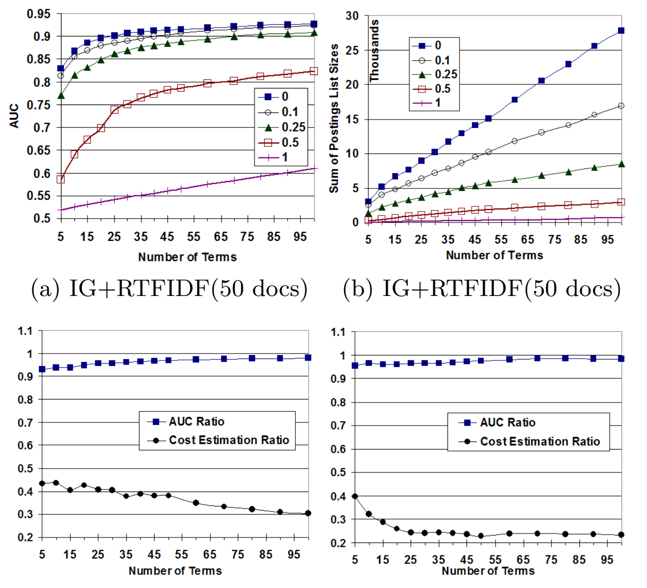
In order to put these numbers in perspective we reproduce some results from [14], which though not strictly comparable (because of tuning of SVM and Rocchio parameters) are nevertheless instructive. Lewis et al. performed classification experiments using SVM and Rocchio classifiers on the RCV1-v2 dataset. As in our work, the classifiers were learned using 23,149 documents and the remaining 781,265 documents were used as the test set. However, the number of features used for learning were $\approx 48,000$. With these settings the F-measure obtained using SVM and Rocchio was 0.81 and 0.69 respectively. We can see that using queries with very few terms ($\leq 0.1\%$ of 48K), our approach achieves a significant fraction of the accuracy. Furthermore, our approach provides the benefit of classification of the whole corpus in "output-sensitive" time (seconds).



(a) RCV1-v2 dataset (b) RCV1-v2 dataset

(c) RCV1-v2 dataset (d) 20 Newsgroups dataset

Figure 5: "Query execution time" (in ms) is plotted against various parameters for the Reuters and 20 Newsgroups dataset. All these queries were created using the Naive Bayes weighting scheme and the IG term selection measure. Section 4.6 contains details about the final cost-estimation formula we obtain.



(a) IG+RTFIDF(50 docs) (b) IG+RTFIDF(50 docs)

(c) IG+RTFIDF(50 docs) (d) Fisher+RTFIDF(50 docs)

Figure 6: Graphs (a) and (b) depict the AUC and sum of postings lists as a function of the number of query terms for different values of α (defined in Section 3.3.2). Graphs (c) and (d) depict the Ratio of AUC and the Cost Estimate for queries created by $\alpha = 0.25$ and $\alpha = 0$, for different query sizes.

4.6 Query Execution Time

In this section, we report on our experiments seeking to determine the function of parameters on which query execution time depends. In Figures 5(a), 5(b), and 5(c) we have plotted some of our attempts, and in Figure 5(c) we see a strong indication that the query execution time is proportional to

$$\left(a \cdot |T^+| \cdot \sum_{t \in T^+} post(t)\right) + \left(b \cdot |T^-| \cdot \sum_{t \in T^-} post(t)\right) + c$$

where T^+ (T^-) is the set of terms in the query with positive (negative) weights, $post(t)$ is the length of the posting list of term t, and a, b, and c are constants. We computed the values of these constants through regression analysis, and for the RCV1-v2 dataset their values are $a = 8.68 \cdot 10^{-5}$, $b = 3.57 \cdot 10^{-6}$, and $c = 4.15 \cdot 10^3$.

This dependency is a result of our implementation of the **WAND** operator, and a different implementation will presumably demonstrate a different dependency. Despite this, the entire procedure that we have followed can be applied to other implementations to deduce results of the same type, that can later guide the term-selection mechanism.

The corresponding graph for the 20 Newsgroups dataset is depicted in Figure 5(d). We can still see the same trend, the linear dependency of the time on the cost estimation formula presented above. However, there is more noise in this case since 20 Newsgroups is a much smaller dataset, and therefore the running time of queries is small (often a few milliseconds) making our measurements more susceptible to other factors. The constants in the cost estimation formula for the 20 Newsgroups dataset are $a = 9.84 \cdot 10^{-5}$, $b = 3.2 \cdot 10^{-5}$, and $c = 5.6$.

4.7 Using Posting-List Sizes for Term Selection

In this section we experiment with the term-selection-score in Equation (2). Specifically, we vary the exponent α to observe the effect of normalizing the term-accuracy-score with the size of the term's postings list. All these experiments were performed on the 20 Newsgroups dataset using RTFIDF based queries, trained with 50 documents per class.

Figure 6(a) graphs the accuracy (in terms of AUC of query results) versus the number of terms in the query. The different curves represent varying amount of importance given to the size of the postings list in term selection (by varying α). As expected, as α increases and more emphasis is laid on reducing the cost of the query at the expense of discriminative power of the terms, the accuracy decreases. Figure 6(b) shows how the sum of postings list of the terms in the query increases with the number of terms in the query. Once again, the different curves represent different values of α. As expected, for higher α the rate of increase of the sum of postings list is lower. From these two plots (Figures 6(a) and 6(b)), we can conclude that $\alpha = 0.25$ seems to offer the best trade-off of accuracy of query results and efficiency of query execution.

In Figures 6(c) and 6(d) we plot the ratio of AUC and "query execution time" (cost estimate based on our investigation in Section 4.6) for the queries generated at $\alpha = 0.25$ over queries generated with the highest possible values (queries created with $\alpha = 0$). As we can see, for the IG selection measure, for as low as 10-15 query terms the loss in accuracy over the best possible is $< 7\%$ with a "time" savings $> 50\%$. The tradeoff is even better for the FI selection measure ($< 3\%$ and $\approx 70\%$ respectively). For both selection measures, the trade-off between accuracy and efficiency gets better as the number of terms in the query increases. Hence, we have shown that by selecting terms after normalizing the term-accuracy-scores with the size of their postings list, we can obtain very efficient queries with minimal loss in classification efficacy.

5. CONCLUSIONS

We showed how classification can be performed effectively and efficiently using a search-engine model. This model has several benefits; such as "output-sensitive" classification time and the ability to function on a corpus processed primarily for search. We detailed a framework for construction of short queries via selection and weighting of terms. We showed that surprisingly good classification accuracy can be achieved on average by queries with as few as 10 terms. Further, we showed that this accuracy could be boosted by judicious addition of negative terms in the query.

We studied the trade-offs between accuracy (effectiveness) and query-processing time (efficiency). As a part of this study, we performed experiments to determine the relationship of the query execution time with number of terms and their postings list sizes in our search engine. Furthermore, we showed that by carefully selecting terms we can can further improve efficiency with minimal loss in accuracy.

6. ACKNOWLEDGEMENTS

For our large scale experiments with the Juru search engine, we would like to thank David Carmel, Steve Gates, Jasmine Novak, and Wilfried Teiken for their help and support. Ravi Kumar and Andrew Tomkins participated in the early discussions that led to the problem formulation and to the paper approach. Finally, we also want to thank Rie Ando, Roberto Bayardo, David Gondek, David Johnson, and Tong Zhang for many useful discussions and suggestions on the content of the paper.

7. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.

[2] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. of the* 12th *International Conference on Information and Knowledge Management*, pages 426–434, 2003.

[3] D. Carmel, E. Amitay, M. Herscovici, Y. S. Maarek, Y. Petruschka, and A. Soffer. Juru at TREC 10 - Experiments with Index Pruning. In *Proc. of the* 10th *Text REtrieval Conference (TREC).* NIST, 2001.

[4] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal*, 7(3):163–178, 1998.

[5] C.-H. Chang and C.-C. Hsu. Enabling concept-based relevance feedback for information retrieval on the WWW. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):595–609, 1999.

[6] P. Domingos and M. J. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[7] T. Fawcett. ROC graphs: Notes and practical considerations for data mining researchers. Technical Report HPL-2003-4, HP Laboratories, Jan. 17 2003.

[8] G. W. Flake, E. J. Glover, S. Lawrence, and C. L. Giles. Extracting query modifications from nonlinear SVMs. In *Proc. of the* 11th *International Conference on World Wide Web*, pages 317–324, 2002.

[9] J. H. Friedman. On bias, variance, 01 loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[10] E. J. Glover, G. W. Flake, S. Lawrence, W. P. Birmingham, A. Kruger, C. L. Giles, and D. Pennock. Improving category specific web search by learning query modifications. In *Symposium on Applications and the Internet, SAINT*, pages 23–31, 2001.

[11] D. Haines and W. B. Croft. Relevance feedback and inference networks. In *Proc. of the* 16th *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2–11, 1993.

[12] M. Hancock-Beaulieu, M. Gatford, X. Huang, S. E. Robertson, S. Walker, and P. W. Williams. Okapi at trec. In 5th *Text REtrieval Conference (TREC)*, 1997.

[13] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. of* 10th *European Conference on Machine Learning*, pages 137–142, 1998.

[14] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[15] A. McCallum and K. Nigam. A comparison of event models for Naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[16] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[17] J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., 1993.

[18] J. Rennie and R. Rifkin. Improving multiclass text classification with the support vector machine. In *Massachusetts Institute of Technology. AI Memo AIM-2001-026*, 2001.

[19] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[20] S. E. Robertson and K. S. Jones. Relevance weighting of search terms. *Journal of the American Society of Information Science*, 27:129–146, 1976.

[21] J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. Prentice-Hall, 1971.

[22] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.

[23] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proc. of the* 22nd *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, August 1999.

[24] Y. Yang and J. Pedersen. A comparative study on feature selection in text categorization. In *Proc. of the* 14th *International Conference on Machine Learning*, pages 412–420, 1997.