Boosting for Document Routing

Raj D. Iyer[*] David D. Lewis[†] Robert E. Schapire Yoram Singer[‡] Amit Singhal

AT&T Labs – Research
Shannon Laboratory
180 Park Avenue
Florham Park, NJ 07932 USA

ABSTRACT

RankBoost is a recently proposed algorithm for learning ranking functions. It is simple to implement and has strong justifications from computational learning theory. We describe the algorithm and present experimental results on applying it to the document routing problem. The first set of results applies RankBoost to a text representation produced using modern term weighting methods. Performance of RankBoost is somewhat inferior to that of a state-of-the-art routing algorithm which is, however, more complex and less theoretically justified than RankBoost. RankBoost achieves comparable performance to the state-of-the-art algorithm when combined with feature or example selection heuristics. Our second set of results examines the behavior of RankBoost when it has to learn not only a ranking function but also all aspects of term weighting from raw data. Performance is usually, though not always, less good here, but the term weighting functions implicit in the resulting ranking functions are intriguing, and the approach could easily be adapted to mixtures of textual and nontextual data.

Keywords: routing, boosting, ranking, supervised learning, text representation.

1. INTRODUCTION

Recent years have seen an explosion in applications of machine learning to information retrieval. Some benefits of this interest have been algorithms with clear theoretical properties, the ability to handle both textual and nontextual data, and, occasionally, better effectiveness. However, this new attention has fallen disproportionately on a subset of IR tasks, in particular the classification tasks of text categorization and document filtering.

Classification is not the only behavior one would like from IR systems. Of even more interest is *ranking*. Ranking of documents by some measure of relevance is an extremely effective interface strategy, one now used millions of times a day by World Wide Web search engines.

IR has developed many approaches for training models to rank documents. These have been applied to both interactive searches of an existing collection, called *ad-hoc retrieval*, and to prioritizing new documents that arrive in the future, a task which has come to be called *routing*.

Most IR techniques for ranking have two facets [13]. One is the selection and weighting of terms for a particular information need. When only a textual query is available, this process is called *query weighting*. When training data is available in the form of documents judged for relevance, the process is called *relevance feedback* in an interactive context and *fitting*, *training*, *tuning*, or *optimization* in a routing context; we will use the term *model fitting*. The techniques used are often similar to those explored in machine learning but with the goal of ranking rather than classification.

The second facet of ranking techniques is the conversion of textual documents first into tokens for index terms (words, phrases, n-grams, etc.) and then into numeric vectors. The process of associating a numeric value with the occurrence of an index term in a document is variously called *document weighting*, *within document weighting*, or *term weighting*. We will use the last term. Term weighting has received little attention outside of IR.

This paper reports an attempt to bring machine learning approaches to bear on both of these facets of ranking. We adapt RankBoost [5], a boosting algorithm designed to produce models for ranking objects, to the widely studied routing task.

Our first study applies RankBoost to documents represented using modern IR term weighting techniques. We compare its ability to fit routing models to that of a state-of-the-art ranking algorithm applied to the same text representation. Our second study discards modern term weighting and tests whether RankBoost can simultaneously learn both term weighting functions and a routing model, starting only with raw textual data represented in various forms.

Our results show that RankBoost tends to overfit in the absence of some additional feature selection mechanism. On a restricted feature set however, results are quite competitive, particularly when substantial training data is available. Interesting term weighting functions were learned in a number of cases, suggesting a new approach for term weighting that may be particularly useful for nonstandard text representations.

We begin in Section 2 by discussing previous approaches to term weighting and model fitting for ranking. Section 3 describes the RankBoost algorithm. Section 4 discusses feature engineering to support the learning of term weighting functions. Section 5 reviews our benchmarking methods. Section 6 details our experiments and presents their results. Section 7 concludes the study and points to some future directions.

[*]Current address: Living Wisdom School, 456 College Avenue, Palo Alto, CA 94306. Research conducted while visiting AT&T Labs and with support from an NSF Graduate Fellowship.

[†]Contact author. Email: `ddlewis2@worldnet.att.net`.

[‡]Current address: School of Computer Science & Engineering, The Hebrew University, Jerusalem 91904, Israel. Research conducted while at AT&T Labs.

2. BACKGROUND

A quandary in both relevance feedback and routing scenarios is that the input to the learning algorithm is (almost always) a set of documents classified according to binary relevance judgments, but the trained model must produce a *ranking* of future documents rather than a binary classification. One way of reconciling this mismatch is to train a model to estimate the probability that a document belongs to the relevant class and then rank documents based on this probability [9]. Probabilistic retrieval [20, 19] is based on this notion, and parameter estimation in a Bayes independence framework is the most common approach to training.

Conversely, one can view relevance as a matter of degree, of which binary relevance judgments are only a coarse reflection. A training algorithm observes these coarse judgments on many training documents and attempts to produce a model estimating the underlying degree of relevance, which is then used to rank documents. Early vector space retrieval relevance feedback algorithms [12, 11] embody this view, emphasizing the construction of a prototypical relevant vector to which similarity can be measured.

In recent years, IR researchers working in both frameworks have used increased computing power to search for models that optimize ranking effectiveness on training data [1, 3, 10]. The resulting algorithms have much in common with techniques from machine learning but lack the theoretical analyses (such as proofs of convergence) often pursued in machine learning. Conversely, machine learning has devoted relatively little attention to ranking, although some areas of statistics and social science have paid a bit more attention. This literature is surveyed elsewhere [4].

3. BOOSTING FOR RANKING

In this section we describe our approach to document routing using the RankBoost boosting algorithm [5]. RankBoost is based on Freund and Schapire's AdaBoost algorithm [6] and its recent successor developed by Schapire and Singer [15]. The goal of RankBoost is to produce a statistical model that, when applied to a set of documents, orders them in a fashion that approximates their true order, that is, an ordering according to relevance. This true ordering may be obtained from a human expert providing *feedback* in the form of relevance judgments. We expect the true ordering to rank all relevant documents above all non-relevant documents; formally it is a two-tier partial order. Thus the goal of RankBoost is to produce an order which places as many relevant documents as possible at the top.

There are various methods that can be used to measure the similarity between the approximate order and the true order. RankBoost attempts to minimize one possible measure which we call *pair-wise disagreement*. This is the number of pairs of documents which the approximate order misorders with respect to the true order. In our study this is simply the number of pairs of documents (d_0, d_1) for which the approximate ranking orders d_0 above d_1 but the feedback judges d_1 as relevant and d_0 as non-relevant.

RankBoost approximates the true ordering by combining many simple rankings of the documents. For example, one simple ranking considered in this study is based on the frequency of a particular term, such as an ordering of the documents according to how many times the word “treaty” appears in each. Clearly, ranking a set of documents by the frequency of even the best single term is likely to place many non-relevant documents above relevant documents. However, the idea of boosting is to generate and combine many different simple rankings in a principled manner to produce a single highly accurate ordering.

Formally, the simple rankings are real-valued functions called *weak hypotheses*. Given a ranking h and document d, we refer to $h(d)$ as the $score$ that h assigns d. Also, h orders d above d' if $h(d) > h(d')$. Boosting assumes access to an algorithm or subroutine for generating these rankings, called the *weak learner*. The boosting algorithm calls the weak learner many times to generate many rankings, and these are then combined into a single ordering called the *final* or *combined hypothesis*.

The boosting algorithm proceeds in rounds. One of its main features is that, during the course of its execution, it assigns different *importance weights* to different pairs of training documents. The weights represent how important it is for the weak learner to differentiate between the two documents (to determine which of the two is more relevant). These weights are not maintained for all possible document pairs: since the boosting algorithm's goal is to order relevant documents over non-relevant documents, the crucial document pairs are of the form (d_0, d_1) where the feedback judges d_1 as relevant and d_0 as non-relevant. The weak learner chooses a simple ranking which correctly orders as many pairs as possible, taking into account the greater importance of correctly ordering pairs which have been assigned greater weight. As the algorithm progresses, pairs of documents that are hard to differentiate correctly get higher weights while pairs that are easy to differentiate get lower weights. This in effect forces the weak learner to concentrate on document pairs that have been misordered by previously derived simple rankings.

The final hypothesis orders a set of new documents by assigning to each a real-valued score. The score of a document is a weighted combination of the scores assigned to that document by the weak hypotheses.

A description of RankBoost is shown in Figure 1. RankBoost takes as input a set X of training documents composed of two disjoint subsets: X_1, the set of relevant documents and X_0, the set of non-relevant documents (as judged by a human expert). As just described, RankBoost calls the weak learner WeakLearn repeatedly in a series of rounds. On round s, RankBoost provides WeakLearn with a set of importance weights over the pairs of training documents. In response, WeakLearn computes a weak hypothesis (simple ranking) h_s which, given a document d, assigns it a real-valued score. We later discuss the weak learners that were used in our experiments.

The importance weights are maintained formally as a distribution D over pairs of training documents from $X_0 \times X_1$. Since this distribution changes after each round, we denote the distribution before round s by D_s. The weight of a pair of training documents (d_0, d_1) (d_1 is relevant and d_0 is non-relevant) under distribution D_s is written $D_s(d_0, d_1)$, and we maintain the conditions that $D_s(d_0, d_1) > 0$ and $\sum_{d_0,d_1} D_s(d_0, d_1) = 1$. (Here and below, it is understood that this sum is over all pairs (d_0, d_1) in $X_0 \times X_1$.) Initially we set all the weights equally, that is, $D_1(d_0, d_1) = 1/(|X_0||X_1|)$.

The goal of the weak learner is to find a simple ranking which misorders as few document pairs as possible, relative to the distribution D_s. Formally, the weak learner attempts to find a weak hypothesis h_s with low weighted pair-wise disagreement

$$\text{disagree}_D(h) = \sum_{d_0,d_1:\ h_s(d_0) \geq h_s(d_1)} D_s(d_0, d_1)\ . \qquad (1)$$

This error can be interpreted as the probability of misordering a document pair chosen randomly according to distribution D_s.

Having obtained a hypothesis h_s from WeakLearn, RankBoost next chooses a value $\alpha_s \in \mathbb{R}$ which, intuitively, is the importance assigned to h_s; its computation is discussed below. Next, RankBoost updates the weights of all the document pairs in such a way that pairs which are correctly ordered by h_s (in the sense that $h_s(d_1) > h_s(d_0)$) get a lower weight while misordered pairs get a higher weight (assuming for the moment that $\alpha_s > 0$, as it usually will be). Finally, to ensure that the new weights D_{s+1} form a distribution (so that $\sum_{d_0,d_1} D_{s+1}(d_0,d_1) = 1$), we renormalize the weights, resulting in the update rule shown in Figure 1.

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 set of new documents is then ordered according to the scores assigned by the following final hypothesis H: for each new document d,

$$H(d) = \sum_{s=1}^{T} \alpha_s h_s(d).$$

That is, the predictions of all the weak hypotheses are evaluated on the new document d, and the average of their predictions, weighted by the α_s's, forms the score assigned to d by H. To generate an ordered list, the set of new documents is sorted decreasing by score.

We now discuss the exact choice of α_s. Freund et al. [5] prove that the pair-wise disagreement (Eq. (1)) of the final hypothesis H is bounded above by the product of the normalization factors, $\prod_{s=1}^{T} Z_s$. Thus, to minimize Eq. (1), on each round we should choose α_s to minimize Z_s. For general weak learners whose hypotheses assign documents arbitrary real numbers, they suggest finding α_s via numerical search. This is the method we used in combination with the weak learner WeakReal, discussed in the next section. If the hypotheses generated by the weak learner have the range $\{0,1\}$, as is the case with our other weak learner, WeakThreshold, Freund et al. provide a direct calculation of α_s:

$$\alpha_s = \tfrac{1}{2} \ln \left(\frac{1 + r_s}{1 - r_s} \right) \qquad (2)$$

where

$$r_s = \sum_{d_0,d_1} D(d_0,d_1)(h_s(d_1) - h_s(d_0)). \qquad (3)$$

To understand what this choice entails, suppose that a highly accurate weak hypothesis h_s has been found. Then r_s will be close to 1 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 about as accurate as a random ranking of documents, then r_s will be close to 0 and α_s will also be close to 0. Thus, the updates to the distribution will be quite conservative, and the predictions of h_s in the final hypothesis will receive rather low weight.

For our task, we allow α_s to be negative. This will be the case whenever a weak hypothesis h_s is found with $r_s < 0$, indicating that h_s is negatively correlated with the data. Such a hypothesis can be useful if we use the opposite of its predictions. However, a weak hypothesis h may be output multiple times during the boosting process, and we do not allow its *cumulative* weight $\sum_{s:\, h_s = h} \alpha_s$ to be negative. We impose this restriction to reduce the danger of overfitting.

In our experiments, we implemented a more efficient version of RankBoost given by Freund et al. [5]. Its behavior is exactly the same as that of the pseudocode given in Figure 1. The code in Figure 1 runs in time proportional to $|X_0||X_1|$, whereas the more efficient code (omitted due to the lack of space) runs in time proportional to $|X_0| + |X_1|$. This is a significant speedup for large text collections such as TREC. Note that this time does not include the running time of WeakLearn; however, a similar speedup is possible for all of the weak learners used in our experiments.

```
Input a set $X$ of documents separated into disjoint subsets
$X_1$ of relevant documents and $X_0$ of non-relevant documents;
Initialize $\forall (d_0,d_1) \in X_0 \times X_1 : D_1(d_0,d_1) = 1/(|X_0|\ |X_1|)$.
Do for $s = 1, \ldots, T$:
    1. Train WeakLearn using distribution $D_s$ over $X_0 \times X_1$.
    2. WeakLearn returns a weak hypothesis $h_s : X \to \mathbb{R}$.
    3. Compute $\alpha_s \in \mathbb{R}$.
    4. Update: $\forall (d_0,d_1) \in X_0 \times X_1$ :
       $D_{s+1}(d_0,d_1) = \frac{D_s(d_0,d_1) \exp(-\alpha_s\ (h_s(d_0) - h_s(d_1)))}{Z_s}$
       where $Z_s$ is the normalization factor:
           $Z_s = \sum_{d_0,d_1} D_s(d_0,d_1) \exp(-\alpha_s\ (h_s(d_0) - h_s(d_1)))$ .
Output the final hypothesis: $H(d) = \sum_{s=1}^{T} \alpha_s h_s(d)$.
```

Figure 1: RankBoost algorithm for document routing

3.1 Weak learners

We now discuss the two weak learners used in our experiments. Since a weak learner is called on each round of boosting, we use the notation of the previous section, omitting s subscripts. A weak learner takes as input a distribution D over the document pairs and a set of N *ranking features*. A ranking feature is a function that assigns a real-valued score to a document. A ranking feature is also allowed to leave some documents unranked, which we indicate by a "score" of $\perp$. For instance, the value of a ranking feature might equal the term frequency of the word *treaty* in the document, or equal $\perp$ if *treaty* does not appear in the document.

The weak learner uses the ranking features to form its weak hypothesis, attempting to find one with small pair-wise disagreement relative to distribution D. Rather than minimize this quantity directly, the weak learner finds a weak hypothesis which minimizes Z, the normalization constant in Figure 1, which is an upper bound on the pair-wise disagreement, as discussed in the previous section.

We describe two weak learners which differ in the types of weak hypotheses they generate. The first weak learner, WeakReal, simply selects one of the available ranking features to be the weak hypothesis. Thus its weak hypotheses are real-valued functions. (In our study, a score of $\perp$ is treated as a zero score for this weak learner.) The second weak learner, WeakThreshold, also selects a ranking feature, but converts it into a binary (0-1) function by choosing a threshold score. The thresholded feature judges documents with scores above the threshold as relevant (score of 1) and documents with scores below the threshold as non-relevant (score of 0). A score of $\perp$ could be mapped to either 1 or 0, but in this study it is always mapped to 0.

WeakReal. This weak learner searches for a ranking feature which minimizes Z. To achieve this, for each ranking feature, the algorithm calculates the α which minimizes Z by running Newton's method for a fixed number of steps. We were able to implement this numerical search in time proportional to $\sum_{i=1}^{N} |X_{f_i}|$, the sum over all features of the number of documents ranked by each feature (details omitted for lack of space). If the ranking features are associated with terms, for example, then this number is exactly the size of an inverted index for these terms.

WeakThreshold. This weak learner takes a set of ranking features $\{f_i\}$ and outputs a weak hypothesis of the form

$$h(d) = \begin{cases} 1 & \text{if } f_i(d) > \theta \\ 0 & \text{if } f_i(d) \leq \theta \text{ or } f_i(d) = \perp \end{cases} \qquad (4)$$

To implement the weak learner efficiently, we search for h to max-

	Rocchio-QZ-DFO	RankBoost with WeakReal	RankBoost with WeakThreshold
Reuters-21578			
all topics	0.6786	0.6284	0.5836
≥ 5 rel. test	0.8078	0.7501	0.7446

Table 1: Non-interpolated average precision results for Reuters-21578 and TREC-3. RankBoost uses all terms that occur in at least two positive training documents as features. The WeakThreshold version uses the *Q* representation (Section 4.3) based on the same terms. The second Reuters line gives results restricted to the 59 topics with at least five relevant test documents.

imize $|r|$ (as defined in Eq. (3)) rather than minimize Z. This is justified since Freund et al. [5] show that, for binary weak hypotheses weighted using α as computed in Eq. (2), $Z \leq \sqrt{1-r^2}$. WeakThreshold searches for h by checking all possible thresholds of all features. This can be done very efficiently, as described by Freund et al. [5] who show how the search can can be carried out in time proportional to $\sum_{i=1}^{N} |X_{f_i}|$, the same running time as WeakReal. This is the implementation we used in our experiments.

Note that WeakThreshold treats unranked documents as if they are ranked below every ranked document.

Choosing the number of rounds. Finally, we need to specify how we set the number of rounds T. Intuitively, as boosting runs for more rounds, the final hypothesis it outputs grows larger and more complicated. Although a larger final hypothesis predicts more accurately on the training data, it may overfit and not generalize well to future data. Thus we want to use the training data to select T in a fashion that minimizes the risk of overfitting. Although there are theoretical analyses of the number of rounds needed for boosting [6, 14], these tend not to give practical answers; therefore, we use heuristics to estimate a good number of rounds of boosting. We found that the following rule yields good results. When running on a particular topic, we looked at the number of features in the feature set (see Section 6.1) and the number of positive examples in the training set, and we set T to be the smaller of the two.

4. TERM WEIGHTING BY WEAK HYPOTHESIS FORMATION

While RankBoost is quite different from typical learning approaches to routing, the ranking function it produces when used with WeakReal has a common form: a linear model applied to real-valued term weights. Since term weighting approaches for handling within document frequency and document length have been highly optimized for use with linear models, there seems little reason to explore alternative term weighting approaches when WeakReal is used.

However, when RankBoost is used with WeakThreshold, the ranking function is a linear combination of thresholded feature values, a model structure rarely used in IR. Applying a standard term weighting function seems peculiar with such a model, since the real-valued feature values would simply be thresholded to 0/1. We instead explored the possibility of using a very raw representation of the document content. RankBoost can then implicitly construct a separate term weighting function for each term by choosing and weighting multiple thresholded weak hypotheses for the term. We describe in this section the nature of the term weighting functions produced using each of several raw representations of text.

	Rocchio-QZ-DFO	RankBoost with WeakReal	RankBoost with WeakThreshold
Reuters-21578			
all topics	0.6786	0.6956	0.5817
≥ 5 rel. test	0.8078	0.8455	0.7483
TREC-3			
all documents	0.4669	0.3974	0.4276
top 1,000 + rel.	0.4669	0.4638	–

Table 2: Non-interpolated average precision results for Reuters-21578 and TREC-3. RankBoost here is restricted to using features given a nonzero weight by Rocchio-QZ-DFO. The WeakThreshold version uses the *Q* representation (Section 4.3). The second Reuters line gives results only on topics with at least five relevant test documents. The second TREC line gives results for the top 1,000 test documents as ranked by Rocchio-QZ-DFO, plus any remaining relevant test documents.

4.1 The *rawtf* representation

In the simplest case we define one ranking feature, f_t, for each term t, where $f_t(d)$ is equal to the raw *tf* (number of occurrences) of term t in document d. If there are no occurrences of t in d, we can view the feature as not ranking the document ($f_t(d) = \perp$) or as having a *tf* of 0 ($f_t(d) = 0$): the effect is the same with our use of WeakThreshold. We call this the *rawtf* representation.

A weak hypothesis $h(d)$ produced by boosting with the *rawtf* representation has the form:

$$h(d) = \begin{cases} 1 & \text{if } tf \geq \theta \\ 0 & \text{if } tf < \theta \end{cases}$$

where θ is a threshold, and *tf* is the within document frequency of some term. Each such hypothesis is given a weight α by the boosting algorithm, so the above hypothesis contributes α to a document's score if the *tf* of the selected term is $\geq \theta$ and contributes 0 otherwise.

However, the final hypothesis $H(d)$ used to rank documents is a weighted combination of many weak hypotheses of the above form. RankBoost may choose several weak hypotheses based on the same indexing term. Each such hypothesis may have a different threshold θ and weight α. Therefore, the overall contribution of a term to document d's score is a step function produced by summing all weak hypotheses derived from that term:

$$g(d) = \begin{cases} w_n & \text{if } tf \geq \theta_n \\ w_{n-1} & \text{if } tf \geq \theta_{n-1} \text{ and } tf < \theta_n \\ w_{n-2} & \text{if } tf \geq \theta_{n-2} \text{ and } tf < \theta_{n-1} \\ \dots & \\ 0 & \text{if } tf < \theta_1 \end{cases}$$

where $\theta_n > \theta_{n-1} > \cdots > \theta_1$ are the set of thresholds seen in weak hypotheses for this term. The w_j's result from summing the α's for all weak hypotheses that are based on the term and that have thresholds $\leq \theta_j$. Note that this implies that $w_n \geq w_{n-1} \geq \cdots \geq 0$, since, as discussed in Section 3.1, we force WeakThreshold to produce a cumlative nonnegative weight for each weak hypothesis.

In other words, as in conventional within document weighting functions, larger values of *tf* cause a term to have a larger contribution to a document's score. The differences are that (1) the within document weighting function is a step function rather than a continuous function such as $\log$ *tf*, and (2) the function is learned rather than prespecified and is different for each term.

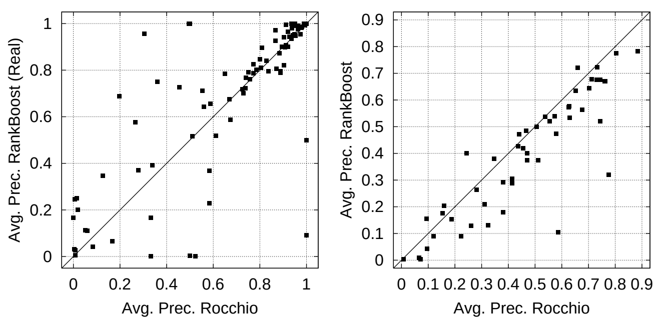

Figure 2: Comparison of RankBoost with WeakReal and Rocchio-QZ-DFO on Reuters-21578 (left) and TREC-3 (right) text collections. Each point in each scatterplot shows the test average precision of the two competing algorithms on a single topic. The x- and y-coordinates of each point give the test average precision of Rocchio-QZ-DFO and RankBoost (respectively) on the given topic.

4.2 The *length* feature and the *rawtf&length* representation

It is common in term weighting to use *document length normalization*, i.e. to reduce the scores that long documents would otherwise get. The rationale is that long documents will tend to have higher *tf* values for all terms, regardless of the actual content of the document. Document length normalization approaches such as cosine normalization or pivoted normalization [17] use the same normalization formula for each ranking task that is presented.

We investigated learning a customized document length normalization for each ranking task. We defined a ranking feature, *length(d)*, whose value equals the negative of the document length if the term occurs in the document and $\perp$ otherwise. We negate the length because WeakThreshold uses positive weights and $\geq$ tests on thresholds, and we want shorter documents to get a boost to their score, not longer ones. A weak hypothesis based on the *length* feature has the form:

$$h(d) = \begin{cases} 1 & \text{if } \textit{length} \leq \theta \\ 0 & \text{if } \textit{length} > \theta \end{cases}$$

where we have reversed the inequalities rather than writing the negated document lengths. As with all weak hypotheses, this one will have some weight α. The cumulative effect of all weak hypotheses based on the length feature is a step function that gives a large positive contribution to short documents with the contribution decreasing as document length increases. (Section 6.2 gives an example.)

We refer to the representation that includes both the *rawtf* and *length* features as the *rawtf&length* representation.

4.3 The *Q* (quadrant) representation

The *length* feature allows only an overall downweighting of long documents. It is also possible to define features that allow a document length to affect each term differently. In this representation, we define multiple ranking features for each term, one for each *tf* value observed for the term on the training corpus. The value of the feature $f_{t,u}$ on document d is the negated document length if that term has a *tf* $\geq u$ on the document and $\perp$ otherwise.

By selecting a feature $f_{t,u}$, WeakThreshold is implicitly choosing a threshold of u on the *tf* value of t. Then, as always, WeakThreshold explicitly chooses a threshold θ on $f_{t,u}(d)$, the negated length of d. The resulting weak hypothesis therefore is based on

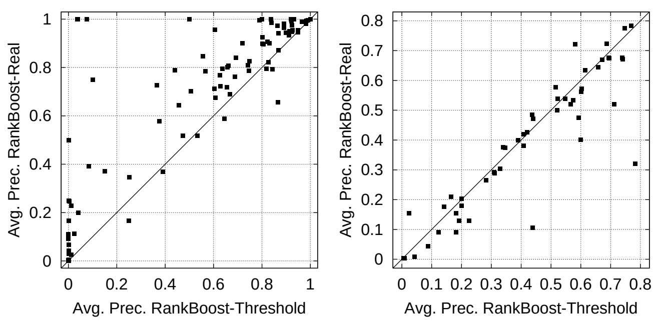

Figure 3: Comparison of RankBoost with two weak learners, WeakReal and WeakThreshold on Reuters-21578 (left) and TREC-3 (right) text collections. WeakThreshold uses the Q_r representation. (See Figure 2.)

two thresholds:

$$h(d) = \begin{cases} 1 & \text{if } \textit{tf} \geq u \text{ and } \textit{length} \leq \theta \\ 0 & \text{otherwise.} \end{cases}$$

We call this representation the *quadrant* or *Q* representation since the set of documents assigned a value of 1 by a weak hypothesis is a quadrant of *length* $\times$ *tf* space. The cumulative effect of all weak hypotheses based on a single term is a complicated function of document length and *tf* value. (See Figure 5 for an example.) We hypothesized that the functions learned in this manner would be similar and possibly superior to conventional term weighting functions. Section 6.2 examines this question.

5. METHODS

This section reviews the datasets that we tested the boosting algorithms on and the standard IR techniques that we compared them to.

5.1 Datasets

Our studies used two datasets previously used in a study of boosting for text classification [16] and used the same procedures for processing those datasets. The TREC-3 routing dataset consists of 741,856 training documents and 336,310 test documents distributed on TREC disks 1-3. We used fifty TREC topics (numbers 101-150) and corresponding relevance judgments from the TREC-3 routing evaluation [8]. Textual queries for each topic exist but were not used in our experiments.

While the TREC-3 routing dataset and other TREC routing collections are the most widely used benchmark for machine learning of ranking functions, their size was problematic for our prototype code. We therefore used as our second dataset the much smaller Reuters-21578 collection. The Reuters-21578 collection consists of 21,578 documents which appeared on the Reuters newswire in 1987. Each document has been categorized with respect to each one of 135 financial topic categories. We used the "ModApte" split of the data into 9,603 training documents, 3,299 test documents, and 8,676 documents which are ignored. We used the 90 categories that have at least one positive training instance and at least one positive test instance under this split. Some results are also presented for the set of 59 categories that have at least one positive training instance and at least *five* positive test instances under this split, since test set average precision figures are less erratic for these categories.[1]

[1]More details on the collection are available at `http://www.research.att.com/~lewis.`

Both datasets were indexed using standard SMART tokenization, stoplists, and phrase formation [16]. Experiments used either raw *tf* (SMART triple *nnn*) or log *tf* pivoted normalization (*Lnu*) versions of the data produced by SMART.

Note that most studies on Reuters-21578 have used effectiveness measures for binary classification. Our use of Reuters-21578 is as a surrogate routing dataset, and so we evaluate our results on both it and TREC-3 using non-interpolated average precision, a widely used measure of ranking effectiveness. Average precision was computed using the standard TREC evaluation software.

5.2 Comparing with a state-of-the-art routing method

A wide range of highly tuned learning algorithms for producing ranking functions have been explored in the TREC routing evaluations. Some of the consistently most successful approaches [2, 21] are multipass optimization algorithms initialized using Rocchio's relevance feedback formula [12, 11]. We therefore compared the effectiveness of RankBoost with the latest in this series of algorithms. This version, which we will call Rocchio-QZ-DFO here, incorporates dynamic feedback optimization [3] and query zoning [18]. The algorithm has 5 parameterized phases and is described in detail elsewhere [16].

6. EXPERIMENTS

We now report our experimental results on applying RankBoost to train ranking models. Study 1 focuses on issues of model fitting, while Study 2 focuses on the character of learned term weighting functions.

6.1 Study 1: Boosting with real-valued weak hypotheses

Our first study focuses on the properties of RankBoost using WeakReal as a weak learner applied in a standard IR context. We therefore used a standard text representation for documents consisting of *Lnu* weighted words and phrases. That is, each ranking feature is associated with a term whose value is given by its *Lnu* weight for that document. The resulting final hypothesis is therefore a linear combination of the within document weights for a set of terms.

We first ran RankBoost with candidate weak hypotheses corresponding to all terms that occurred in at least two positive training instances of the class. With this as the only feature selection, RankBoost's performance was worse than Rocchio-QZ-DFO's on the Reuters collection (Table 1). RankBoost gave better average precision than Rocchio-QZ-DFO on only 22 of the topics, while Rocchio-QZ-DFO was better on 61. Averaged over the 90 topics, RankBoost's non-interpolated average precision was 7.4% worse than Rocchio-QZ-DFO's. If we restricted topics to those that had at least 5 test examples, the difference was still 7.1%.

These initial results suggested that RankBoost was doing a better job of weighting features than selecting them, so we conjectured that restricting it to use a high quality feature set would result in better effectiveness. We tested this by simply restricting RankBoost to use exactly the features that Rocchio-QZ-DFO selects. This significantly improved the effectiveness of RankBoost/WeakReal (from 0.6284 to 0.6956), as seen in Table 2, so that RankBoost now is actually performing slightly better than Rocchio-QZ-DFO. Figure 2 (left) shows in a scatterplot how RankBoost compares to Rocchio-QZ-DFO on all 90 topics in the Reuters collection. Now RankBoost outperforms Rocchio-QZ-DFO on 53 topics, while Rocchio-QZ-DFO has the advantage on only 29 topics (and tied on the remaining 8 topics).

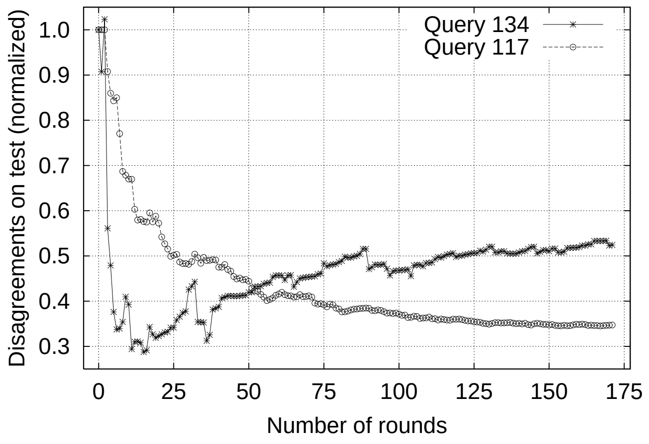

Figure 4: For two TREC-3 topics, the pairwise disagreement on a test set for RankBoost with WeakReal, measured as a function of the number of rounds of boosting.

We next compared RankBoost (with Rocchio-QZ-DFO feature selection) to Rocchio-QZ-DFO on the TREC-3 collection. Our current implementation of RankBoost with WeakReal is too slow to run on all of the available training data, even with feature selection. We instead trained RankBoost using the top 10,000 documents ranked by Rocchio-QZ-DFO, plus any remaining relevant documents. Rocchio-QZ-DFO, on the other hand, was trained on the entire corpus. As Table 2 and Figure 2 (right) show, RankBoost's performance (0.3974 average precision) was still worse than Rocchio-QZ-DFO's (0.4669 average precision), though it is still better than all but one of the 49 original TREC-3 runs [8].

We believe that the inferior performance of RankBoost on the TREC-3 data is due in large part to the limited dataset used in training, since it certainly had very different statistical properties from either the entire training set or the entire test set. To test this hypothesis, we evaluated RankBoost on a test set selected in a similar manner to that used in choosing the training set, namely, the top 1,000 test documents (as ranked by Rocchio-QZ-DFO) together with all remaining relevant documents. On this more limited test set, RankBoost's performance (0.4638 average precision) was comparable to that of Rocchio-QZ-DFO, as can be seen in Table 2. (Rocchio-QZ-DFO's effectiveness does not change when evaluated on this subset, because the TREC evaluation software only takes into account the top 1000 retrieved documents when computing effectiveness.)

One of the problematic aspects of RankBoost is choosing the number of rounds of boosting to perform. In the above experiments, we used the heuristic presented in Section 3. How critical is the choice of the number of rounds? We found that RankBoost tended to exhibit two sorts of behavior. On some topics, RankBoost overfit badly, meaning that the pairwise disagreement on a test set, as a function of the number of rounds of boosting, quickly reached a minimum and then rose significantly. This can be seen in the learning curve for TREC-3 topic 134 in Figure 4. On other topics, however, RankBoost did not overfit; instead, the pairwise disagreement continued to drop, eventually reaching an asymptote. This behavior is seen in Figure 4 for TREC-3 topic 117. The former case seems to occur on topics with few relevant documents, and this also is the case in which RankBoost's performance is most likely to be worse than Rocchio-QZ-DFO's. Conversely, RankBoost tends to perform much better in the latter case. Thus, overfitting on topics with few relevant documents seems to be a significant cause of

	rawtf	*rawtf&length*	*Q*
Reuters-21578			
all features	0.6420	0.6557	0.5836
Rocchio features	0.5834	0.5952	0.5638
TREC-3			
all features	0.3797	–	0.4233
Rocchio features	0.3909	–	0.4276

Table 3: Non-interpolated average precision results for Reuters-21578 and TREC-3 using RankBoost with WeakThreshold. Six text representations are compared: *rawtf*, *rawtf&length*, and *Q* (Section 4.3), plus the versions of these restricted to features selected by Rocchio ($rawtf_r$, *rawtf&length*$_r$, and Q_r). TREC-3 data for the *rawtf&length* representations was lost due to a bug.

failure for RankBoost.

6.2 Study 2: Learning term-weighting functions from raw data

Our second study tested the ability of RankBoost to extract effective term weighting functions from raw data. We began with raw *tf* (*nnn*) vectors and converted them into sets of ranking features of the forms *rawtf*, *rawtf&length*, and *Q* described in Section 4. Document length was computed as the sum of raw *tf* values for words (not phrases) in that document. We also produced pruned sets of ranking features for each topic using only the features assigned nonzero weight by Rocchio-QZ-DFO (as in Study 1). We refer to these pruned feature sets as $rawtf_r$, *rawtf&length*$_r$, and Q_r. Results with these Rocchio-QZ-DFO-pruned feature sets were slightly better in general, so we concentrate on them here.

RankBoost with the WeakThreshold weak learner was applied to training data represented in each of these forms. Boosting stopped after the number of rounds specified by the heuristic in Section 3.1. The learned model was run on the test data as represented by the same type of ranking features used in the training.

Results are summarized in Table 3. Restricting the representations to features chosen by Rocchio-QZ-DFO helps for TREC-3 but hurts for Reuters, perhaps because of the larger amount of labeled data available for Reuters. The *Q* representations are better for TREC-3 but worse for Reuters, probably due to the greater range of document lengths for TREC-3. In no case does the mean value of noninterpolated average precision for RankBoost with WeakThreshold exceed that for Rocchio-QZ-DFO. Table 2 compares the Q_r results for RankBoost/WeakThreshold with those for RankBoost/WeakReal and Rocchio-QZ-DFO. Figure 3 plots this data in more detail. RankBoost/WeakReal substantially outperforms RankBoost/WeakThreshold (again using the Q_r representation) on Reuters, while results are mixed on TREC. Overall, the thresholding method did not appear to have produced term weighting quite as good as the traditional highly tuned methods.

We hypothesized in Section 4 that RankBoost/WeakThreshold would, by selecting multiple weak hypotheses based on the same term but with different thresholds, learn topic-specific within document weighting functions. This happened in some cases, but more often all weak hypotheses for a term used the same threshold, particularly with the *rawtf* and *rawtf&length* variants. In retrospect, this is not surprising. For boosting to select different thresholds of a ranking feature on different rounds, that ranking feature must have not only substantial overlap with both positive and negative training instances but also a substantial variation in raw *tf* values among those instances.

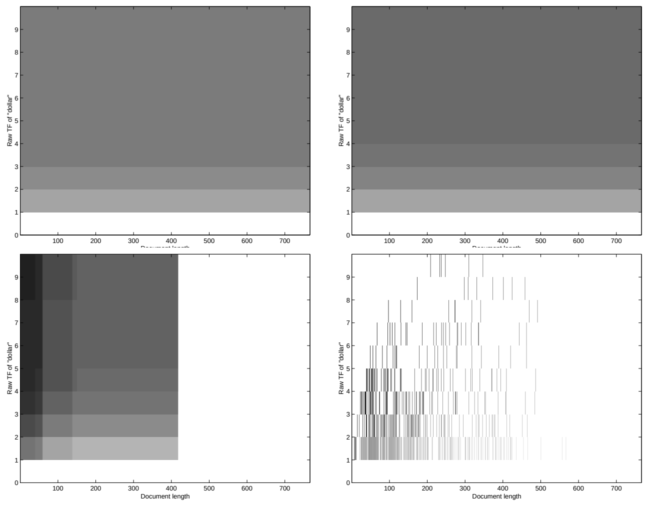

Figure 5: Contribution of *dollar* for different *tf* values and document lengths under $rawtf_r$, *rawtf&length*$_r$, Q_r, and *Lnu* representations.

For our *Q* features, however, the feature values were document lengths. These vary more than *tf* values, so selection of multiple thresholds was more common. We give a particularly interesting example in the next section.

An example of learned weighting functions. Figure 5 plots four term weighting functions for the term *dollar* on Reuters topic *DLR* (stories about the US dollar). This term, the best predictor for the DLR topic, has a nonzero *tf* on 628 Reuters training documents, 124 of which are drawn from the 131 positive training instances of the *DLR* topic.

The x-axis of the plots is the document length, and the y-axis is the raw *tf* of *dollar*. The gray scale intensity encodes the contribution that *dollar* makes to the score of a document with this length and this *dollar tf*. Intensities are normalized so that an average length document (75 non-stopword tokens) with a *dollar tf* of 1 has the same intensity on all graphs.

The upper left plot shows the contribution of *dollar* for the $rawtf_r$ model. A total of 19 weak hypotheses with 3 thresholds among them were selected in the first 69 rounds. Document length is not explicitly represented in this feature set, so the model can take it into account only by giving high *tf*'s slightly less weight than it might otherwise do.

This can be seen (though only barely without a color plot) by comparing the upper left and upper right plot. The upper right plot shows the contribution of *dollar* (19 weak hypotheses covering 4 thresholds) for the *rawtf&length*$_r$ model. The contribution of a *dollar tf* of 4 is 5.48811 in the *rawtf&length*$_r$ model, which has the ability to downweight long documents, but only 5.07414 in the $rawtf_r$ model. More impact can be given larger tf's in the *rawtf&length*$_r$ model, because the *length* feature keeps long documents from having inappropriately high scores.

The *rawtf&length*$_r$ model in fact incorporates 20 weak hypotheses (covering 10 thresholds) derived from the *length* feature. The combined effect of these is a decreasing contribution for document length, ranging, in 11 steps, from 3.30744 for documents with length 11 or less, down to 0.0 for documents with length greater than 149.

The lower left plot shows the effect of the 19 weak hypotheses for *dollar* in the Q_r model. Among the 19 weak hypotheses are

11 distinct ones representing 6 different *tf* thresholds and 8 different length thresholds. The effect is similar to that of standard term weighting functions: a contribution that increases with *tf* at sublinear pace, but where high *tf*s have less impact for long documents than short ones.

However, the plot also shows a glaring problem with the resulting term weighting function. Documents with a length greater than 430 get no contribution for *dollar*, no matter how many times it appears. The problem is that any given training set has a longest positive example, and RankBoost will see no advantage to setting a threshold longer than that length. For our approach to be practical, the system will need to be given some bias in favor of infinite length thresholds.

For comparison, the lower right plot shows the contribution of *dollar* under *Lnu* weighting, as used by RankBoost/WeakReal and Rocchio-QZ-DFO. Since *Lnu* weights are not a function of the sum of *tf*'s, we cannot compute *Lnu*'s for hypothetical documents on the grid. We instead plot the actual value of *dollar*'s *Lnu* weight for all Reuters training and test documents. The pattern is notably similar to that learned with the Q_r representation, showing that, for this term at least, RankBoost has induced a sensible term weighting function from raw data.

7. CONCLUSIONS AND FUTURE WORK

We have presented preliminary evidence that RankBoost, a simple boosting algorithm with strong theoretical foundations, can learn ranking models with effectiveness roughly comparable to that of leading edge routing algorithms developed in IR. We believe we can strengthen the case for RankBoost by replacing our use of Rocchio-QZ-DFO for feature selection (a choice made for convenience) with a simpler and theoretically motivated feature selection method. We also hope to produce a more efficient implementation of RankBoost that would eliminate the need for example selection.

Our results show that a boosting approach applied to a raw characterization of document content (*tf* weights and document lengths) can, in some cases, learn term weighting functions similar to those used in IR. For most terms, however, only one or two distinct weak hypotheses were used, giving a crude treatment of term weighting for that term.

Attempting to learn a separate term weighting function for each term is perhaps too much to ask from any reasonably sized training set. An alternative would be to allow great flexibility in learning the term weighting function, but force the same function to be used for all terms. This might be viewed as an automated version of the exploratory data analysis approach to term weighting proposed by Greiff [7].

In any case, our approach of inducing term weighting functions from raw data is perhaps of less interest for ranking based on textual features (where highly refined term weighting methods are already known) than for ranking based on nontextual or mixed nontext/text data, a less well studied task that is of interest in text mining applications.

As a final note, all our experiments assumed binary relevance feedback, but RankBoost can equally well work with graded relevance judgments. This raises the possibility of learning when *no* completely relevant examples are available, as long as judgments of degree of partial relevance are possible. This is often the case, for instance, in Web search, and we plan to investigate the use of RankBoost there.

8. REFERENCES

[1] B.T. Bartell, G.W. Cottrell, and R.K. Belew. Automatic combination of multiple ranked retrieval systems. In *Proceedings of the 17th Annual International Conference on Research and Development in Information Retrieval*, 1994.

[2] Chris Buckley, James Allan, and Gerard Salton. Automatic routing and ad-hoc retrieval using SMART: TREC 2. In *Proceedings of the Second Text Retrieval Conference*, pages 45–56, 1994.

[3] Chris Buckley and Gerard Salton. Optimization of relevance feedback weights. In *Proceedings of the 18th Annual International Conference on Research and Development in Information Retrieval*, pages 351–357, July 1995.

[4] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[5] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. In *Machine Learning: Proceedings of the Fifteenth International Conference*, 1998.

[6] 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, August 1997.

[7] Warren R. Greiff. A theory of term weighting based on exploratory data analysis. In *Proceedings of the 21st International Conference on Research and Development in Information Retrieval*, pages 11–19, 1998.

[8] Donna Harman. Overview of the third text retrieval conference. In *Proceedings of the Third Text Retrieval Conference*, pages 1–27, 1995.

[9] S. E. Robertson. The probability ranking principle in IR. *Journal of Documentation*, 33(4):294–304, December 1977.

[10] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of the Third Text Retrieval Conference*, pages 109–126, 1995.

[11] J. Rocchio. Relevance feedback information retrieval. In *The Smart retrieval system—experiments in automatic document processing*, pages 313–323. Prentice Hall, 1971.

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

[13] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[14] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, October 1998.

[15] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, December 1999.

[16] Robert E. Schapire, Yoram Singer, and Amit Singhal. Boosting and Rocchio applied to text filtering. In *Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval*, 1998.

[17] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *Proceedings of the 19th Annual International Conference on Research and Development in Information Retrieval*, pages 21–29, 1996.

[18] Amit Singhal, Mandar Mitra, and Chris Buckley. Learning routing queries in a query zone. In *Proceedings of the 20th Annual International Conference on Research and Development in Information Retrieval*, pages 25–32, 1997.

[19] Howard Turtle and W. Bruce Croft. Inference networks for document retrieval. In *Proceedings of the 13th Annual International Conference on Research and Development in Information Retrieval*, pages 1–24, 1990.

[20] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, second edition, 1979.

[21] E.M. Voorhees and D.K. Harman. Overview of the sixth text retrieval conference. In *Proceedings of the Sixth Text Retrieval Conference*, pages 1–24, 1998.