# A Meta-Learning Approach for Robust Rank Learning

Vitor R. Carvalho, Jonathan L. Elsas, William W. Cohen and Jaime G. Carbonell
Language Technologies Institute
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA
{vitor,jelsas,wcohen,jgc}@cs.cmu.edu

## ABSTRACT

Learning effective feature-based ranking functions is a fundamental task for search engines, and has recently become an active area of research [10, 3, 2]. Many of these recent algorithms are based on the pairwise preference framework, in which instead of taking documents in isolation, document pairs are used as instances in the learning process. One disadvantage of this process is that a noisy relevance judgment on a single document can lead to a large number of mis-labeled document pairs. This can jeopardize robustness and deteriorate overall ranking performance. In this paper we study the effects of outlying pairs in rank learning with pairwise preferences and introduce a new meta-learning algorithm capable of suppressing these undesirable effects. This algorithm works as a second optimization step in which any linear baseline ranker can be used as input. Experiments on eight different ranking datasets show that this optimization step produces statistically significant performance gains over various state-of-the-art baseline rankers.

## Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models; H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness)

## General Terms

Algorithms, Design, Experimentation

## Keywords

Learning to rank, Empirical Risk

## 1. INTRODUCTION

The "Learning to Rank" problem has gained much attention from the information retrieval research and industry communities recently. This is the problem of using queries previously submitted to a search engine and relevance information on the retrieved results to improve performance of a document ranking algorithm. One goal of this machine learning task is to automatically adjust the parameters in the ranking algorithm to return more relevant documents higher in the results list for future queries.

Much of the recent research in rank learning has focused on document ranking, but Learning to Rank (LETOR) is a widely applicable machine learning task. Ranking (or re-ranking) has long been important in machine translation [20], named-entity extraction [7] and expert finding [6] to name just a few areas. In all of these tasks, objects (named entity labels, target-language translations, candidate experts) are ranked in response to some "query" (text span, source-language sentences, keyword query).

One popular approach to learning ranking functions is to learn a *preference function* over pairs of documents given a query. This preference function indicates to which degree one document is expected to be more relevant than another with respect to the query. When these preference functions are transitive, as is typically the case, the document collection can be ranked in descending order of preference.

There is evidence that assessment of pairwise preferences is easier for assessors and yields higher inter-annotator agreement [5]. There are also many practical advantages in adopting a pairwise preference approach for automatic learning of feature-based ranking functions. First, most classification methods can be easily adapted to this formulation of the ranking problem. Second, this framework can be generalized to any graded relevance levels (e.g. definitely relevant, somewhat relevant, non-relevant). Third, in many scenarios it is easier to obtain large amounts of pairwise preference data [14].

Using pairwise preferences, however, does pose some risks. In the presence of labeling errors or other "noise" in the document relevance information, creating a training set by pairing documents causes a quadratic increase in the number of noisy outlier observations. As we will show, this can have a strong negative impact on the quality of the learned ranking function.

In this paper, we propose a two-stage optimization strategy for learning ranking functions that is robust to outliers and applicable to any method that learns a linear ranking function. This meta-ranker is computationally economical and, although developed for document ranking, this method generalizes across many ranking tasks. Experimental results on eight different ranking collections show consistent and significant improvements over a range of baseline ranking functions.

*SIGIR 2008 LR4IR - Workshop on Learning to Rank for Information Retrieval* Singapore

# 2. LEARNING TO RANK

## 2.1 Related Work

Learning feature-based ranking functions has become an active area of research, with the recent introduction of several new ranking algorithms. RankNet [2] learns a ranking function by minimizing the number of mis-ranked pairs via gradient descent on a probabilistic loss function. RankBoost [11] and AdaRank [25] are examples of algorithms that use the boosting framework to learn feature-based ranking functions. Various methods based on the Perceptron algorithm have also been proposed to learn ranking functions for applications such as information retrieval [12, 10], reranking for named-entity extraction [7] and machine translation [20].

One limitation of these algorithms is that they do not directly maximize information retrieval performance metrics, such as NDCG (Normalized Discounted Cumulative Gain) [13] or MAP (Mean Average Precision) [1]. Instead, different approximations for these metrics have been used as proxies in the learning procedure. Matveeva et al. [17] and Taylor et al. [22] present two distinct ways to tailor RankNet to optimize NDCG. 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, Yue et al. [26] presented a method for adapting RankSVM to maximize an approximation of average precision. More recently, Taylor et al.[21] described SoftRank, a method that optimizes "softNDCG", an approximation of the NDCG metric.

Other common approaches to optimizing specific retrieval performance metrics include grid-search, coordinate ascent and line-search [18]. These methods perform heuristic exploration of ranking function parameters (the hypothesis space), evaluating sampled hypotheses against the target performance measure. Although these methods directly optimize any given performance metric on the training set, they are generally very expensive when applied to ranking functions with more than just a few parameters.

## 2.2 Pairwise-preference Ranking

Many of the recently proposed approaches to learn feature-based ranking functions take a pairwise preference approach [2, 3, 14, 10, 26]. In the pairwise framework, instead of taking documents in isolation, document pairs are taken as instances in the learning process. The goal in this setting is to learn a *preference function* over document pairs, where the output of the learned function indicates the degree to which one document is preferred over another for a given query.

This approach is appealing for several reasons. First, learning a preference function on pairs of documents reduces the ranking problem to a binary classification problem: a correct (or incorrect) classification corresponds to correctly (or incorrectly) ordering a document pair. Many classification algorithms have been adapted to this task, including support vector machines [14], perceptron algorithms [10, 12] as well as gradient descent algorithms [2].

Second, this approach imposes very few assumptions on the structure of the training data — only that preferences among documents are somehow expressed. Explicit document preferences assessment [5] can clearly be used with this learning approach. Additionally, traditional absolute relevance judgements (binary or graded relevance) can be easily converted to a pairwise preference training set by taking all pairs of document with differing relevance levels. Clickthrough data has also been used by assuming that a clicked-on document expresses a preference for that document over documents occurring higher in the document ranking [14].

We represent our learning setting as follow: a ranking dataset consists of a set of queries $q \in Q$, and a set of documents for each query $d_i \in D_q$ with some associated relevance judgement of document $d$ for query $q$, $y_{qi}$. Our training set for a single query is then $S_q = \{(d_{q1}, y_{q1}), (d_{q2}, y_{q2}), ...\}$. The relevance judgements $y$ are discrete and ordered, with values such as $\{$*Probably Relevant*, *Possibly Relevant*, *Not Relevant*$\}$, and a total ordering $\triangleright$ exists between relevance levels, e.g. *Probably Relevant* $\triangleright$ *Possibly Relevant* $\triangleright$ *Not Relevant.*

Documents are represented by a vector of query-dependent feature weights $f_k$. For instance, document $d_i$ given query $q$ is represented as:

$$d_{qi} = [f_0(d_i, q), f_1(d_i, q), ..., f_m(d_i, q)] \quad (1)$$

where each feature scoring functions $f_k$ represents some measure of similarity between the document and query. These can be derived from low-level features typically used in information retrieval systems (such as query term frequency or inverse document frequency), higher-level features such as the score assigned by a baseline ranking algorithm for document $d_i$ on query $q$ (such as BM25), or even query independent document quality measurements (such as PageRank).

The goal of the learning procedure in the pairwise framework is to induce a document score function $s(\bullet)$ such that

$$y_{qi} \triangleright y_{qj} \iff s(d_{qi}) > s(d_{qj}) \quad (2)$$

i.e., whenever document $d_{qi}$ is preferred over ($\triangleright$) $d_{qj}$, the scoring function $s$ will return a larger value for $d_{qi}$ than for $d_{qj}$. This formulation makes clear the connection between pairwise-preference learning and binary classification: given the preference relationship on the left in Equation 2 a correct (or incorrect) classification corresponds to maintaining (or violating) the inequality on the right.

It is often useful to explicitly model this task as binary classification. In this view, one can build a new "paired" dataset $S'$ for each query $q$ by creating document pairs from documents with different relevant levels $(d_{qi}, d_{qj})_l$ and associated preference labels $z_l$. That is, $S'_q = \{((d_{qi}, d_{qj})_l, z_{ql}) \,|y_{qi} \neq y_{qj}, \forall i, j\}$ and

$$z_{ql} = \begin{cases} +1 & \text{if } y_{qi} \triangleright y_{qj}; \\ -1 & \text{if } y_{qi} \triangleleft y_{qj}. \end{cases}$$

Analogously, we can write Equation 2 with this notation, letting $P_l$ be the *pairwise score* of document pair $(d_{qi}, d_{qj})_l$

$$P_l = z_{ql} \times (s(d_{iq}) - s(d_{jq})) > 0. \quad (3)$$

Minimizing the number of *misranks*, i.e. incorrectly ordered document pairs, is in principle a good criterion for rank optimization. It has been shown that minimizing the number of misranks is equivalent to maximizing a lower-bound on various information retrieval performance metrics, such as average precision and reciprocal rank [10]. However, the direct optimization of the number of misranks is an NP-hard problem [14], so approximations are necessary. We will show below several approaches to approximating this minimization.

In this paper we are concerned with linear score functions $s(\bullet)$ that can be parameterized by a single weight vector $w = [w_1, w_2, ..., w_m]$. Thus the learning algorithms output scoring functions can be expressed as $s(d_{qi}) = \langle d_{qi}, w\rangle$, where $\langle \bullet, \bullet\rangle$ is the inner product operation. After learning this score function, the final document rankings can be derived from this function by ranking in descending order according to their score[1].

## 2.3 Outliers in Pairwise Preference Ranking

Creating pairwise training data from absolute judgements may have some undesirable consequences. Specifically, mislabeling of a single document's absolute judgement will lead to many "mis-labeled" document pair preferences. When using graded relevance levels, confusion or inconsistencies between different relevance levels may make mis-labeling a common problem. If we consider each mis-labeled document a noisy observation or outlier, the process of pairing each document with all others of differing relevance levels yields a quadratic increase in the number of outliers in the training data. This increase can have a serious detrimental effect on performance, as we will show below.

Mis-labeling the absolute relevance level of a document is not the only source of outliers. Due to the nature of keyword search, we have an extremely impoverished view of the information need — typically only 2-3 terms per query. For this reason, the query-document features ($f_i$ in Equation 1) may not be expressive enough to truly distinguish relevant from non-relevant documents. This may result in many non-relevant documents "looking similar" to relevant in the query-document feature space. These non-relevant documents can also be considered outliers, and similar to misjudged documents, yield a quadratic increase in the number of pairwise outliers.

To illustrate the effect of outliers on rank learning, we trained a RankSVM model (see Section 3.1) on SEAL-1, a subset of the Set Expansion ranking dataset described in Section 4.1. Given the model learned $w$, we calculated the pairwise decision scores $P_l$ (Equation 3) for all training data instances and constructed a histogram, as shown in the top of Figure 1. Most pairwise instances had positive scores $P_l$ (top right in the figure), showing that the learned ranking model correctly ordered most of the training instances. Some instances, however, had negative scores and the few having the most negative scores may be outliers (top left, Figure 1).

In order to measure the previously mentioned outlier effect, we then retrained our model on a smaller training set after removing increasing numbers of outlier instances. That is, we trained the same RankSVM model excluding from the training data a few instances whose scores were below a cutoff value, $P_l'$, and then evaluated the learned model on the same test set. The bottom of Figure 1 shows test MAP results when training is performed excluding outliers with pairwise score below a threshold from the training data. In this figure, the dashed horizontal line shows performance when all instances are used for training. The leftmost point shows the performance when instances with score below $-15$ were removed from training. As the removal cuttoff increases up to $-4$, performance goes up, indicating that the removal of outliers improves the ranker's performance. For larger cutoffs, this effect is curtailed by the larger numbers of instances being discarded and performance drops.

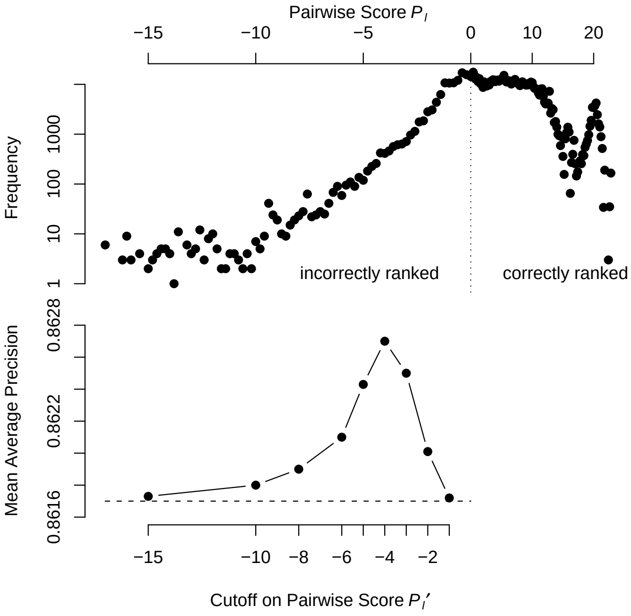

**Figure 1: Example of outliers in pairwise Ranking. (top) Histogram of pairwise scores. (bottom) Mean Average Precision on the same test collection when excluding training instances whose scores were below cutoff.**

Empirical evidence from several studies also suggests that performance of pairwise learning algorithms can be improved by removing or down-weighting these outliers. In perceptron-based learning algorithms, outliers were identified as document pairs that were consistently mis-ranked in several iterations through the training data, and removal of these document-pairs improved the performance and stability of the learned ranking function [10, 12]. This technique, known as the $\alpha$-bound [15], limits the influence of potential outlier observations on the final learned hypothesis.

Although the $\alpha$-bound is reported to work well with perceptron-based learners, it is unclear how it generalizes to other learning algorithms. In this work we develop a general mechanism to down-weight the influence of these outliers in pairwise preference learning and apply this technique to a variety of learning algorithms. Results show significantly improved performance of learned ranking functions across a variety of ranking tasks.

# 3. ROBUST PAIRWISE RANKING

In this section we propose a new learning algorithm to counteract the effect of outliers in pairwise rank learning. The algorithm takes as input the linear model learned by any linear ranker (a base model), and uses a non-convex optimization procedure to output a more robust and effective final linear ranking model. To help explain the algorithm, we start with a brief explanation of RankSVM.

[1] Notice that the score function $s(\bullet)$ takes a single document as argument, and not the document pair.

## 3.1 RankSVM

One of the most successful algorithms for classification, Support Vector Machines (SVM), has recently been successfully adapted to ranking using the pairwise framework [14]. Given a binary paired dataset $S'$ from a set of queries, an SVM classifier can be naturally adapted to model this problem. The SVM model will attempt to solve the following quadratic optimization problem:

$$\min_w \frac{1}{2} \|w\|^2 + C \sum_{q,l} \xi_{ql} \tag{4}$$

subject to

$$\xi_{ql} > 0, \quad z_{ql} \langle w, d_{qi} - d_{qj} \rangle \geq 1 - \xi_{ql} \quad \forall q, l$$

where non-negative slack variables $\xi_{ql}$ were introduced, and the tradeoff between margin size and training error [14] is controlled by the parameter $C$.

The optimization problem in equation 4 is equivalent to:

$$\min_w \lambda \|w\|^2 + \sum_{q,l} [1 - z_{ql} \langle w, d_{qi} - d_{qj} \rangle]_+ \tag{5}$$

where $\lambda = \frac{1}{2C}$ and $[\ ]_+$ is *hinge* operator:

$$[x]_+ = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The first term in Equation 5 is a regularization term, and the second is frequently referred to as the *hinge loss*[3]. The hinge loss, a convex function, is an approximation to the empirical 0/1 loss of minimizing all misranks. Minimization of the hinge loss places an upper bound on the number of misranks in the paired dataset [14]. An illustration of the hinge loss function can be seen in Figure 2, the dashed line.

## 3.2 Sigmoid Approximation

One of the disadvantages of the hinge loss function is its sensitivity to outliers. Outlier points produce large negative scores (the far left of the score range in Figure 2). Because the hinge loss linearly increases with larger negative scores, these outliers have a strong contribution to the global loss. This large loss contribution in turn gives these outliers an important role in determining the final learned hypothesis.

To address this problem, we propose to approximate the number of misranks (the empirical 0/1 loss) using a non-linear sigmoidal function. This function can be expressed as $g(\sigma, P_l) = 1 - sigmoid(\sigma, P_l)$, where $P_l$ is the pairwise score (Equation 3), and $\sigma$ is a parameter that determines the steepness of the sigmoid function. The sigmoid function is defined as:

$$sigmoid(\sigma, x) = \frac{1}{1 + e^{-\sigma x}}.$$

The sigmoid loss with several values of $\sigma$ is illustrated as the solid lines in Figure 2.

There are at least two advantages in using this particular loss function. First, this non-linear penalty suppresses the effect of outliers, i.e., not giving larger loss values to instances with very large negative pairwise scores. Second, this penalty can arbitrarily approximate the empirical 0/1 loss by increasing the $\sigma$ parameter.

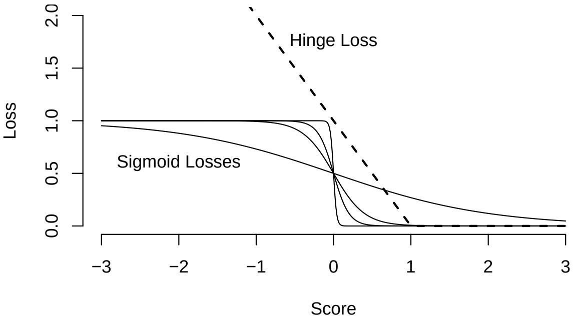

**Figure 2: Loss Functions**

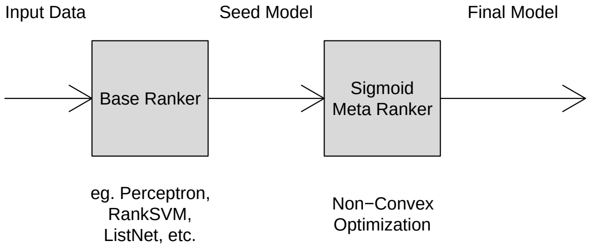

**Figure 3: Meta ranking scheme: non-convex optimization procedure is seeded with the output of a base ranking model.**

Similar to equation 5, the optimization problem with the sigmoid-based loss function can then be expressed as:

$$\min_w L(w) = \lambda \|w\|^2 + \sum_{q,l} [1 - sigmoid(\sigma, z_{ql} \langle w, d_{qi} - d_{qj} \rangle)]\,. \tag{7}$$

The sigmoid loss function is not convex, thus the learning procedure is only guaranteed to reach a local maximum. To avoid learning poor locally optimal solutions, the sigmoid ranker is used as a second optimization step, refining the hypothesis produced by another ranker. Specifically, sigmoid-based optimization is seeded with the hypothesis learned from a base ranker, such as RankSVM, and then it converges to a local optimum close to the (presumably good) seed hypothesis. The complete meta learning scheme is illustrated in Figure 3.

This sigmoidal meta-ranker is closely related to the two-stage optimization scheme proposed by Perez-Cruz et al. [19]. These researchers presented different loss functions, including a sigmoidal one, to be applied for classification tasks after being seeded by a traditional SVM classifier model. Their main goal was to better approximate the empirical classification loss (number of classification mistakes), and not to suppress outliers[19].

Tsai et. al. [23] also found that substituting a sub-linear fidelity loss function for RankNet's asymptotically linear cross-entropy loss improved ranking performance. In this regard, their algorithm, FRank, bears some similarity to the sigmoid meta-ranker proposed here. Although these two algorithms share a similar motivation, the algorithm presented here acts as a general-purpose meta-ranker for any linear seed ranker and provides added flexibility of the $\sigma$ parameter, controlling the steepness of the loss. Additionally, the FRank algorithm uses a boosting framework to optimize the fidelity loss, whereas the algorithm presented here is optimized through gradient descent (see below). Boosting tends to have slower convergence properties, taking several hundred iterations through the dataset to converge [23]. As we describe below, even when seeded with a weak base learner, the gradient descent approach typically converges much faster.

## 3.3 Learning

We utilized a gradient descent technique to learn the final ranking model $w$. Specifically, we can differentiate the sigmoid-based loss function (Equation 7) with respect to the parameter vector $w$ to obtain:

$$\frac{\partial L(w)}{\partial w} = 2w\lambda - \sum_{q,l} \sigma F(\sigma, q, l) \left[1 - F(\sigma, q, l)\right] \qquad (8)$$

where $F(\sigma, q, l) = sigmoid(\sigma, z_{ql} \langle w, d_{qi} - d_{qj} \rangle)$.

The gradient descent algorithm can then be written as:

$$w^{(k+1)} = w^{(k)} - \eta_k \frac{\partial L(w^{(k)})}{\partial w} \qquad (9)$$

where, the index $k$ defines the number of iterations (epochs) and the step size $\eta_k$ in principle can be chosen based on a line search along the descent direction, i.e.,

$$\eta_k = \operatorname*{argmin}_{\eta \geq 0} L(w^{(k)} - \eta \frac{\partial L(w^{(k)})}{\partial w}).$$

In practice, however, we used a step-size-halving heuristic for $\eta$, initially setting $\eta = 0.05$. Whenever $\eta$ was too large to yield a decrease in the loss function, $\eta$ was set to $\eta/2$, and learning stops when the relative decrease in loss was less than $10^{-8}$.

# 4. EXPERIMENTS

## 4.1 Datasets

We performed experiments on eight different ranking datasets. The first three ranking datasets are part of the Learning to Rank (LETOR) Benchmark dataset [16]. This dataset attempts to provide a standard set of document-query features over several test collections. 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 [9, 8]. 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 OHSUMED collection contains 106 queries and 25 features, TREC 2003 has 50 documents and 44 features, and there are 75 queries and 44 features in the TREC 2004 collection. Please refer to the original reference [16] for a detailed explanation of the feature sets. In our experiments, the query-document 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.

The next two ranking datasets were collected from the *Recipient Recommendation* task [6], where the goal is to find persons who are potential recipients of an email message under composition given its current contents and its previously-specified recipients. This is a ranking task in a sense that, compared to traditional document retrieval, the email under composition is the equivalent of a query, and the email addresses in the address book are the analogous of documents.

The *TOCCBCC* prediction subtask aims at predicting all recipients of a message being composed, while the *CCBCC* subtask ranks all recipients inserted in the CC or BCC fields of the message under composition. Thus the CCBCC task, in addition to the text, can use information extracted from the recipients already specified in the TO field of the email. The collection contains more than 44000 queries from 36 different users, where an average of 1267 queries per user are used for training. For testing, the TOCCBCC tasks uses an average of 144 queries per user, while 20 queries per user are used for CCBCC testing. The number of documents (email addresses) to be ranked averages 377 per user [6]. The TOCCBCC task utilizes four different features for ranking (derived from frequency, recency and summarized textual scores). The CCBCC task dataset contains 7 features: the same 4 features from the TOCCBCC task, and three additional co-occurrence features (derived from the already-specified addresses in the TO field of the message).

The last three ranking datasets were derived from SEAL, a Set Expander for Any Language system [24]. Set Expansion is the task of expanding an initial set of objects into a larger and more complete set[2] of objects of the same type. More specifically, SEAL expands textual seeds (such as “California”, “Colorado” and “Florida”) by automatically finding semi-structured web pages having lists of items, and then aggregating these lists and ranking the “most promising” items higher. The ranking then considers different set of features such as the ones derived from proximity metrics, suffixes and prefixes extracted from wrappers, and similarity scores calculated from random walks in an entity graph [24]. Our sample dataset from SEAL contains queries in three different languages. Each language contained approximately 60 queries, and each document (entity) was represented with 18 features. Experiments were carried out with a 3-fold cross-validation split with two of the languages used for training, and the remaining language use for testing.

## 4.2 Performance

In this section we describe experiments conducted with the sigmoid ranker using three baseline rankers: RankSVM, the averaged ranking perceptron [10] and ListNet [4].

The averaged perceptron ranking algorithm [10] is a simple and fast online ranking algorithm that scales linearly with the number of training examples. Although recent results suggested that this algorithm may require thousands of iterations to produce reasonable performance [10], in this paper we trained it with five iterations only. By crippling the algorithm we produced a low quality input model to the meta ranker, and investigated how the meta ranker responds to a weak initialization.

ListNet is a recent feature-based ranking algorithm [4] that instead of learning by minimizing a document pair loss

[2] Google Sets is a well-known example of a set expansion system on the web.

functions, it minimizes a probabilistic listwise loss function. That is, it utilizes document lists, instead of document pairs, as instances in the learning procedure. Although it is not a pairwise ranking algorithm, ListNet outputs a linear ranking model that can be used as input in the sigmoid optimization. Hence, not only can we investigate how ListNet compares with other pairwise baseline learners, but also study if the sigmoid meta ranker can improve a non-pairwise base ranking model.

Unless otherwise noted, in all experiments the sigmoid $\sigma$ parameter was set to 1.0, and the regularization parameter C for RankSVM was selected from a search within the discrete set $C \in \left\{10^{-5}, 10^{-4}, .., 10^{1}\right\}$ using a holdout set.

We start with experimental results from the largest ranking collection, the two recipient recommendation tasks. Performance results for these ranking tasks are illustrated in Figures 4, showing AUC (Area Under the ROC Curve), R-Precision and Mean Average Precision results for both TOCCBCC and CCBCC ranking tasks.

It is noticeable from these figures the large performance gains that the sigmoid optimization achieves with the perceptron algorithm baseline. On both tasks, the meta ranker produced significantly better results than the averaged perceptron ranker. There are also visible performance gains for sigmoid ranker applied to RankSVM, although more modest. The sigmoid optimization applied after ListNet did not seem to improve performance on the TOCCBCC tasks, even though it boosted results for the CCBCC task.

Performance on the LETOR collections are illustrated in Figure 5, showing MAP for each test collection and base learner. Mean average precision results are shown for each one of the LETOR collections (TREC-04, TREC-03 and OHSUMED) and for each ranker. In all tasks, the sigmoid optimization significantly improved results for the averaged perceptron ranker. For RankSVM, the sigmoid ranker produced improvements in all collections, with the largest gain for TREC-03. The ListNet + sigmoid ranker, on the other hand, experienced its largest performance improvement on the TREC-04 collection, although a small gain was also observed in TREC-03 as well.

Experimental results on the Set Expansion ranking collections are pictured in Figure 6. Again, visible MAP improvements in all three datasets can be observed for the sigmoid ranker on the top of the averaged perceptron. More surprising perhaps are the even larger performance gains obtained on the top of ListNet for all three datasets. Although smaller in magnitude, the sigmoid ranker also produced visible performance gains for all three SEAL datasets when applied to RankSVM.

Full results for Mean Average Precision are given in Table 1. Statistical significance tests of the "+sigmoid" columns over the values on the previous columns are indicated with $^{\star}$ or $^{\star\star}$ (for paired t-test with $p < 0.05$ or 0.01, respectively) and $^{\dagger}$ or $^{\dagger\dagger}$ (for the Wilcoxon Matched-Pairs Signed-Ranks test with $p < 0.05$ or 0.01, respectively).

Improvements provided by the sigmoid ranker were statistically significant for all base learners on all three SEAL ranking datasets. The sigmoid optimization also increases average perceptron in all ranking problems. MAP values obtained by ListNet+Sigmoid were also significantly better for the TREC-04 and CCBCC ranking tasks. Additionally, the meta ranker significantly improved RankSVM on the TOCCBCC ranking task.

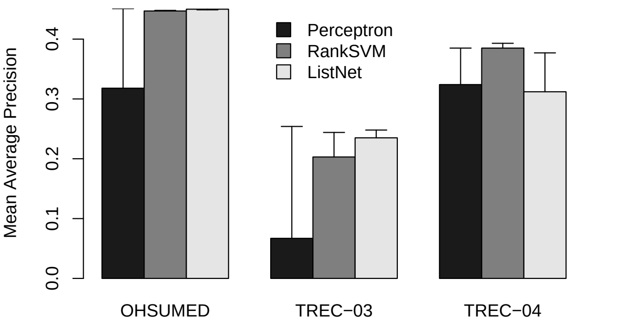

**Figure 5: Performance (MAP) on LETOR Dataset. Whisker shows baseline + sigmoid.**

It is interesting to note how significantly the perceptron ranker can be improved by the meta ranker. Its final performance numbers were comparable, and sometimes slightly better, than those obtained using the sigmoid optimization on top of the stronger base rankers. Although the sigmoid meta-ranker is only guaranteed to find a local optima, this local optima is sometimes better when the learner is seeded with a relatively weak ranking model. This may be an indication that initially using a method that is sensitive to outliers can lead the learner astray, yielding a seed model that is too strongly influenced by those outliers. The perceptron learner, however, was intentionally crippled, only making a small number of passes through the data. This training process doesn't allow the outliers to have such a strong influence on the seed model, potentially yielding a better final model.

Overall, the sigmoid meta ranker significantly improved ranking performances for most test cases in Table 1. In the LETOR datasets, however, this was not the case — although the meta ranker improved performance on average, these improvements were not statistically significant. Because the LETOR collections have a relatively larger number of features and a smaller number of queries, we speculate that these ranking models are overfitting the training data. In fact, we observed that very small changes in the RankSVM regularization parameter $C$ produced very different ranking performance on these three collections.

These results also highlight that the sigmoid ranker is in fact a general purpose linear meta ranker. Not only can it improve pairwise ranking functions, but also fine-tune any linear ranking model — as attested by the ListNet + sigmoid performance.

## 4.3 Learning Curve

Typical sigmoid ranker learning curves can be seen in Figure 7. This curve illustrates training set AUC (i.e., performance on the training set in terms of Area Under the ROC Curve) versus the number of sigmoid gradient descent iterations for a particular CCBCC prediction task [3].

The initial points (epoch=0) in Figure 7 show the AUC values obtained by the base rankers. This is the starting point of the sigmoid rank optimization. In this particular example, RankSVM provides a higher initial AUC than ListNet, which in turn outperforms the averaged perceptron.

[3] The training set AUC was shown here because it corresponds directly to minimizing the number of misranks in the training set [26].

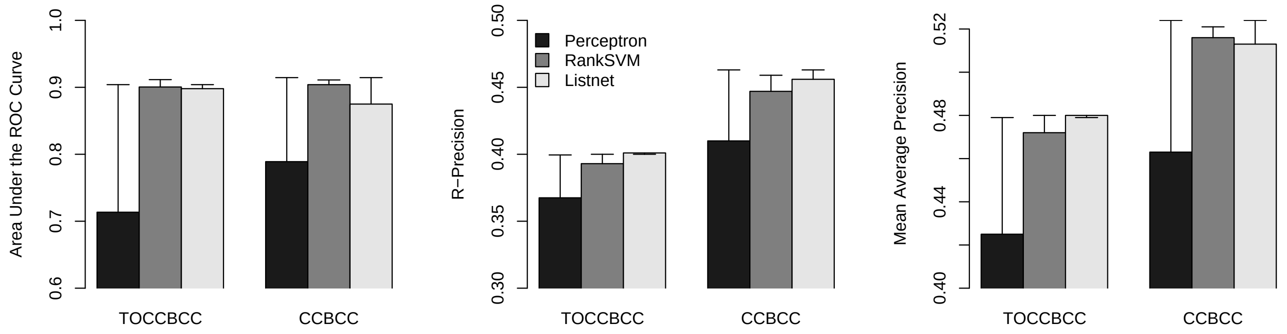

**Figure 4: Performance for the recipient recommendation ranking tasks. Whisker shows baseline + sigmoid.**

| Collection | Perceptron | +Sigmoid | RankSVM | +Sigmoid | ListNet | +Sigmoid |
|---|---|---|---|---|---|---|
| OHSUMED | 0.318 | $0.451^{**\dagger\dagger}$ | 0.447 | 0.448 | 0.450 | 0.449 |
| TREC-03 | 0.067 | $0.254^{**\dagger\dagger}$ | 0.203 | 0.244 | 0.235 | 0.248 |
| TREC-04 | 0.324 | $0.385^{*\dagger}$ | 0.385 | 0.393 | 0.312 | $0.377^{**\dagger\dagger}$ |
| SEAL-1 | 0.851 | $0.866^{**\dagger\dagger}$ | 0.862 | $0.866^{\dagger\dagger}$ | 0.843 | $0.866^{**\dagger\dagger}$ |
| SEAL-2 | 0.869 | $0.893^{**\dagger\dagger}$ | 0.890 | $0.894^{\dagger\dagger}$ | 0.864 | $0.893^{**\dagger\dagger}$ |
| SEAL-3 | 0.906 | $0.924^{**\dagger\dagger}$ | 0.916 | $0.920^{*\dagger}$ | 0.901 | $0.923^{**\dagger\dagger}$ |
| TOCCBCC | 0.425 | $0.479^{**\dagger\dagger}$ | 0.472 | $0.480^{**\dagger\dagger}$ | 0.480 | 0.479 |
| CCBCC | 0.463 | $0.524^{**\dagger\dagger}$ | 0.516 | 0.521 | 0.513 | $0.524^{**\dagger\dagger}$ |

**Table 1: Mean Average Precision values for experiments in all collections. Statistical significance tests over the values on the previous column are indicated with $^{*}$ or $^{**}$ (for paired t-test with $p < 0.05$ or 0.01, respectively) and $^{\dagger}$ or $^{\dagger\dagger}$ (for the Wilcoxon Matched-Pairs Signed-Ranks test with $p < 0.05$ or 0.01, respectively).**

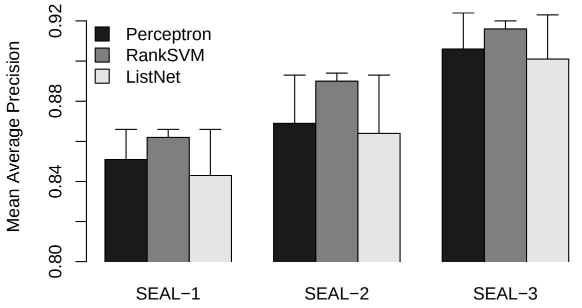

**Figure 6: Performance (MAP) on Set Expansion Experiments. Whisker shows baseline + sigmoid.**

The RankSVM+Sigmoid optimization then proceeds smoothly, with performance values reaching a plateau around 13 gradient descent iterations. ListNet+Sigmoid and Perceptron+Sigmoid start from different hypotheses, but are able to reach relatively high performance levels in less than three gradient descent iterations, and then converge to approximately the same plateau in less than 17 iterations. For comparison, one more curve was included in Figure 7: a sigmoid ranker with a random initial model. As expected, it takes considerably longer to reach reasonable AUC values, and converges to plateau levels in less than 30 iterations.

Figure 7 illustrates two reasons why the sigmoid meta ranker can provide robust pairwise ranking with a small extra computational cost. First, the number iterations necessary for convergence in the sigmoid ranker was usually small, since the starting point (the output of base learner) was already a well-tuned model. Second, the first few gradient steps were usually responsible for most of the performance gains observed. It is also important to note that this optimization step is powerful even as a stand-alone rank learner — the performance with a random initialization approaches the maximum performance of the best seeded rankers in less than 30 gradient descent steps.

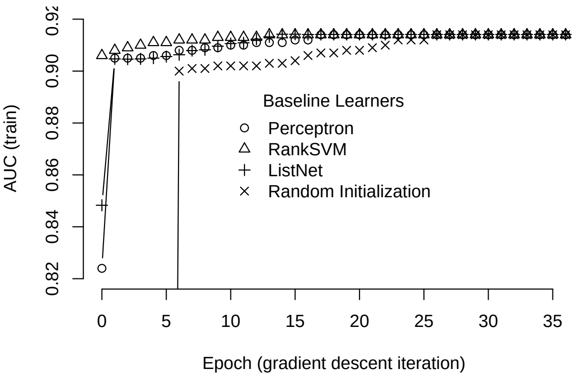

**Figure 7: Learning curve of sigmoid ranker for several baseline algorithms.**

## 4.4 Sigma Parameter

The steepness of the sigmoid function is controlled by the parameter $\sigma$. In principle, one can arbitrarily approximate the true 0/1 empirical loss function by increasing the values for this parameter. Experiments below showed, however, that increasing values of $\sigma$ do not correspond to better overall ranking performance.

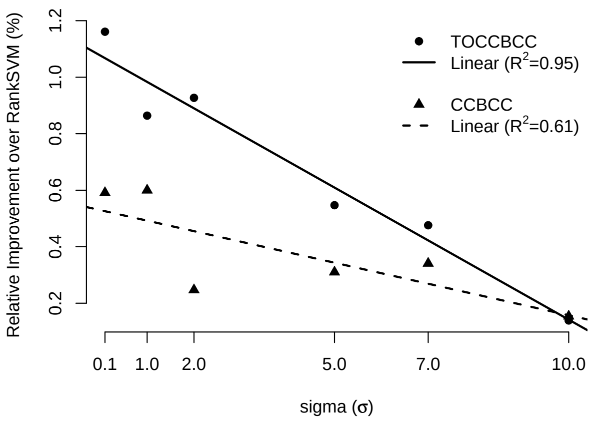

**Figure 8: Relative Improvements in MAP over RankSVM for different sigma ($\sigma$) values.**

Figure 8 shows, for both recipient prediction tasks, the relative improvements in MAP over RankSVM obtained by the sigmoid optimization, versus different values for the $\sigma$ parameter in the sigmoid function. The values of $\sigma$ considered were $\{0.1, 1, 2, 5, 7, 10\}$. Figure 8 clearly shows a trend that smaller values of $\sigma$ produce better ranking performance for both ranking tasks.

Arbitrarily increasing the $\sigma$ parameter generates steeper loss curves whose gradient information is largely concentrated around the decision region. We speculate that, for large $\sigma$ values, this lack or reduction of gradient information from other regions of the loss function is responsible for the observed lower performance.

## 5. CONCLUSIONS

We considered the effects of outliers in learning pairwise feature-based ranking functions. In pairwise ranking, pairs of documents with different label levels are taken as instances in the learning process. While this process is known to bring advantages to rank learning, it can also produce many outliers, particularly when arbitrary label level judgments are used or simply when human labelers make mistakes. Outliers in the learning procedure can compromise ranking function robustness, and consequently deteriorate ranking performance.

We illustrated the effects of outliers in pairwise ranking functions, and then introduced a new meta-learning algorithm able to suppress the undesirable outlier effects. The algorithm is a non-convex optimization procedure using a sigmoid loss, in which any linear baseline ranking function can be used as input. Experiments on several different ranking datasets showed that this meta ranker produced statistically significant performance gains over various state-of-the-art baseline rankers.

This sigmoid meta-learning algorithm provided consistent and significant performance improvements when seeded by a weak rank learner, the average perceptron. When seeded by strong baseline rankers, RankSVM and ListNet, the meta-learning algorithm improved performance 88% of the time, with 64% of those performance gains statistically significant.

## 6. REFERENCES

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

[2] 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.

[3] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking svm to document retrieval. In *SIGIR*, pages 186–193, New York, NY, USA, 2006.

[4] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*. ACM, 2007.

[5] B. Carterette, P. N. Bennett, D. M. Chickering, and S. T. Dumais. Here or there: Preference judgments for relevance. In *European Conference on Information Retrieval*, 2008.

[6] V. R. Carvalho and W. W. Cohen. Ranking users for intelligent message addressing. In *European Conference on Information Retrieval*, 2008.

[7] 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.

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

[9] 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.

[10] J. Elsas, V. R. Carvalho, and J. G. Carbonell. Fast learning of document ranking functions with the committee perceptron. In *ACM International Conference on Web Search and Data Mining*, 2008.

[11] 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.

[12] 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.

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

[14] 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.

[15] R. Khardon and G. Wachman. Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8:227–248, 2007.

[16] 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*, 2007.

[17] 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.

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

[19] F. Perez-Cruz, A. Navia-Vazquez, A. R. Figueiras-Vidal, and A. Artes-Rodriguez. Empirical risk minimization for support vector classifiers. *IEEE Transactions on Neural Networks*, 14:296–303, Mar 2003.

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

[21] M. Taylor, J. Guiver, S. Robertson, and T. Minka. Softrank: Optimizing non-smooth rank metrics. In *ACM WSDM*, 2008.

[22] 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.

[23] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. FRank: a ranking method with fidelity loss. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383–390, New York, NY, USA, 2007. ACM Press.

[24] R. C. Wang and W. W. Cohen. Language-independent set expansion of named entities using the web. In *IEEE International Conference on Data Mining (ICDM)*, 2007.

[25] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*. ACM, 2007.

[26] 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.