Boosting and Rocchio Applied to Text Filtering

Robert E. Schapire, Yoram Singer, Amit Singhal

AT&T Labs — Research
180 Park Avenue, Florham Park, NJ 07932
{schapire,singer,singhal}@research.att.com

Abstract

We discuss two learning algorithms for text filtering: modified Rocchio and a boosting algorithm called AdaBoost. We show how both algorithms can be adapted to maximize any general utility matrix that associates cost (or gain) for each pair of machine prediction and correct label. We first show that AdaBoost significantly outperforms another highly effective text filtering algorithm. We then compare AdaBoost and Rocchio over three large text filtering tasks. Overall both algorithms are comparable and are quite effective. AdaBoost produces better classifiers than Rocchio when the training collection contains a very large number of relevant documents. However, on these tasks, Rocchio runs much faster than AdaBoost.

1 Introduction

With the explosion in the amount of information available electronically, information filtering systems that automatically send articles of potential interest to a user are becoming increasingly important. If users indicate their interests to a filtering system with some examples of their liking and disliking, a system can automatically learn a *user profile* or a *relevance classifier* for a user. As and when a new article exhibits a substantial match to a user's profile, it is filtered and sent to the user. Thus text filtering is just binary text classification into the categories "relevant" and "not relevant."

The problem of text filtering has been studied in two different communities — machine learning (ML) and information retrieval (IR). Many algorithms for text filtering have been proposed and evaluated in the past, for example, Bayesian classifiers, k nearest neighbors, neural networks, rule-learning algorithms, and many more [17, 20, 40, 2, 41, 14, 22, 8, 24]. Most studies use Rocchio's method [28], a well known algorithm in the IR community (traditionally used for relevance feedback and more recently for document routing [38]), as a comparison baseline for their classifiers. However, most such studies use a weak version of Rocchio's algorithm, not well-suited for text filtering. In recent years, the IR community has proposed several modifications to Rocchio's algorithm that have vastly improved the performance of this algorithm: better term weighting [26, 35], query-zoning [36], and dynamic feedback optimization [6] being the three most notable improvements. In this study, we adapt a state of the art Rocchio's algorithm for the text filtering task, and compare it to a fairly new ML algorithm called "boosting."

We first develop a text filtering algorithm based on Freund and Schapire's AdaBoost algorithm [9], which is currently the most successful of a family of boosting algorithms. The main idea of boosting is to generate many, relatively weak classification rules and to combine these into a single highly accurate classification rule. Boosting algorithms have attractive theoretical properties, and have also been shown to perform well experimentally on more standard machine learning tasks [10, 25, 4]. We compare AdaBoost to Sleeping-Experts, an algorithm proposed by Blum [3], studied further by Freund et al. [11], and first applied to text filtering by Cohen and Singer [8]. This algorithm has been shown to be more effective than many current text filtering algorithms [8]. We show that, for text filtering, AdaBoost is definitively superior to Sleeping-Experts. We then compare AdaBoost to Rocchio's method and show that the two algorithms are quite competitive. Even though both algorithms learn a linear classifier, AdaBoost is superior to Rocchio when there is a large amount of training data to learn from. However, it is much faster to train a Rocchio classifier.

Previous studies in text filtering have used many different datasets and many different evaluation measures. This renders a relative comparison of any two approaches almost impossible. As described in the next section, most evaluation measures used in the past for evaluating filtering effectiveness are unfit for the purpose. Recently the TREC conferences have been moving toward the use of *utility* as the measure of choice for evaluating text filtering [18, 19, 13]. This study also presents results using utility that can be used by other researchers for comparison purposes in the future.

To summarize, in this study we aim to:

- Develop a new algorithm for text filtering based on boosting, and show that our algorithm is better than Sleeping-Experts, another highly effective algorithm for text filtering.
- Adapt a recent version of Rocchio's algorithm for text filtering, and study the relative merits of boosting-based and Rocchio-based classifiers.
- Present results based on new and better evaluation measures that can be used by other researchers in the future for comparison.

The rest of this study is organized as follows. Section 2 discusses the evaluation measures used for text filtering. Section 3 describes an adaptation of AdaBoost for text filtering. Section 4 presents a modified version of Rocchio's algorithm for text filtering. Section 5 describes the datasets used in this study. Section 6 describes our experiments and discusses the results. Finally, Section 7 concludes the study.

2 Evaluation measures

Past studies on text filtering have used a variety of measures for evaluating performance. One measure that is frequently used in doing cross-system comparisons is the recall-precision break-even point. Proposed by Lewis in [17], this measure has been the measure of choice in many studies on text filtering [17, 21, 8, 24, 39, 23, 15]. Roughly speaking, break-even point is the point at which recall of a filtering system is the same as its precision. So if the break-even point of a system is said to be 0.45, then at recall 0.45, the precision of the system is also 0.45. The aim of a filtering system is to obtain as high a break-even point as possible.

This measure, though popular, has several problems for evaluating a filtering system [16]:

- Often, we need to interpolate the scores to obtain the break-even point. Interpolation gives values not achievable by the system.
- The point where recall equals precision is neither a desirable nor an informative target from a user's perspective.

We strongly believe that break-even point should not be used for evaluating text filtering effectiveness, and do not use it in this study.

Some other measures that have been used to evaluate text filtering are:

- Average precision, or precision at a fixed rank cutoff: Many studies have used one of these measures to evaluate filtering effectiveness [2, 40, 41, 22, 1, 7]. These measures are intended to evaluate the ranking effectiveness of a system [31], not its filtering effectiveness. Even though the filtering effectiveness of a system is related to its ranking effectiveness, this relationship is not strong enough to use ranking evaluation measures to evaluate text filtering.
- Van Rijsbergen's F-measures: Used in [20, 22, 8, 39] to evaluate filtering, this is a single valued measure that depends upon the relative importance a user assigns to recall and precision (see [37], pp. 168–176). The main drawbacks of this measure are that its value is not directly interpretable by a user, and it is usually hard for a user to judge the relative importance of recall and precision. For example, most users would find it hard to say whether recall is twice as important as precision for them, or thrice, or some other ratio. However, in our view, the F-measures is the best suited measure (among the above measures) for evaluating filtering.

A big drawback of all the measures listed above is their dependence on recall. Recall is not available until all the test documents have been seen. These measures can't be used to evaluate a system on-the-fly, *i.e.* to compute any of these measure if only a portion of the documents have been classified and the rest still need to be classified. Thus, if for instance a user wants to check at the end of the day how a filtering system is doing, he or she would not be able to assess the performance of the system using the above measures.

In this study we use three utility-based evaluation measures that don't suffer from the drawbacks mentioned above. We also report performance results for non-interpolated average precision[1], and $F_{\beta=1}$ (which is $\frac{2 \cdot recall \cdot precision}{recall + precision}$) through our web page[2]. Though we believe that average precision (and to some degree $F_{\beta=1}$) is not well suited to evaluate filtering effectiveness, we still report these figures for comparison purposes with other research.

[1]Following the notation used in later sections, let $Rank(d)$ be the rank assigned by the classifier to document d and let the set of relevant document be denoted by Rel. Then, the non-interpolated average precision is,

$$\frac{1}{|Rel|} \sum_{d \in Rel} \frac{|\{d' | d' \in Rel,\ Rank(d') \leq Rank(d)\}|}{Rank(d)} .$$

[2]`www.research.att.com/~singhal/sigir98-rocboost.dat`

Utility

Recently, the TREC text filtering evaluations have been using utility measures, which assign rewards (or penalties) for each pair of machine prediction and correct label [19, 13, 14]. Let r_+ be the number of relevant documents that are classified relevant by the machine, and r_- the number of relevant documents misclassified as irrelevant. Similarly, n_+ and n_- are the number of non-relevant documents classified as relevant and irrelevant, respectively. With each pair of human judgement: relevant or non-relevant (*rel* or *nrel*, respectively), and machine prediction ($+$ or $-$) we associate a utility value. We denote the utility values by u_{rel+}, u_{nrel+}, u_{rel-} and u_{nrel-}. Therefore, the overall performance of a classifier in terms of the utility matrix is $r_+ u_{rel+} + r_- u_{rel-} + n_+ u_{nrel+} + n_- u_{nrel-}$. The aim of a filtering system then, is to maximize the utility.

The first evaluation measure we use—classification error—is simply the number of mistakes a classifier makes, *i.e.* the sum of the number of positive (relevant) documents that are classified as negative, and the number of non-relevant documents classified as positive. Note that minimizing the classification error is equivalent to maximizing a utility when $u_{rel+} = u_{nrel-} = 0$ and $u_{rel-} = u_{nrel+} = -1$. Therefore, a learning algorithm that maximizes utility can be used for minimizing the classification error.

One problem with the classification error is that for datasets with very few relevant documents, a classifier that uses the simple strategy of predicting that every document is non-relevant, is able to achieve very low error. To handle this common difficulty, we need to specifically reward a classifier for finding relevant documents. We therefore use two other utility measures, Util-1 and Util-2 (described below) which explicitly reward a system for finding relevant documents. In summary, the utility measures used in this study are:

	u_{rel+}	u_{rel-}	u_{nrel+}	u_{nrel-}
Error:	0	-1	-1	0
Util-1:	3	0	-2	0
Util-2:	3	-1	-1	0

3 Boosting for text filtering

In this section, we describe how we have adapted Freund and Schapire's AdaBoost boosting algorithm [10] for text filtering. The main idea of boosting is to combine many "rules-of-thumb." For example, in this study, we use rules-of-thumb which test on the presence of a term, such as the following simple rule: "If the word 'money' appears in the document then predict that the document is 'relevant'; otherwise predict 'not relevant.' " Clearly, a simple-minded rule of this kind will misclassify many documents. The main idea of boosting, however, is to generate and combine many such rules in a principled manner to produce a single highly accurate classification rule.

Formally, the rules-of-thumb are called *weak hypotheses*. Boosting assumes access to an algorithm or subroutine for generating these rules-of-thumb, called the *weak learner* or *weak learning algorithm*. The boosting algorithm calls the weak learner many times to generate many rules-of-thumb, and these are then combined into a single classification rule called the *final* or *combined hypothesis*.

One main feature of the boosting algorithm is that, during the course of its execution, it assigns different *importance weights* to different training documents. The weak learning algorithm takes these weights into consideration, and chooses each rule-of-thumb so as to correctly classify as many documents as possible, taking into account the greater importance of correctly classifying documents which have been assigned greater weight. As the algorithm progresses, training documents that are hard to classify correctly get incrementally higher weights while documents

```
Input:
  $N$ documents and labels: $\langle (d_1, y_1), \ldots, (d_N, y_N) \rangle$ where $y_i \in \{-1, +1\}$;
  integer $T$ specifying number of iterations
Initialize $D_1(i)$    (for classification error, $D_1(i) = 1/N$)
Do for $s = 1, 2, \ldots, T$:
  1. Call WeakLearn and get a weak hypothesis $h_s$
  2. Calculate the error of $h_s$: $\epsilon_s = \sum_{i: h_s(d_i) \neq y_i} D_s(i)$.
  3. Set $\alpha_s = \frac{1}{2} \ln \left( \frac{1 - \epsilon_s}{\epsilon_s} \right)$.
  4. Update distribution:
       $D_{s+1}(i) = \frac{D_s(i) \exp(-\alpha_s y_i h_s(d_i))}{Z_s} = \frac{D_s(i)}{Z_s} \times \begin{cases} e^{-\alpha_s} & \text{if } h_s(d_i) = y_i \\ e^{\alpha_s} & \text{if } h_s(d_i) \neq y_i \end{cases}$
     where $Z_s$ is a normalization factor.
Output the final hypothesis:
  $h_{fin}(d) = \text{sign} \left( \sum_{s=1}^{T} \alpha_s h_s(d) \right)$.
```

Figure 1: AdaBoost algorithm for binary text filtering

that are easy to classify get lower weights. This, in effect, forces the weak learning algorithm to concentrate on documents that have been misclassified most often by the previously derived rules-of-thumb.

The final combined hypothesis classifies a new document by computing the prediction of each of the weak hypotheses on this document and taking a (weighted) vote of these predictions.

A description of AdaBoost is shown in Figure 1. AdaBoost takes as input a training set of N document-judgment pairs $\langle (d_1, y_1), \ldots, (d_N, y_N) \rangle$ where d_i is a document in the training collection, and $y_i \in \{-1, +1\}$ is the label associated with the document where $+1$ or -1 means that the document is relevant or irrelevant (as judged by a human expert).

As just described, AdaBoost calls the weak learning algorithm WeakLearn repeatedly in a series of rounds. On round s, AdaBoost provides WeakLearn with a set of importance weights over the training set. In response, WeakLearn computes a weak hypothesis (rule-of-thumb) h_s which, given a document d, classifies it as $+1$ (relevant) or -1 (non-relevant). We later discuss the weak learner that was used in our experiments.

The importance weights are maintained formally as a distribution D over training documents. As this distribution changes after each round, we denote the distribution before round s by D_s. The weight of a training document d_i under distribution D_s is written $D_s(i)$, and we maintain the condition that $D_s(i)$ is always positive and $\sum_{i=1}^{N} D_s(i) = 1$. Initially, for classification error, we set all the weights equally so that $D_1(i) = 1/N$. (As described below, a different initialization procedure is used for other utility functions.)

The goal of the weak learner is to find a rule-of-thumb which misclassifies as few documents as possible, relative to the distribution D_s. Formally, the weak learner attempts to find a weak hypothesis h_s with low weighted error

$$\epsilon_s = \sum_{i: h_s(d_i) \neq y_i} D_s(i). \tag{1}$$

This error can be interpreted as the probability of misclassifying a document chosen randomly according to distribution D_s. It is the sum of the weights of all relevant documents classified as irrelevant by h_s, and all irrelevant documents classified as relevant.

Having obtained a hypothesis h_s from WeakLearn, AdaBoost next updates the weights of all the documents in such a way that documents classified correctly get a lower weight and the misclassified documents get a higher weight. To be more specific, AdaBoost multiplies the weight of each document correctly classified by h_s by $e^{-\alpha_s}$, and the weight of each incorrectly classified document by e^{α_s}. Here, α_s is a number whose computation is discussed below. Assuming for the moment that α_s is positive, this update of the weights has the effect of decreasing the weights of correctly classified documents (since $e^{-\alpha_s} < 1$) and increasing the weight of correctly classified documents (since $e^{\alpha_s} > 1$).

To ensure that the new weights D_{s+1} form a distribution (so that $\sum_{i=1}^{N} D_{s+1}(i) = 1$), we then renormalize the weights, resulting in the update rule shown in Figure 1:

$$D_{s+1}(i) = \frac{D_s(i)}{Z_s} \times \begin{cases} e^{-\alpha_s} & \text{if } h_s(d_i) = y_i \\ e^{\alpha_s} & \text{if } h_s(d_i) \neq y_i \end{cases}$$

where Z_s is a normalization factor. Note also that, because y_i and $h_s(d_i)$ are each -1 or $+1$, their product is equal to -1 if they disagree, and $+1$ otherwise. Thus, in general, we can rewrite the update rule more succinctly as in the figure.

This process of generating weak hypotheses and updating the weights is repeated for T rounds. How we decide on a value of T is discussed later in this section. After T rounds, we have T hypotheses $h_1, \ldots, h_T$, as well as the values $\alpha_1, \ldots, \alpha_T$. A new document d is then classified using the following final hypothesis:

$$h_{fin}(d) = \text{sign} \left(\sum_{s=1}^{T} \alpha_s h_s(d) \right),$$

where $\text{sign}(x)$ is $+1$ if $x > 0$ and -1 otherwise. In words, the predictions of all of the weak hypotheses are evaluated on the new document d, and their predictions are combined by voting. If more weak hypotheses predict the document is relevant $(+1)$ rather than irrelevant (-1) then the sum above will be positive and the combined prediction will be relevant $(+1)$; otherwise the prediction is irrelevant. However, we do not assign equal importance to the predictions of each of the weak hypotheses. Instead, we weight the votes of the different weak hypotheses using the same values α_s which were previously used to update the distribution D_s.

We now discuss the exact choice of α_s. Let ϵ_s be the weighted error of weak hypothesis h_s (as computed in Figure 1 and Eq. (1)). We compute α_s as

$$\alpha_s = \frac{1}{2} \ln \left(\frac{1 - \epsilon_s}{\epsilon_s} \right).$$

To understand what this choice entails, suppose that a highly accurate weak hypothesis h_s has been found. Then ϵ_s will be small and α_s will be large. This translates into more drastic updates to the distribution and a greater weight assigned to the predictions of h_s in the computation of the final hypothesis. On the other hand, if h_s is highly inaccurate (with error ϵ_s close to $1/2$), then α_s will be small, the updates to the distribution will be quite conservative, and the predictions of h_s in the final hypothesis will receive rather low weight. See Freund and Schapire [10] for more complete motivation for this choice of α_s.

For our task, we also allow α_s to be negative. This will be the case whenever a weak hypothesis h_s is found with error ϵ_s greater than $1/2$. This is discussed further below.

3.1 Finding weak hypotheses

In our algorithm, the weak learner generates the hypothesis h_s as follows. All words and pairs of adjacent words are potential terms. Our implementation is capable of using arbitrary long n-grams but we restrict ourselves to words and word bigrams for this study. For each term, the weak learner computes the error (relative to the distribution D_s) incurred by predicting that a document is relevant if and only if it contains that term. Formally, this error is

$$\epsilon(t) \;=\; \sum_{i:t\in d_i,\; d_i\notin \mathit{Rel}} D_s(i) \;+\; \sum_{i:t\notin d_i,\; d_i\in \mathit{Rel}} D_s(i).$$

(Here, $t \in d$ means term t occurs in document d, and $d \in \mathit{Rel}$ means document d is relevant.)

Ordinarily, one would select the term which has the lowest classification error. However, consistent with the main aim of a classifier, we instead select the term that maximally differentiates relevance and non-relevance. For example, if term t_a has the lowest error, say 0.25, and term t_b has the highest error, say 0.90, then t_b is a better discriminator of non-relevance than t_a is of relevance. This is because whenever t_b is present, we can say that the document is non-relevant with a much higher confidence than the confidence we have in the relevance of a document if it contains t_a. Therefore, we select t_b for use in hypothesis h_s.

Formally, then, we choose the term that minimizes either $\epsilon(t)$ or $1 - \epsilon(t)$. Let t_s be the selected term. We then form the hypothesis h_s for a document d using the rule:

$$h_s(d) = \begin{cases} +1 & \text{if } t_s \in d \\ -1 & \text{if } t_s \notin d. \end{cases}$$

As mentioned above, if we select a term with very high error (more than $1/2$) for use in hypothesis h_s, the boosting algorithm automatically assigns a negative weight to that hypothesis. This means that if a word is a better predictor of non-relevance, then its presence would automatically *reduce* the score of a document.

3.2 Boosting with general utility functions

It remains to describe how to modify our algorithm in order to maximize gain for a general utility matrix. Let

$$U = \begin{pmatrix} u_{rel+} & u_{nrel+} \\ u_{rel-} & u_{nrel-} \end{pmatrix}$$

be a utility matrix. Using the notation introduced in Section 2, the gain of the classifier is, $r_+u_{rel+} + r_-u_{rel-} + n_+u_{nrel+} + n_-u_{nrel-}$. We make the natural assumption that there is more to be gained from correct classifications than incorrect classifications, that is, $u_{rel+} > u_{rel-}$ and $u_{nrel-} > u_{nrel+}$. The minimum possible gain for a classifier is when it classifies every document incorrectly; that gain is $u_{rel-}(r_+ + r_-) + u_{nrel+}(n_+ + n_-)$. If we assign this initial gain to the classifier, then the aim of a classifier is to maximize its additional gain above this amount. Every time the classifier classifies a relevant document correctly, we increase its gain by $u_{rel+} - u_{rel-}$, and every time a non-relevant document is classified correctly, we increase its gain by $u_{nrel-} - u_{nrel+}$. Nothing is done on a misclassification since we have already accounted for all possible misclassifications.

Thus, we would get the same results if we use the following utility matrix:

$$U' = \begin{pmatrix} u_{rel+} - u_{rel-} & 0 \\ 0 & u_{nrel-} - u_{nrel+} \end{pmatrix}.$$

This utility matrix implies that each relevant document is "worth" $\zeta = (u_{rel+} - u_{rel-})/(u_{nrel-} - u_{nrel+})$ irrelevant documents. For instance, for Util-1 we get that each relevant document is worth $\frac{3-0}{0+2} = 1.5$ non-relevant documents, and for Util-2 we get a factor of $\frac{3+1}{0+1} = 4$. We therefore need to set the initial distribution of the examples before the first round of boosting so that it will reflect this ratio between positive and negative examples. It is fairly straightforward to show that maximizing the utility for a matrix U is equivalent to minimizing the error when the initial distribution is such that the weight of the relevant documents is ζ times the weight of the irrelevant documents.

Formally, then, to handle general utility functions, we need to modify AdaBoost only in the manner in which the weights D_1 are initialized. Specifically, we set

$$D_1(i) = \frac{1}{Z_0}\begin{cases} u_{rel+} - u_{rel-} & \text{if } y_i = +1 \text{ } (d_i \text{ is relevant}) \\ u_{nrel-} - u_{nrel+} & \text{if } y_i = -1 \text{ } (d_i \text{ is irrelevant}) \end{cases}$$

where Z_0 is a normalization factor which ensures that $\sum_i D_1(i) = 1$. Note that the rest of the algorithm is unaffected by the change of the initial distribution.

3.3 Choosing the number of rounds

Finally, we need to specify how we set the number of rounds T. Since there is no theoretical analysis of the number of rounds needed to boost a weak hypothesis, we set this value empirically. We found that the following procedure yields good results. We first run the boosting algorithm until the *training* error reaches its minimal value, which often is zero.[3] Let T_0 be the number of rounds needed to reach the minimal training error. We then continue boosting for an addition $T_0/10$ rounds. Thus, the total number of rounds is $(1.1)T_0$. This typically means that we run more rounds of the boosting algorithm if the problem is "hard," requiring many features to attain a small training error. Put another way, the size of the resulting classifier, as a function of the number of features it employs, depends on how "easy" the classification problem is: the more difficult the problem is the larger the classifier we build.

4 Rocchio for text filtering

Retrieving useful documents for a user-query has always been a challenging problem in the field of information retrieval. In its early days, researchers realized that it is very hard for an average user to formulate a "good query," and therefore, for successful retrieval, aids for good query formulation should be provided to users. Automatic query formulation using *relevance feedback*, once the user has marked some of the documents (possibly retrieved by the initial user-query) as relevant and some as non-relevant, has been one of the most successful methods in IR [30].

A feedback query creation algorithm developed by Rocchio in the mid-1960's has, over the years, proven to be one of the best relevance feedback algorithms [27, 28]. Rocchio's algorithm was developed in the framework of the vector space model [32]. When documents are to be ranked for a query, an *ideal query* should rank all the relevant documents above all non-relevant documents. However, such a query might just not exist, or even if it does exists for the training data, it might be over-fitting the training documents and might not generalize to new (test) documents. Therefore we lower our aims and instead develop a query that maximizes the difference between the average score of a relevant document and the average score for a non-relevant

[3] Although the training error of the combined hypothesis may be zero, it is possible and not uncommon for boosting to proceed and for further reductions in the test error to occur. See Schapire et al. [33] for further discussion.

document. Rocchio calls this an *optimal query* ([28], page 315). Rocchio shows that under this definition, an optimal query vector is the difference vector of the centroid vectors for the relevant and the non-relevant articles,

$$\vec{Q}_{opt} = \frac{1}{R} \sum_{d \in Rel} \vec{d} - \frac{1}{N-R} \sum_{d \notin Rel} \vec{d} \qquad (2)$$

where $\vec{d}$ denotes the weighted term vector for document d, $R = |Rel|$ is the number of relevant articles, and N is the total number of articles in the collection. All negative components of the resulting query are assigned a zero weight.

To maintain focus of the query, researchers have found that it is useful to include the original user-query in the feedback query creation process. Coefficients have been introduced in Rocchio's formulation which control the contribution of the original query, the relevant articles, and the non-relevant articles to the feedback query. These modifications yield the following query reformulation function [30]:

$$\vec{Q}_{new} = \alpha \vec{Q}_{orig} + \beta \frac{1}{R} \sum_{d \in Rel} \vec{d} - \gamma \frac{1}{N-R} \sum_{d \notin Rel} \vec{d} \qquad (3)$$

This formulation, which was developed for ranking documents after relevance feedback, mainly in interactive settings, has also been used successfully for text filtering. In an information filtering scenario, once several documents have been marked as relevant to a user's information need, a "user profile" is created using Rocchio's formulation (Eq. (3)). Any new article that has high similarity (we use vector inner-product as the similarity measure in all our experiments, see [29], page 318) to this user profile is considered potentially useful for the user and is sent to the user.

Several techniques are known to improve the effectiveness of Rocchio's method. The three new developments that have been quite effective in conjunction with Rocchio's algorithm are:

1. **Better Term Weights**: A much better understanding of term weights has been developed in the IR community in recent years [26, 35]. Better term weights in the training documents yield a better Rocchio query. A better Rocchio query along with better term weights for the test documents yields much improved scores (*i.e.* better ranking) for the test documents.

2. **Query Zoning**: Recently Singhal et al. [36] have proposed that only a selected set of non-relevant documents that have some relationship to a user's interest should be used in Rocchio's method. They proposed sampling of the non-relevant documents to form a *query zone*.

3. **Dynamic Feedback Optimization**: Buckley et al. [6] have shown that further optimizing the term weights proposed by Rocchio's formulation on the training collection improves the quality of a feedback query for the test data.

We view these techniques as tools that bring a Rocchio query closer to the *ideal query*. We use all these techniques in our version of Rocchio.

Since there is no initial user-query in a text filtering scenario, the first factor in Eq. (3), $\alpha \vec{Q}_{orig}$, is not used in our system. Also, when using query zones, Singhal et al. [36] have shown that $\beta = \gamma$ in Eq. (3) is a reasonable parameter setting. Therefore for text filtering, we are back to using the original Rocchio formulation of Eq. (2) instead of Eq. (3).

We use the centroid vector of the relevant documents in the training corpus as the initial query and use this vector to form the query zone. Here are the steps involved in modified Rocchio's method for text filtering:

l tf factor:	$1 + log(tf)$
L tf factor:	$\frac{1+log(tf)}{1+log(\textit{average tf in text})}$
t idf factor:	$log(\frac{N+1}{df})$
u normalization factor:	$\frac{1}{0.8+0.2\,\frac{\textit{number of unique words in text}}{\textit{average number of unique words per document}}}$
tf	is the term's frequency in text (query/document)
N	is the total number of documents in the training collection
df	is the number of documents that contain the term, and

the average number of words/document is 45 for Reuters-21578, 137 for AP-BODY, and 110 for TREC.
the average number of phrases/document is 27 for Reuters-21578, 40 for AP-BODY, and 37 for TREC.

ltu weighting: **l** factor $\times$ **t** factor $\times$ **u** factor
Lnu weighting: **L** factor $\times$ **u** factor
Ltu weighting: **L** factor $\times$ **t** factor $\times$ **u** factor

Table 1: Term weights

1. Initial query: Create the centroid vector for the relevant documents in the training data. Documents are *Ltu* weighted (see Table 1). Remove all words that appear in fewer than 5% of the relevant documents and all phrases that appear in fewer than 2% of the relevant documents. This keeps infrequent (possibly "random") terms from influencing the query. Select the highest weighted n_w words and n_p phrases, where n_w is the average number of words per document, and n_p is the average number of phrases per document (Table 1). This is the initial query.

2. Using the above initial query, and *Lnu* weighted *training documents*, form a training "query-zone" by selecting the most similar $MAX(N/100, R)$ non-relevant documents for the initial query (using the inner-product similarity). Here N is the total number of documents in the training collection and R is the number of relevant documents for this query (class) in the training collection.

3. Using the non-relevant documents in the query-zone and *all* the relevant documents in the training corpus, form a feedback query using Rocchio's formulation using the following constraints/parameters:
   - Document terms are *Ltu* weighted.
   - Only the "non-random" words and phrases, *i.e.* the words that appear in at least 5% of the relevant articles, and phrases that occur in at least 2% of the relevant articles are considered for use in the feedback query.
   - Top n_w words and n_p phrases (same as step 1), as weighted by the original Rocchio formula (Eq. 2) are retained in the feedback query with weights predicted by the above formula.

4. Term weights in this query of n_w words and n_p phrases are further optimized using three-pass dynamic feedback optimization (DFO) with pass ratios 1.00, 0.50, and 0.25 [6]. Since DFO optimizes average precision in the training collection, and a fixed number of top documents are ranked in this process [6], we rank the top $MAX(500, 5R)$ documents in this step.

5. The optimized feedback query is used to rank the *Lnu* weighted training documents (using inner-product similarity). By going down in this ranked list of training documents, find a similarity threshold that would maximize the evaluation measure (error rate or utility) on the training data. This will be the *threshold* for the classifier.

6. The test documents are *Lnu* weighted. If a test document has a similarity higher than the threshold to a feedback query (classifier), then it is classified as relevant, otherwise non-relevant.

The above algorithm is quite similar to the routing algorithm used in [34] except for the following differences:

- Since the user query is not being used in any way, the relevant centroid is used to form the query zone. In previous work on query zones, the initial user query was used for this purpose [36].
- The query zone size ($MAX(\frac{N}{100}, R)$) grows with the class size for classes that have more than $\frac{N}{100}$ training relevant documents.
- DFO is also modified to rank more documents for very large classes ($MAX(500, 5R)$).
- Word cooccurrence pairs are not used.

We deliberately didn't tune Rocchio's algorithm any further to cover some of its weaknesses. For example, post-hoc analysis shows that it is possible to improve Rocchio's method for large classes by increasing the number of terms (words and phrases) used in the classifier. But we didn't do any such tuning. Though not straightforward, it is possible to enhance DFO and tune a Rocchio classifier explicitly to maximize utility. We did not do any such tuning for this study but plan to investigate this in near future.

5 Test corpora

We use three different text corpora for testing our algorithms: Reuters-21578, AP-Body, and TREC-3. For the Reuters-21578 and the AP-Body collections, one should pay special attention to which particular documents are used in the collection, there are some possible variations. For the TREC-3 data, we would like to emphasize that the original topic text supplied by the users is *not* being used in our experiments.

5.1 Reuters-21578

The Reuters-21578 text categorization test collection has been made publicly available from the web page

`http://www.research.att.com/~lewis`

by David Lewis, who originally compiled the collection. Documents in this collection were collected from Reuters newswire in 1987. We use the modified Apte split (ModApte), which assigns 9,603 documents dated before April 8, 1987 to the training set, and 3,299 documents dated from April 8, 1987 to the test set. In our experiments, we use the *ninety* TOPIC categories that have at least one relevant (positive) training documents and at least one relevant test document.

5.2 AP-Body

This test collection is made up of documents from the AP newswire included in the TREC disks 1–3 [12]. 142,791 documents from the years 1988 and 1989 are used as the training collection, and 66,992 documents from the year 1990 are used as the test collection. Each document in this collection has a distinct title field (marked by the SGML tag <HEAD>), and a distinct body field (marked by the SGML tag <TEXT>). We only use the body of a document in our experiments. There are *twenty* classes in this collection. See [20] for a description of how the classes for this corpus were derived.

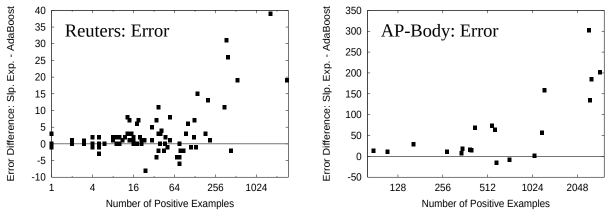

Figure 2: Comparison of AdaBoost and Sleeping-Experts.

This collection was first used by Lewis et al. in [22]. One should note that even though the original distribution of this data has 79,919 documents from the year 1988, and 84,678 from the year 1989, making a total of 164,597 training documents, 21,806 of the documents that have non-standard formats or structures have been omitted by Lewis et al. Similarly, out of the 78,321 possible test documents, 11,329 have been skipped. Details of what documents were skipped in the creation of this collection are available from David Lewis (`lewis@research.att.com`).

5.3 TREC-3

This collection, once again from the TREC disks 1–3, was used in the routing task in the third Text REtrieval Conference (TREC-3) [12]. The training collection contains all documents on TREC disks 1 and 2, whereas the test collection is made up of all the document contained in disk 3. There are a total of 741,856 training documents and 336,310 test documents. Fifty TREC topics, numbered 101–150 are used as individual classes in this collection [12].

Even though these TREC topics are long user-queries which contain many useful words for text filtering, we again emphasize that we *do not* use the topic texts in either of our systems. The relevance judgments for these topics are also available with the TREC data. A summary of the collections used in our experiments is shown in Table 2.

5.4 Preprocessing

In our experiments, we used the entire Reuters-21578 and AP-Body collections for training and testing. Unfortunately, it would have taken a very long time to run AdaBoost had we used the entire TREC collection. We therefore used a subset of the training collection as a training set. We selected the top $10,000$ documents retrieved from the training collection by a query learned using Rocchio's method (following the idea of query-zoning as in [36]). In addition, we added all relevant documents *not* retrieved by the above procedure to the training set. We also applied the same procedure to the collection of test documents. The classifier built by AdaBoost was tested on all the relevant documents in the test collection and only on the the top $5,000$ non-relevant test documents that were retrieved by Rocchio's query, the same query used for obtaining a sample of the training collection. Since Rocchio's method was used as the means of sub-sampling both the training and test collections, these sub-sampled collections include many of non-relevant documents that are relatively difficult to classify. We therefore believe that the results obtained on the sub-sampled datasets are a good estimate of the performance of the classifiers. But it is also true that such sampling of the test collection automatically rejects a large number of non-relevant documents from the pool of documents to be classified by the AdaBoost classifier. These documents can potentially be misclassified and can possibly yield results poorer than the results reported in this study for the AdaBoost algorithm for the TREC-3 task.

Dataset	Utility Checked	Number of Tasks	Min. Pos. (Train)	Max. Pos. (Train)	Train Set Size	Test Set Size
Reuters-21578	Err, Util-1, Util-2	90	1	2877	9,603	3,299
AP-Body	Err, Util-1, Util-2	20	88	2901	142,791	66,992
TREC-3	Err, Util-1, Util-2	50	31	803	741,856	336,310

Table 2: Summary of datasets used in the experiments.

6 Experiments and Results

This section discusses our experiments and results.

6.1 AdaBoost compared to Sleeping Experts

We first give experimental results which show that our adaptation of AdaBoost for text filtering achieves better results than, Sleeping-Experts, another effective algorithm for text filtering studied recently by Cohen and Singer in [8]. We compare the performance of AdaBoost and Sleeping-Experts on the AP-Body and the Reuters-21578 tasks. Figure 2 shows the results of this comparison. The scatter plot on the left hand side of Figure 2 show the error-difference on corresponding classes between Sleeping-Experts and AdaBoost as function of the number of relevant documents in the training collection for the Reuters collection. The right hand side plot is for the AP collection. A point above the x-axis indicates that Sleeping-Experts is inferior to AdaBoost as it makes more errors. These plots indicate that for classes of all size (where size of a class is the number of relevant training documents for that class), AdaBoost generally outperforms Sleeping-Experts, often with a large margin.

6.2 Reuters-21578

Figure 3 (top row) shows similar comparative plots for the performance of AdaBoost and Rocchio on Reuters-21578 dataset. The left hand side scatter plot give the difference in error, the middle plot shows the difference in Util-1, and the right plot shows the difference in Util-2 between AdaBoost and Rocchio as a function of the number of relevant training documents. As before, point above the x-axis indicates that AdaBoost achieves better results, whereas a point below the x-axis indicates that for that class, Rocchio outperforms AdaBoost.

For error and Util-1, the scatter plots are "skewed" towards the top-right corner indicating that AdaBoost is better than Rocchio for classes that have a large number of relevant documents in the training collection. When we look at the raw numbers, we find that the two classes for which AdaBoost significantly outperforms Rocchio, *earn* and *acq* are also the classes with the highest number of positive training documents, 2,877 and 1,650, respectively. In general we observe that whenever a class has a large number of relevant documents in its training set, AdaBoost tends to achieve lower error rates and higher utility values. However, we observe that for Util-2, Rocchio is somewhat better than AdaBoost.

6.3 AP-Body

The results for this dataset, shown in Figure 3 (middle row), are quite parallel to the results for Reuters-21578. Out of the twenty classes, AdaBoost is better than Rocchio in terms of Error and Util-1 for the four largest classes. These classes are: `britx, bush, israel, japan`. Each of these classes has more than 2,000 relevant documents in its training set. However, for Util-2 this advantage is not there. Overall, there isn't much difference in the performance of these two algorithms.

We also timed our algorithms for the AP-Body task. Our current implementation of AdaBoost, which is rather non-optimized, takes on an average 180 minutes per class to learn a classifier. In contrast, our implementation of Rocchio, which can also be further optimized, takes about 3 minutes per class on an average. This difference in running time is significant and makes the use of Rocchio's method quite attractive even if it comes at a slight loss of effectiveness (*e.g.*, for big classes).

6.4 TREC-3

For the TREC task, we once again observe in Figure 3 (bottom row) that there is no noticeable difference in the performance of the two algorithms. We observe in the scatter plots that many points for classes with few relevant training examples are below the x-axis indicating that Rocchio is marginally better for these cases. Overall, for this task as well, there is no noticeable difference between the two methods. As in the other two collections, we observe that for TREC as well, the relative performance of AdaBoost is weaker when evaluated using Util-2, and Rocchio's method is clearly better than AdaBoost for Util-2.

6.5 Analysis

In our view, one of the foremost results of this study is that a state of the art version of Rocchio's algorithm is quite competitive with modern machine learning algorithms for text filtering. This result contradicts the claims made in several previous studies [22, 8, 39, 15] that infer that Rocchio's method is inferior to state of the art machine learning algorithms.

These results show that when there is enough training data to learn from, a principled learning algorithm (AdaBoost), which is derived from theoretical foundations of computational learning and is specifically designed for general classification, does learn a better classifier than an algorithm designed to rank documents (Rocchio) which does minimal learning.

On the contrary, when there is little data to learn from, a strong learning algorithm like AdaBoost stands a chance of over-fitting to the data. For this reason we would have expected Rocchio to consistently outperform AdaBoost for small classes. Even though, for small classes, Rocchio is quite effective at the task it was designed for, namely ranking relevant documents above non-relevant documents, it often fumbles in selecting a threshold for filtering. For example, for the class *nkr* in Reuters-21578, Rocchio achieves an average precision of 0.5085 which is much better than average precision of AdaBoost (0.0018). However, when evaluated using the utility measures, the two algorithms have essentially the same performance indicating that Rocchio was unable to capitalize on its superior ranking. Similar behavior is observed for many other classes, *e.g.*, *dfl*, *instal-debt*, and *sun-meal*. This also indicates why using average precision to evaluate text filtering is not sufficient.

Our current implementation of AdaBoost does not utilize term weights, which are known to be crucial for most IR tasks [5] and are the basis of good performance of Rocchio's algorithm. We believe that AdaBoost would benefit significantly by using term weights, and we are currently studying ways of incorporating these weights into AdaBoost.

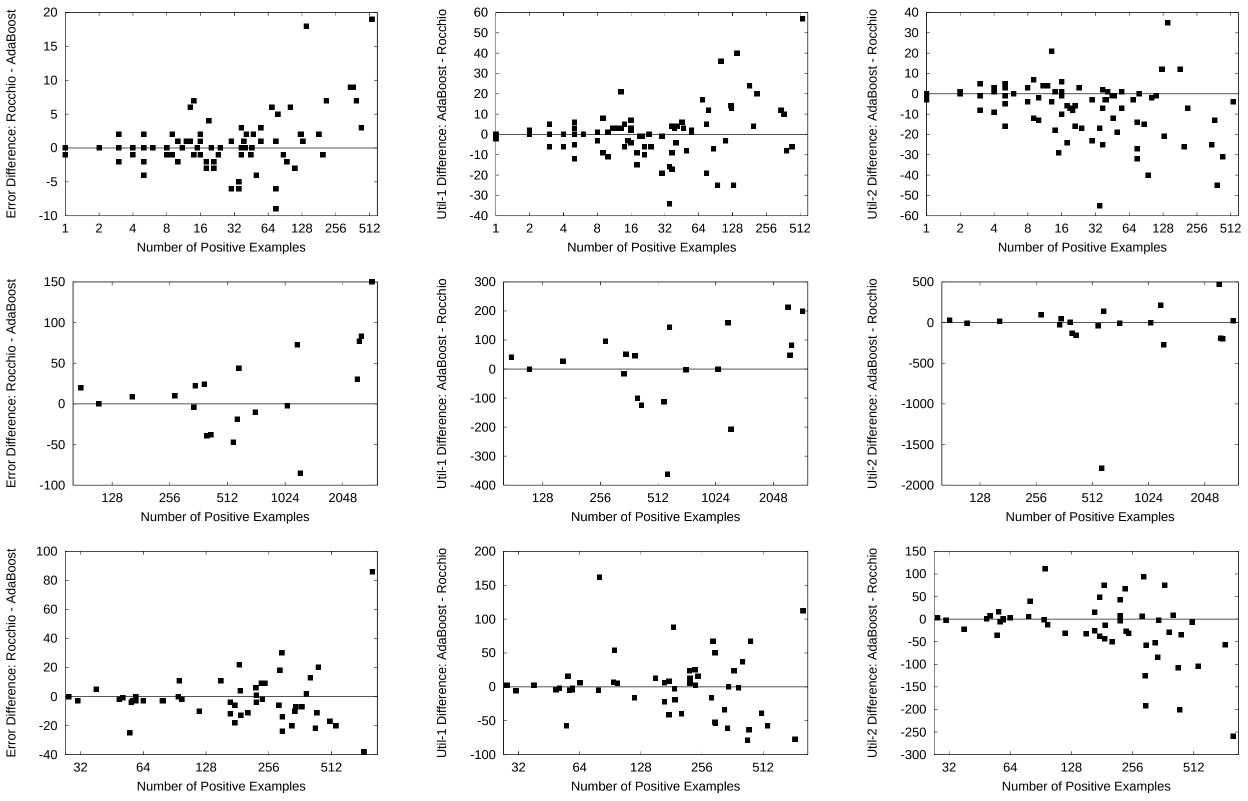

Figure 3: Comparison of AdaBoost and Rocchio.

These results also show that for all tasks, filtering for Util-2, when a relevant documents is much more important than a non-relevant document, is a hard job. For many classes, both the algorithms get a negative Util-2 value, indicating that they are having troubles classifying for this measure. Apparently Rocchio is more robust to this skew in the relative importance of relevant documents. AdaBoost doesn't do as well for Util-2 as it does for error and Util-1 when compared to Rocchio. We haven't studied this phenomenon in greater details yet, but we suspect that the high importance of relevant documents is forcing AdaBoost to select non-general features from the relevant documents, which results in over-fitting of the classifier to the training relevant documents.

The poor performance of AdaBoost on TREC can possibly be attributed to the sampling used to train the algorithm for this task (see Section 5.4). When AdaBoost is trained using a small set of training documents, it is unable to learn the global occurrence pattern for words. Rocchio, on the other hand, uses this information as the idf-factor (Table 1) in term weights. Due to such sampling, AdaBoost tends to over-emphasize common terms that happen to occur in the relevant documents in the sample of documents it is trained on. Had it been trained on the entire collections, this wouldn't be a problem, and the results should have been better.

7 Conclusions

We have developed two effective algorithms for text filtering. Both algorithms, the first, AdaBoost, based on recent developments in the Machine Learning community, and the second based on Rocchio's method which was developed in the Information Retrieval community, are quite competitive. We find that AdaBoost is significantly better than Sleeping-Experts, another effective text filtering algorithms available from prior research. AdaBoost is also better than Rocchio if there is a large number (hundreds or even thousands) of relevant documents to learn from. Otherwise, there is no noticeable difference between the performance of the two algorithms. It is not clear how often we get a large number of relevant documents in an operational text filtering system. From our current experiments, Rocchio is significantly faster than AdaBoost. Thus, it seems that classifiers based on Rocchio's method are and would be viable tools in large scale filtering systems. In order to make AdaBoost more attractive for large problems with sparse relevant documents, algorithmic improvements that will significantly reduce the computation time should be sought.

Acknowledgments

We would like to thank David Lewis for help with the data and for his useful comments on this work.

References

[1] James Allan. Incremental relevance feedback for information filtering. In *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 270–278. Association for Computing Machinery, New York, August 1996.

[2] Chidanand Apte, Fred Damerau, and Sholom Weiss. Towards language independent automated learning of text categorisation methods. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 23–30, July 1994.

[3] Avrim Blum. Empirical support for winnow and weighted-majority based algorithms: results on a calendar scheduling domain. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 64–72, 1995.

Per class error, utility, average precision, and F-measure values for this study are available from:

`http://www.research.att.com/∼singhal/sigir98-rocboost.dat`

[4] Leo Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California at Berkeley, 1996.

[5] Chris Buckley. The importance of proper weighting methods. In *Human Language Technology*. Morgan Kaufman, 1993.

[6] Chris Buckley and Gerard Salton. Optimization of relevance feedback weights. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 351–357, July 1995.

[7] J.P. Callan. Information filtering with inference networks. In *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 262–269. Association for Computing Machinery, New York, August 1996.

[8] William W. Cohen and Yoram Singer. Context-sensitive learning methods for text categorization. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 307–315, 1996.

[9] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[10] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[11] Yoav Freund, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. Using and combining predictors that specialize. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 334–343, 1997.

[12] D. K. Harman. Overview of the third Text REtrieval Conference (TREC-3). In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, pages 1–19. NIST Special Publication 500-225, April 1995.

[13] David Hull. The TREC-6 filtering track: Description and analysis. In *Proceedings of the Sixth Text REtrieval Conference (TREC-6)*, 1998 (to appear).

[14] David Hull, Jan Pedersen, and Hinrich Schutze. Method combination for document filtering. In *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 279–288, August 1996.

[15] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning (ECML'98)*, 1998 (to appear).

[16] David Lewis. Personal Communication.

[17] David Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 37–50, June 1992.

[18] David Lewis. Evaluating and optimizing autonomous text classification systems. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 246–255, July 1995.

[19] David Lewis. The TERC-5 filtering track. In *Proceedings of the Fifth Text REtrieval Conference (TREC-5)*, pages 75–96. NIST Special Publication 500-238, November 1997.

[20] David Lewis and William Gale. A sequential algorithm for training text classifiers. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, July 1994.

[21] David Lewis and Mark Ringuette. A comparison of two learning algorithms for text categorization. In *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, pages 81–93, Las Vegas, NV, April 1994.

[22] David Lewis, Robert Schapire, James Callan, and Ron Papka. Training algorithm for linear text classifiers. In *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 298–306, August 1996.

[23] Andrew McCallum and Kamal Nigam. Employing em in pool-based active learning for text classification. In *ICML-98*, 1998 (to appear).

[24] H.T. Ng, W.B. Gog, and K.L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of the Twentieth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 67–73, July 1997.

[25] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730, 1996.

[26] S.E. Robertson and S. Walker. Some simple effective approximations to the 2–poisson model for probabilistic weighted retrieval. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–241. Springer-Verlag, New York, July 1994.

[27] J.J. Rocchio. *Document Retrieval Systems–Optimization and Evaluation*. PhD thesis, Harvard Computational Laboratory, Cambridge, MA, 1966.

[28] J.J. Rocchio. Relevance feedback in information retrieval. In *The SMART Retrieval System—Experiments in Automatic Document Processing*, pages 313–323, Englewood Cliffs, NJ, 1971. Prentice Hall, Inc.

[29] Gerard Salton. *Automatic text processing—the transformation, analysis and retrieval of information by computer*. Addison-Wesley Publishing Co., Reading, MA, 1989.

[30] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, 1990.

[31] Gerard Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill Book Co., New York, 1983.

[32] Gerard Salton, A. Wong, and C.S. Yang. A vector space model for information retrieval. *Communications of the ACM*, 18(11):613–620, November 1975.

[33] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Machine Learning: Proceedings of the Fourteenth International Conference*, 1997.

[34] Amit Singhal. AT&T at TREC-6. In *Proceedings of the Sixth Text REtrieval Conference (TREC-6)*, 1998 (to appear).

[35] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–29. Association for Computing Machinery, New York, August 1996.

[36] Amit Singhal, Mandar Mitra, and Chris Buckley. Learning routing queries in a query zone. In *Proceedings of the Twentieth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 25–32, July 1997.

[37] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[38] E. M. Voorhees and D. K. Harman. Overview of the sixth Text REtrieval Conference (TREC-6). In E. M. Voorhees and D. K. Harman, editors, *Proceedings of the Sixth Text REtrieval Conference (TREC-6)*, 1998 (to appear).

[39] Yimin Yang. An evaluation of statistical approaches to text categorization. Technical Report CMU-CS-97-127, School of Computer Science, Carnegie Mellon University, April 1997.

[40] Yiming Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 13–22, July 1994.

[41] Yiming Yang. Noise reduction in a statistical approach to text categorization. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–263, July 1995.