Fast Learning of Document Ranking Functions with the Committee Perceptron

Jonathan L. Elsas
jelsas@cs.cmu.edu

Vitor R. Carvalho
vitor@cs.cmu.edu

Jaime G. Carbonell
jgc@cs.cmu.edu

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, 15213

ABSTRACT

This paper presents a new variant of the perceptron algorithm using selective committee averaging (or voting). We apply this agorithm to the problem of learning ranking functions for document retrieval, known as the "Learning to Rank" problem. Most previous algorithms proposed to address this problem focus on minimizing the number of misranked document pairs in the training set. The committee perceptron algorithm improves upon existing solutions by biasing the final solution towards maximizing an arbitrary rank-based performance metrics. This method performs comparably or better than two state-of-the-art rank learning algorithms, and also provides significant training time improvements over those methods, showing over a 45-fold reduction in training time compared to ranking SVM.

Categories and Subject Descriptors

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

General Terms

Design, Algorithms, Experimentation

Keywords

learning to rank, perceptron algorithm, document retrieval

1. INTRODUCTION

Learning to rank objects has become a critical task in all aspects of serving users' information needs: ranking product recommendations based on previous purchase history, ranking online advertisements with respect to information content on a web page and ranking news articles in order of likely interest to users. This paper investigates the ranking problem in the ad-hoc information retrieval domain: ranking documents in response to users' queries. Learning ranking functions in this domain poses several interesting challenges.

When concerned with document retrieval and ranking, the learning algorithm must (1) adapt to increasing richness and complexity of document representations, (2) adapt to varying tasks and user needs, and (3) scale when applied to larger and larger collections.

In traditional ad-hoc information retrieval systems, a ranking function is developed using a small set of features designed to reflect the similarity of a document to query. Typically these features include measures such as query term frequency in the document, inverse document frequency of the query terms and others. A small number of these measures are combined using a few parameters and those parameters are often chosen manually based on experimentation. But, as is often the case as systems develop over time, new features are added to the model, increasing the number of parameters that need to be tuned to optimize the performance of the system. For example, when retrieval systems were extended to the web domain the use of link-based features and structural document features created a much richer representation of the documents [5]. Further exploitation of web-centric features through link network analysis, bookmarking and tagging trends, and clickthrough data may continue to increase the quality of our document representations[1, 22]. Better document representations could improve the performance of retrieval systems significantly, but adding these features to our ranking function inevitably adds more complexity to the retrieval model and more parameters that need to be tuned.

An additional challenge in document ranking is that the objective of a retrieval system can change from task to task, or even from user to user. In some situations such as patent application search, users may be more interested in high recall. In others situations such as named-page-finding, it would be more appropriate to optimize performance metrics designed for very high precision. In extremely large collections like the web, it is particularly important to maximize performance at the top of the ranked list as it has been shown that users rarely look beyond the first few retrieved documents [1].

Finally, as collection sizes increase, the algorithms that learn ranking functions must scale appropriately. State-of-the-art algorithms for learning document ranking functions can be prohibitively slow on large collections.

The question addressed in this paper is: *how to efficiently and automatically learn document ranking functions in order to optimize a task-appropriate performance measure*, particularly those that are focused at the higher end of the ranked list.

To answer this, we propose a new method to learn ranking functions that are biased towards maxmizing arbitrary rank-based performance metrics (such as NDCG, MAP, Reciprocal Rank, etc), while keeping training time fast and scalable to large datasets. This method, the *committee perceptron*, inherits the speed and simplicity of the original perceptron algorithm, thereby allowing the algorithm to run quickly even on large collections. The committee perceptron allows tuning of the committee size across the full operational range, as opposed other perceptron variants which just operate at the extremes. By maintaining a committee of n-best hypotheses during training time, we can also estimate the performance of each hypothesis accurately for any retrieval performance metric, yielding a more task-appropriate solution. The end result is a fast and scalable algorithm that can be easily tuned for an arbitrary retrieval performance metric.

The remainder of this paper is organized as follows: we first describe some of the related work in this area. Then, we present a discussion of optimization of rank-based performance metrics with respect to learning of ranking functions. The committee perceptron algorithm is described, followed by experimental results comparing this algorithm to several other ranking function learners.

2. RELATED WORK

2.1 Problem Setting

Many of the previous approaches to learning ranking functions for document retrieval, as well as the approach presented here, are concerned with functions that can be parameterized by a single vector of weights, $\mathbf{w}$. Documents are represented by a vector of query-dependent feature scores,

$$\mathbf{d}_{iq} = (f_0(d_i, q), f_1(d_i, q), \ldots, f_m(d_i, q))$$

. These feature scoring functions f_i are typically derived from low-level features used in information retrieval systems such as term frequency or inverse document frequency, or higher-level aggregated scores such as the score assigned by a baseline ranking algorithm for document d_i on query q. The learning algorithms presented in this paper output scoring functions of the form:

$$Score(\mathbf{d}_{iq}, \mathbf{w}) =< \mathbf{d}_{iq}, \mathbf{w} > \quad (1)$$

where $< \cdot, \cdot >$ is the inner product. A final document ranking is derived from this function in the obvious way, where documents are ranked in descending order according to their score.

The goal, then, for learning the ranking function is to find the weight vector $\mathbf{w}$ that yields the best performance in the document ranking task.

2.2 Learning Ranking Functions

Learning ranking functions for document retrieval has recently become a popular research topic. RankSVM [10, 22] adapts the SVM classification algorithm to learning ranking functions. RankNet [7] uses a probabilistic cost function for mis-ranking pairs of documents and trains using the gradient of this cost function. RankBoost [16] applies a boosting algorithm (similar to AdaBoost) to the ranking problem, combining the output of weak rankers to learn a powerful ranking function. [30] and [19] apply the perceptron algorithm to ranking (or re-ranking) in several different NLP and IR domains.

Most of the above approaches to learning ranking functions adapt binary classification algorithms to the this task by constructing a training set using relative preference (or relevance levels) of training instances. The ranking problem is concerned with producing scores for instances so that the relevant instances score higher than less relevant instances. Using the relative preference of training instances, we can adapt most binary classification algorithms to learn rankings by training on pairs of instances that have differing relevance levels. A correct classification corresponds to ranking that pair so that the more relevant instance is scored higher than the less relevant instance.

Some of the above learning approaches have also been adapted to learn rankings functions to optimize specific retrieval performance metrics such as NDCG[20]. [27] and [31] present two ways to tailor RankNet to optimize NDCG at the higher end of the ranked list. The first approach iteratively reranks smaller and smaller portions of an originally produced ranked list of documents in order to focus on the portion of the ranked list that has a higher likelihood of being relevant. The second adapts RankNet's cost function to directly optimize parameters in the BM25 ranking function while maximizing NDCG on a held-out set. In recent work, [32] present a method for adapting RankSVM to optimize average precision.

Other common approaches to optimizing specific retrieval performance metrics include grid-search, coordinate ascent and line-search [19, 28]. Although these methods directly optimize any given performance metric, they are generally considered too slow when applied to more than just a few parameters in the model. Grid-search uses brute-force enumeration of all the hypotheses on a lattice in the search space and evaluates the performance metric of interest for each hypothesis. Line-search uses a more focused heuristic exploration, sampling the hypothesis space around the current hypothesis to identify a promising "direction" to move the hypothesis for an update. Finally, coordinate ascent is another directed heuristic exploration of the hypothesis space, sequentially adjusting each parameter to maximize an objective function while holding the other parameters fixed.

3. RANK-BASED PERFORMANCE METRICS & OPTIMIZATION

In the typical classification setting the objective is to minimize the classification error rate. In the context of learning ranking functions with pairwise document preferences, a classification error corresponds to a mis-ranked ("discordant") document pair, or inversion in the ranked list. Although this objective makes intuitive sense and allows us to conveniently apply classification algorithms to the problem

of ranking, it may not be a sufficient objective. As a simple example, consider a situation with three relevant and three non-relevant documents $\{R, R, R, N, N, N\}$. The following three rankings have the same number of inversions in the ranked lists, Q, but different Normalized Discounted Cumulative Gain (NDCG), Reciprocal Rank (RR) and Average Precision (AP) scores [4, 20]:

Ranked List	Q	NDCG	RR	AP
R N R N R N	3	0.783	1.0	0.756
N R R R N N	3	0.810	0.5	0.639
R R N N N R	3	0.907	1.0	0.833

This is a simple example, but it clearly shows that although minimizing Q may be a sensible goal, it is not a sufficient metric to optimize when higher positions in the ranked list are clearly more important than lower positions. This disconnect between the goal of minimizing the number of inversions in the ranked list and maximizing a specific retrieval performance metric poses significant challenges when learning ranking functions with pairwise preference training data.

In [22], it was shown that minimizing the number of discordant document pairs places a lower bound on average precision. By a similar proof presented in the appendix, we show that minimizing the number of inversions will place a lower bound on *any* retrieval evaluation metric that can be written as:

$$\Phi_{Z,m}(r_1, \ldots, r_R) = \frac{1}{Z} \sum_{i=1}^{m} \frac{i}{r_i}$$

where Z is a measure-dependent normalization, m indicates how far down the ranked list to look, r_i is the rank of the ith relevant document and R is the number of relevant documents. Average precision, precision at K, reciprocal rank and R-precision are all common document retrieval evaluation metrics that can be written this way. Although this lower bound validates the pairwise-preference classification approach to learning ranking functions, there is no guarantee on the tightness of the bound. In the section below, we explore a retrieval evaluation metrics that have a closer relationship to the number of discordant document pairs, one of which is *directly* maximized when reducing the number of mis-ranked document pairs.

3.1 BPREF, RankEFF and Misranked Document Pairs

The evaluation metric *bpref* [6] is also bounded from below and the metric *RankEff* [2] is *directly* maximized when minimizing the number of mis-ranked document pairs in the ranked list. *bpref* was designed as a stable performance metric when relevance judgments are incomplete, and *RankEff* builds upon the *bpref* measure, taking into account all retrieved non-relevant documents. Both measures are known to correlate well with average precision in TREC data [2, 9, 6] and *bpref* is currently reported in annual TREC evaluation results [8]. These metrics are defined as:

$$bpref \equiv \frac{1}{R} \sum_{i=1}^{R} 1 - \frac{\sum_{j=1}^{\min(N,R)} I(n_j < r_i)}{\min(N, R)}$$

$$RankEff \equiv \frac{1}{R} \sum_{i=1}^{R} 1 - \frac{\sum_{j=1}^{N} I(n_j < r_i)}{N}$$

where R is the number of judged relevant documents, N is the number of judged non-relevant documents, n_j and r_i are the ranks of non-relevant and relevant documents, and $I(\cdot) \in \{0, 1\}$ is an indicator function. In the *bpref* definition above, $\sum_{j=1}^{\min(N,R)} I(n_j < r_i)$ gives the number of non-relevant documents up to R ranked higher than the relevant document at rank r_i, and summing over all the relevant documents gives a value that is bounded by the number of discordant documents pairs, $Q_R \leq Q$. Assuming $R < N$, which is a safe assumption in document retrieval, *bpref* can therefore be written in terms of Q and Q_R as follows:

$$bpref = 1 - \frac{Q_R}{N \times R} \geq 1 - \frac{Q}{N \times R}$$

Similarly, we can write *RankEff* as:

$$RankEff = 1 - \frac{Q}{N \times R}$$

Therefore, the traditional pairwise-preference approach to learning ranking functions by minimizing Q is equivalent to maximizing a lower bound on *bpref* and directly maximizing *RankEff*. As noted in [10], care must be taken to balance the training data across queries in order to avoid disproportionately weighting those queries with more relevant documents.

Because these are well understood performance metrics and both correlate well with MAP [2, 9, 6], we would argue that when trying to optimize a ranking function on the full ranked list, minimizing Q is an appropriate approach to take. But, when desiring a ranking function that is tailored more towards high-precision at the top of the ranked list, the learning method should be adapted to maximize evaluation metrics more sensitive to this criteria.

4. THE PERCEPTRON ALGORITHM & ITS VARIANTS

The perceptron algorithm, originally proposed by Rosenblatt in 1958 [29], is an error-minimization online learner that uses a simple additive update rule. It can be show that given linearly separable data, the perceptron will converge to a solution that perfectly classifies the data. The reader is referred to [17] for this convergence proof and analysis of the perceptron algorithm's mistake bounds.

The perceptron algorithm has been applied to document ranking function learning [19], and this paper extends that approach. Table 1 describes several variants of the perceptron algorithm as applied to document ranking. The algorithm description throughout this paper assumes that relevance judgments in the test collection are binary, but this can be easily extended to cases with graded relevance levels. In that case, relevant/non-relevant document pairs can be formed by any pair of documents with differing relevance

Table 1: Perceptron Algorithm Variants for Document Ranking Function Learning.

Input: Number of iterations T, List of training document pairs $S = R \times N = \{(\mathbf{d}_{nq}, \mathbf{d}_{rq})\}$ where d_n is non-relevant and d_r is relevant to query q
Output: Parameter Vector $\mathbf{w}$

1. Initialize $i = 0$, success counter $c_i = 0$, initial parameters $\mathbf{w}^0$, query balancing factor $\eta_q = 1/|S_q|$ where S_q are the training instances for query q.
2. For $t = 0, \ldots, T$:

   For each training sample $(\mathbf{d}_{nq}, \mathbf{d}_{rq}) \in S$:
   If $Score(\mathbf{d}_{nq}, \mathbf{w}^i) \geq Score(\mathbf{d}_{rq}, \mathbf{w}^i)$ then
   update: $\mathbf{w}^{i+1} = \mathbf{w}^i + \eta_q(\mathbf{d}_{rq} - \mathbf{d}_{nq})$
   $i = i + 1$
   Else update: $c_i = c_i + 1$
3. Output (for several variants of the algorithm):
   Last: $\mathbf{w}^i$
   Pocket: $\mathbf{w}^l$ where $l = \arg\max_i c_i$
   Average: $\frac{1}{Z}\sum_i c_i \mathbf{w}_i$ where $Z = \sum_i c_i$

judgments. Note also that the perceptron algorithm typically uses a global learning rate η for the weight updates. We do away with this learning rate in our implementation, and instead balance the weight updates across queries with a per-query balancing factor η_q. This is in the same spirit as a technique presented in [10] for balancing RankSVM training across queries.

Since each weight update in the perceptron simply adds (or subtracts) the document vector to or from the current weight, we can express the weight vector learned up to a given time as follows:

$$\mathbf{w}^i = \mathbf{w}^0 + \sum_{\mathbf{d}_{iq} \in R \cup N} \eta_q \alpha_{iq} \mathbf{d}_{iq}$$

where $R \cup N$ are the union of relevant and non-relevant documents to this query, $|\alpha_{iq}|$ is the number of times a document was mis-ranked during the training, and is positive (negative) for relevant (non-relevant) documents. Note that α_{iq} can be zero if a document is never mis-ranked during training. Our document scoring function given in equation 1 can be re-parameterized in terms of the α's and written as follows (assuming the initial weight vector $\mathbf{w}^0$ is zero):

$$Score(\mathbf{d}_{jq'}, \boldsymbol{\alpha}) = \sum_{\mathbf{d}_{iq} \in R \cup N} \eta_q \alpha_{iq} < \mathbf{d}_{jq'}, \mathbf{d}_{iq} > \qquad (2)$$

where $R \cup N$ are all the relevant and non-relevant documents in our training set. This dual form of the learned perceptron weights is often used to apply kernel methods to the perceptron learner, substituting the dot product, $< \cdot, \cdot >$ with a suitable kernel, such as the polynomial or Gaussian kernel.

Although the perceptron is elegant and simple, with provable convergence properties and mistake bounds, it can be quite unstable when applied to data that is not linearly separable. Many variants of the perceptron algorithm have been proposed to alleviate this problem, ranging from attempts to maximize the classification margin with SVM-like constraints [13, 25], to voting or averaging schemes[17], to down-weighting "noisy" samples in the training set [23].

Voting and averaging of the learned hypotheses has led to improvements in the perceptron's performance and stability. The original perceptron algorithm simply returned the last learned weights vector. This approach, however, could lead to learning poor hypotheses because the weight vector may have been recently updated with noisy examples and only tested against very few training instances. Gallant proposed the "longest survivor" or "pocket perceptron" to get around this problem[18]. In this algorithm, the weight vector is returned that correctly classified the most training instances in a row. Freund and Schapire proposed the "voted perceptron" which returns all the learned weight vectors and composed a final classifier by letting each intermediate learner vote on new instances, weighting the votes by the number of correct classifications made during training, c_i [17]. This approach can be expensive, and an approximation to voting is often used by averaging all the weight vectors[11]. In [24], a "selective voting" approach was taken which only used a subset of the learned weights, choosing a threshold B and setting $c_i = 0$ for $i < B$.

Another method of making the perceptron more stable when applied to noisy non-linearly separable data is to "clean" the data while training [19, 23]. One of these techniques, referred to as the *α-bound*, attempts to minimize the impact of a single training observation on the final learned hypothesis. While proceeding through a training iteration, if an instance is mis-classified more than a pre-determined number of times, that instance is removed from training set for future iterations. This in effect puts a bound on the α weights in the dual form of the perceptron, equation 2. The bound on α can be specified as a fraction of the number of iterations through the data.

5. COMMITTEE PERCEPTRON

In this section we describe the *committee perceptron* algorithm in detail. Because this algorithm is a straightforward extension to the average- and pocket-perceptron algorithms, we expect the same theoretical properties of those algorithms apply to the committee perceptron as well. In extending those algorithms, the committee perceptron allows optimization across a full range of committee sizes instead of just the extremes.

Most basically, the committee perceptron votes or averages the n-best learned hypotheses instead of all of the hypotheses. This simple modification would not initially seem to provide a significant advantage over the average perceptron. But, as will be shown later, the committee perceptron approaches a better solution faster than other variants, and the "best" hypotheses selected for voting can be biased towards an arbitrary retrieval performance metric. Table 2 presents

Table 2: Committee Perceptron Algorithm for Document Ranking Function Learning.

Input: Number of iterations T, Committee Size N_{com}, List of training document pairs $S = R \times N = \{(\mathbf{d}_{nq}, \mathbf{d}_{rq})\}$ where d_n is non-relevant and d_r is relevant to query q
Output: Set of Parameter Vectors and their success counters $K = \{(\mathbf{w}^k, c_k) | k = 1 \dots N_{com}\}$

1. Initialize $i = 0$, success counter $c_i = 0$, initial parameters $\mathbf{w}^0$, committee $K = \emptyset$, query balancing factor $\eta_q = 1/|S_q|$ where S_q are the training instances for query q.
2. For $t = 0, \dots, T$:

   For each training sample $(\mathbf{d}_{nq}, \mathbf{d}_{rq}) \in S$:
   If $Score(\mathbf{d}_{nq}, \mathbf{w}^i) \geq Score(\mathbf{d}_{rq}, \mathbf{w}^i)$ then
   $(\mathbf{w}^{\min}, c_{\min}) \in K$ s.t. $c_{\min} = \min_k c_k \in K$
   If $c_i > c_{\min}$ then
   add $(\mathbf{w}^i, c_i)$ to K
   while $|K| > N_{sub}$
   remove $(\mathbf{w}^{\min}, c_{\min})$ from K
   update: $\mathbf{w}^{i+1} = \mathbf{w}^i + \eta_q(\mathbf{d}_{rq} - \mathbf{d}_{nq})$
   $i = i + 1$
   Else update: $c_i = c_i + 1$
3. Output: K

the committee perceptron algorithm.

The committee perceptron proceeds like the traditional perceptron algorithm, iterating through the training instances and updating the hypothesis weight vector $\mathbf{w}^i$ after each ranking mistake. Before the current hypothesis is updated, it is considered for induction into the committee. The hypothesis is added to the committee if its success counter c_i is greater than the minimum success counter of the hypotheses already in the committee or if the committee has not reached its maximum size yet, N_{com}. If the committee size exceeds N_{com} when a new hypothesis is added to the committee, the hypothesis with the minimum success counter in the committee is retired in order to keep the committee size fixed.

In the voted- or pocket-perceptron algorithms, performance of a hypothesis is judged only by the number of examples correctly ranked in a row, i.e. its success counter c_i. But, by maintaining a committee of hypotheses, we can form a more accurate and potentially more task-appropriate estimate of the performance of those hypotheses. At the end of training, each hypothesis can be evaluated against a hold-out development set using a user-specified performance metric. Hypotheses are then combined in some weighted scheme using their performance as the weights. This has the effect of favoring hypotheses that perform better with regard to this performance metric.

One final advantage of the committee perceptron over the average perceptron is that it enables us to use true voting schemes rather than just averaging the hypotheses. When classifying new examples with the voted perceptron algorithm, each intermediate hypothesis must score the example and sort the list of document to produce its vote. With large amounts of training data, data that is far from being linearly separable, or a high number of iterations through the data, the number of generated hypotheses can be quite large. This situation can make voting impractical in general. But, when limiting the number of hypotheses that can place a vote by using a committee, voting is a much more practical strategy. In the context of pairwise preference learning of ranking functions, (weighted) majority voting corresponds to Borda-counting or BordaFuse [3]. In this work, we apply hypothesis averaging and BordaFuse to produce the final system output, weighting each hypothesis by a function of its performance on the validation set.

Note that this algorithm is a direct generalization of the previously presented perceptron variants. Setting $N_{com} = 1$ corresponds to the pocket-perceptron and setting $N_{com} = \infty$ corresponds to the average-perceptron algorithms, when using an hypothesis averaging combination strategy.

Table 3: Collection sizes in the LETOR dataset, showing the average number of judged document per query and the percentage of relevant documents in that set at each relevance levels.

Collection	Features	Queries	Docs/Q	%Rel
OHSUMED	25	106	152.24	16% / 14%
TREC 2003	44	50	983.42	1.0%
TREC 2004	44	75	988.93	0.6%

6. EXPERIMENTAL RESULTS

In this section we will describe experiments conducted with the committee perceptron, and compare its performance to two state-of-the-art rank learning algorithms, RankSVM [10, 22] and RankBoost [16].

6.1 Datasets

The experiments presented here were all conducted using the recently released Learning to Rank (LETOR) Benchmark dataset [26]. This dataset attempts to provide a standard set of document-query features over several test collections for use in learning document ranking functions. These features were extracted from all the query-document pairs in the OHSUMED collection and the .GOV test collection using the queries and judgments from the TREC 2003 and 2004 web track topic distillation tasks[15, 14]. The relevance judgments in the TREC collections are binary and in the OHSUMED collection are graded in three levels: "definitely relevant", "possibly relevant" and "not relevant". The LETOR dataset also contains standardized train/validation/test splits for 5-fold cross validation. The sizes of each collection in the datset, as well as the percent relevant documents for each relevance level, are shown in table 3.

Some of the document-query features provided in this collection are query term frequency in title or abstract (for OHSUMED), BM25 score, and $P(query|document)$ with various smoothing parameters from the language modeling

retrieval model [33]. The reader is referred to [26] for a detailed explanation of the feature sets. In our experiments, the query-docuent feature values were normalized on a per-query basis to the $[0, 1]$ interval using the linear scaling suggested by the producers of the LETOR dataset and no additional feature selection or processing was done.

6.2 Algorithm Performance

We evaluate the performance of our algorithm across various parameter settings and against two state-of-the-art baseline algorithms that have been widely used in document ranking function learning: RankSVM[22] and RankBoost [12]. Throughout these experiments we evaluate against the validation set using NDCG@10 as a representative high-precision performance metric. The algorithm described above does not make any assumptions about the performance metric to be optimized, and any high-precision performance metric could be substituted in its place. The validation set was used to select the number of iterations to run (T) and used to set the weights for the committee member combination strategies: weighted averaging and weighted Borda count.

Unless otherwise noted, all test results reported are 5-fold cross validation results, with the number of iterations T chosen to maximize NDCG@10 on the validation set. All weights in the perceptron algorithms are initialized to zeros. Test results varying the committee and subcommittee sizes up to 100 members did not show significant differences in performance, and settings in this range are used in all of the tests described here. Additionally, we employed an α-*bound* of 0.85 for all perceptron variant tests as described above, removing instances that were misclassified in more than 85% of the iterations through the data. As noted in [19], this technique improved the stability of the algorithm over all of the perceptron variants. The exact values of the committee sizes and α-*bound* was chosen from preliminary experimentation, and we leave selecting these parameters in a more principled way as future work.

Looking at the learning rates of several perceptron variants, some trends are clear. The results in figure 1 show that the committee perceptron approaches a stable solution in many fewer iterations than both the pocket- and average-perceptron algorithms. This solution also consistently outperforms the other perceptron variants. Throughout our experimentation, the hypothesis learned from the committee perceptron reached its maximum performance on the validation set before roughly 50 iterations through the training data, whereas the average perceptron continued to improve even after 500 iterations.

Table 4 shows the performance of the committee perceptron and other perceptron variants as compared to two state-of-the-art rank learning algorithms, RankSVM [10, 22] and RankBoost [16]. In all tests, the committee perceptron performs comparably to these state of the art algorithms, and significantly better than the average- and pocket-perceptron variants. Statistically significant performance improvements over both baseline algorithms were seen in some of the OHSUMED and TD2003 dataset tests. In all cases where the baseline algorithms outperformed the committee perceptron, that difference is not statistically significant. All statistical signifi-

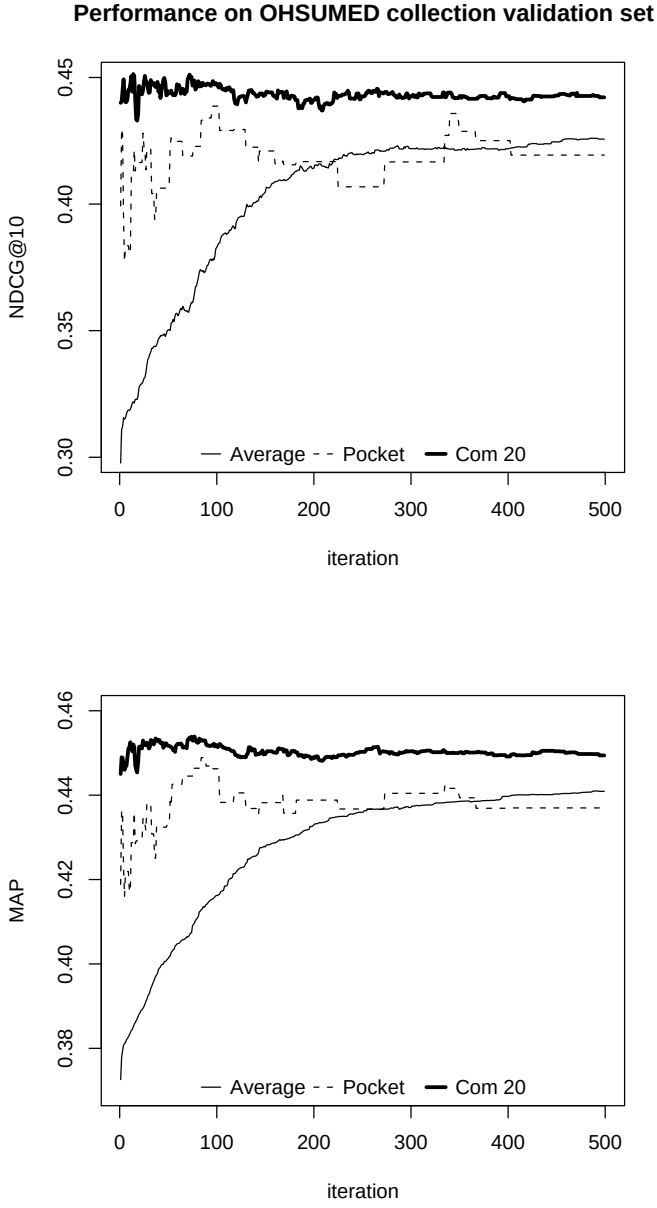

Figure 1: NDCG@10 and MAP on the OHSUMED collection validation set across iterations for several perceptron variants. *Average* = average perceptron, *Pocket* = pocket perceptron, *Com 20* = committee perceptron with committee size of 20.

Table 5: Training Time of the Committee Perceptron algorithm (50 iterations, $N_{com} = 20$) compared to RankSVM. All times are in seconds.

Fold	Committee Perceptron	RankSVM
1	462.85	27487.40
2	527.14	35887.08
3	489.62	28584.22
4	414.18	7328.77
5	375.01	6902.77
Average	453.76	21238.05

cance is at the $p < 0.05$ level with a one-tailed paired t-test.

As observed by Liu et. al. [26], the performance of the baseline algorithms on the TD2004 dataset raises some interesting questions. The TD2003 and TD2004 datasets are built from the same document collection using the same features, only the queries are different. There are more queries in the TD2004 collection (75) than the TD2003 collection (50), and the increase in training set size likely accounts for the increased performance across all algorithms between the two TD collections. The magnitude of the increase, however, is dramatically different across the algorithms. Both RankSVM and the committee perceptron realize less than a 20% performance improvement across the TD collections, but the RankBoost performance gain is more than 70%. There are several possibilities for this discrepancy: the difference between the linear ranking functions of RankSVM and the perceptron variants vs. the non-linear nature of RankBoost's final ranking function, the implicit feature selection performed by RankBoost, or a combination of these factors. Additionally, Liu et. al. [26] suggest that this performance difference may be indicative of inherent instability in RankBoost. We leave further investigation of this phenomenon for future work.

6.3 Running Time Analysis

We evaluated our algorithm's running time against the popular SVM implementation, SVM^{light} [1][21], which supports learning ranking functions from pairwise preference data. These tests were performed on the OHSUMED corpus, and the results in table 5 show training time for each training fold. These results show that our algorithm is more than 45 times faster to train than RankSVM on this dataset. Note that this could be considered a worst-case comparison: our committee perceptron algorithm implementation is in Java[2], whereas SVM^{light} is implemented in C.

7. CONCLUSION & FUTURE WORK

In this paper we present a new variant of the perceptron algorithm, the committee perceptron, and applied it to the problem of learning document ranking functions. This algorithm is a generalization of the pocket- and average-perceptron algorithms, allowing optimization across the full operational range of committee sizes rather than just the extremes. By maintaining a set of n-best hypotheses, the committee perceptron algorithm is also capable of being tuned to favor hypotheses that perform better on an arbitrary retrieval performance metrics. We applied this algorithm to learning of document ranking functions for ad-hoc retrieval and compared against two baseline algorithms applied to this problem, RankSVM and RankBoost. The committee perceptron algorithm performs comparably to or better than two state-of-the-art algorithms on three large public domain corpora, and also is capable of being trained in a fraction of the time – more than a 45-fold decrease in training time as compared to RankSVM. Tests show that the committee perceptron algorithm more quickly approaches a better hypothesis than other variants of the perceptron algorithm such as the pocket- and average-perceptron.

There are several refinements remaining in the development of this algorithm and applying it to the problem of learning ranking functions. The performance on the TD2004 collection shows that there is still room for significant improvement in the algorithm presented here. Future work includes investigating why RankBoost so dramatically outperforms the others on that collection, and if there are any features of RankBoost that can be integrated into the committee perceptron.

[1] http://svmlight.joachims.org/

[2] http://java.sun.com/

8. ACKNOWLEDGMENTS

We would like to thank the producers of the LETOR dataset for generously making that data publicly available. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract Numbers NBCHD030010 and IBM W0550432 (DARPA PRIME Agreement # HR0011-06-2-0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), or the Department of Interior-National Business Center (DOI-NBC).

9. REFERENCES

[1] E. Agichtein, E. Brill, S. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10, New York, NY, USA, 2006. ACM Press.

[2] P. Ahlgren and L. Grönqvist. Retrieval evaluation with incomplete relevance data: a comparative study of three measures. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 872–873, New York, NY, USA, 2006. ACM Press.

[3] J. A. Aslam and M. Montague. Models for metasearch. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 276–284, New York, NY, USA, 2001. ACM Press.

[4] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW7: Proceedings of the seventh international conference on World Wide Web 7*, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science

Table 4: Performance across all test sets. *Com 30* is the committee perceptron with $N_{com} = 30$. "A" indicates weighted averaging of the hypotheses, and "B" indicates weighted Borda count (or BordaFuse). $*$ and † indicate statistical significance at the 0.05 level using a one-tailed paired t-test over RankBoost and RankSVM respectively. No tests were significant over the committee perceptron.

		NDCG@		
	MAP	1	5	10
OHSUMED				
RankSVM	0.447*	0.495	0.458	0.441
RankBoost	0.440	0.498	0.450	0.436
Com 30 A	0.447*	0.539	**0.464**	**0.450**
Com 30 B	**0.449***	**0.558†***	**0.464**	0.447
Pocket	0.440	0.510	0.426	0.418
Average	0.443	0.508	0.445	0.429
TREC 2003				
RankSVM	**0.256***	0.420*	0.347*	0.341*
RankBoost	0.212	0.260	0.279	0.285
Com 30 A	0.247*	**0.440***	0.367*	**0.359***
Com 30 B	0.239	0.400*	**0.371***	0.351*
Pocket	0.208	0.360	0.303	0.308
Average	0.143	0.200	0.173	0.191
TREC 2004				
RankSVM	0.350	0.440	0.393	0.420
RankBoost	**0.384†**	**0.48**	**0.437**	**0.472†**
Com 30 A	0.358	0.440	0.396	0.432
Com 30 B	0.358	0.440	0.403	0.434
Pocket	0.356	0.440	0.397	0.437
Average	0.327	0.413	0.366	0.400

Publishers B. V.

[6] C. Buckley and E. M. Voorhees. Retrieval evaluation with incomplete information. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 25–32, New York, NY, USA, 2004. ACM Press.

[7] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 89–96, New York, NY, USA, 2005. ACM Press.

[8] S. Büttcher, C. L. A. Clarke, and I. Soboroff. The trec 2006 terabyte track. In *15th Text REtrieval Conference (TREC 2006)*, November 2006.

[9] S. Büttcher, C. L. A. Clarke, P. C. K. Yeung, and I. Soboroff. Reliable information retrieval evaluation with incomplete and biased judgements. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 63–70, New York, NY, USA, 2007. ACM Press.

[10] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193, New York, NY, USA, 2006. ACM Press.

[11] V. R. Carvalho and W. W. Cohen. Single-pass online learning: performance, voting schemes and online feature selection. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 548–553, New York, NY, USA, 2006. ACM Press.

[12] M. Collins. Ranking algorithms for named-entity extraction: boosting and the voted perceptron. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 489–496, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[13] K. Crammer and Y. Singer. A family of additive online algorithms for category ranking. *J. Mach. Learn. Res.*, 3:1025–1058, 2003.

[14] N. Craswell and D. Hawking. Overview of the trec 2004 web track. In *13th Text REtrieval Conference (TREC 2004)*, November 2004.

[15] N. Craswell, D. Hawking, R. Wilkinson, and M. Wu. Overview of the trec 2003 web track. In *12th Text REtrieval Conference (TREC 2003)*, November 2003.

[16] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

[17] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3):277–296, 1999.

[18] S. I. Gallant. Perceptron-based learning algorithms. *Neural Networks, IEEE Transactions on*, 1(2):179–191, 1990.

[19] J. Gao, H. Qi, X. Xia, and J.-Y. Nie. Linear discriminant model for information retrieval. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 290–297, New York,

NY, USA, 2005. ACM Press.

[20] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[21] T. Joachims. Making large-scale support vector machine learning practical. *Advances in kernel methods: support vector learning*, pages 169–184, 1999.

[22] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM Press.

[23] R. Khardon and G. M. Wachman. Noise tolerant variants of the perceptron algorithm. Technical report, Tufts University, Medford, MA. TR-2005-8, November 2005.

[24] Y. Li. Selective voting for perceptron-like online learning. In *Proc. 17th International Conf. on Machine Learning*, pages 559–566. Morgan Kaufmann, San Francisco, CA, 2000.

[25] Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46(1–3):361–387, 2002.

[26] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *SIGIR '07: Proceedings of the Learning to Rank workshop in the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007.

[27] I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong. High accuracy retrieval with multiple nested ranker. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 437–444, New York, NY, USA, 2006. ACM Press.

[28] Metzler, Donald, B. Croft, and W. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, June 2007.

[29] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Neurocomputing: foundations of research*, pages 89–114, 1988.

[30] L. Shen and A. K. Joshi. Ranking and reranking with perceptron. *Mach. Learn.*, 60(1-3):73–96, 2005.

[31] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges. Optimisation methods for ranking functions with multiple parameters. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 585–593, New York, NY, USA, 2006. ACM Press.

[32] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278, New York, NY, USA, 2007. ACM Press.

[33] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.

APPENDIX

Let r_{rel} be a "perfect" ranking where all relevant documents are ahead of all non relevant documents and let r_{sys} be a system produced ranking. Define a class of ranking performance metrics as

$$\Phi_{Z,m}(r_{sys}, r_{rel}) = \frac{1}{Z}\sum_{i=1}^{m}\frac{i}{r_i}$$

where Z is a measure-dependent normalization, m indicates how far down the ranked list to look, and r_i is the rank of the ith relevant document in the system produced ranking. This class of ranking performance includes many commonly used measures retrieval, and their definitions in terms of Φ are given in table 6.

Table 6: Some common retrieval performance metrics expressed in terms of Z and m, where $\Phi_{Z,m} = \frac{1}{Z}\sum_{i=1}^{m}\frac{i}{r_i}$

$\Phi_{Z,m}$	Z	m
Average Precision	R	R
Precision at K	K	m_k s.t. $r_{m_k} \leq K < r_{m_{k+1}}$
Reciprocal Rank	1	1
R-Precision	R	m_r s.t. $r_{m_r} \leq R < r_{m_{r+1}}$

We will show that this class of performance metrics is bounded from below when minimizing the number of inversions in a ranked list. The following theorem and proof are a generalization of the bound on average precision presented in [22], and the proof here closely follows that proof.

THEOREM 1. *let r_{rel}, r_{sys} and $\Phi_{Z,m}(r_{sys}, r_{rel})$ be as defined above. If R is the number of relevant documents, Q is the number of discordant document pairs or inversions between r_{rel} and r_{sys}, then our performance metric is bounded from below by:*

$$\Phi_{Z,m}(r_{sys}, r_{rel}) \geq \frac{1}{Z}\left[Q + \binom{R+1}{2}\right]^{-1}\left(\sum_{i=1}^{m}\sqrt{i}\right)^2$$

PROOF. In a perfect ranking, the sum of relevant ranks is:

$$\sum_{i=1}^{R} r_i = \binom{R+1}{2}$$

. For each inversion of a relevant and non-relevant document in this ranking, we reduce the rank (increase r_i) by one thereby adding one to this sum. So, in general we can express this sum as

$$\sum_{i=1}^{R} r_i = Q + \binom{R+1}{2} \tag{3}$$

. Similarly, we can focus on the top $m \leq R$ relevant documents in that ranking and let Q_m be the number of inversions involving only those documents.

$$\sum_{i=1}^{m} r_i = Q_m + \binom{m+1}{2} \leq \sum_{i=1}^{R} r_i \tag{4}$$

Given this relationship, we can take Q_m as fixed and express the lowest value of our performance metric $\Phi_{Z,m}$ as the following integer optimization problem:

minimize:

$$\Phi_{Z,m}(r_{sys}, r_{rel}) = \frac{1}{Z} \sum_{i=1}^{m} \frac{i}{r_i} \tag{5}$$

subject to:

$$\sum_{i=1}^{m} r_i = Q_m + \binom{m+1}{2} \tag{6}$$

$$1 \leq r_1 < \ldots < r_m \tag{7}$$

$$r_1, \ldots, r_m \text{ integer} \tag{8}$$

Removing the last two constraints in this optimization problem only lowers the minimum value, and therefore still presents a solution to that lower bound. Without those constraints, this becomes a convex optimization problem and can be solved with Lagrange multipliers. The Lagrangian is:

$$L(r_1, \ldots, r_m, \beta) = \frac{1}{Z} \sum_{i=1}^{m} \frac{i}{r_i} + \beta \left[\sum_{i=1}^{m} r_i - Q_m - \binom{m+1}{2} \right] \tag{9}$$

Differentiating with respect to the r_i gives the following:

$$\frac{\delta L(r_1, \ldots, r_m, \beta)}{\delta r_i} = -iZ^{-1} r_i^{-2} + \beta \tag{10}$$

Setting 10 equal to zero, solving for r_i and substituting back into 9 gives:

$$L(r_1, \ldots, r_m, \beta) = 2\sqrt{\frac{\beta}{Z}} \sum_{i=1}^{m} \sqrt{i} - \beta \left[Q_m + \binom{m+1}{2} \right] \tag{11}$$

Taking the derivative with respect to β gives:

$$\frac{\delta L(r_1, \ldots, r_m, \beta)}{\delta \beta} = \sqrt{\frac{1}{\beta Z}} \sum_{i=1}^{m} \sqrt{i} - \left[Q_m + \binom{m+1}{2} \right] \tag{12}$$

Setting 12 equal to zero, solving for β and substituting back into 11 gives the following solution:

$$\Phi_{Z,m}(r_{sys}, r_{rel}) \geq \frac{1}{Z} \left[Q_m + \binom{m+q}{2} \right]^{-1} \left(\sum_{i=1}^{m} \sqrt{i} \right)^2 \tag{13}$$

By equations 3, 4 and 13, we have proven the theorem. □