The LambdaLoss Framework for Ranking Metric Optimization

Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, Marc Najork
Google
Mountain View, CA
{xuanhui,chgli,nadavg,bemike,najork}@google.com

ABSTRACT

How to optimize ranking metrics such as Normalized Discounted Cumulative Gain (NDCG) is an important but challenging problem, because ranking metrics are either flat or discontinuous everywhere, which makes them hard to be optimized directly. Among existing approaches, LambdaRank is a novel algorithm that incorporates ranking metrics into its learning procedure. Though empirically effective, it still lacks theoretical justification. For example, the underlying loss that LambdaRank optimizes for remains unknown until now. Due to this, there is no principled way to advance the LambdaRank algorithm further. In this paper, we present LambdaLoss, a probabilistic framework for ranking metric optimization. We show that LambdaRank is a special configuration with a well-defined loss in the LambdaLoss framework, and thus provide theoretical justification for it. More importantly, the LambdaLoss framework allows us to define metric-driven loss functions that have clear connection to different ranking metrics. We show a few cases in this paper and evaluate them on three publicly available data sets. Experimental results show that our metric-driven loss functions can significantly improve the state-of-the-art learning-to-rank algorithms.

CCS CONCEPTS

• **Information systems → Learning to rank**;

KEYWORDS

Ranking metric optimization; LambdaLoss; LambdaRank

ACM Reference Format:
Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, Marc Najork. 2018. The LambdaLoss Framework for Ranking Metric Optimization. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271784

1 INTRODUCTION

Information Retrieval (IR) system performance is measured by ranking metrics such as Normalized Discounted Cumulative Gain (NDCG) [16], Mean Average Precision (MAP) [1], Mean Reciprocal Rank (MRR) [37], etc. These metrics are defined on the retrieved list of documents, and are intended to capture its utility for the end users. Since users, when presented with a ranked list of documents, are more likely to scan documents downwards starting at the top, most ranking metrics are rank-dependent and reward relevance of top-ranked documents more.

Learning-to-rank is an interdisciplinary research area that employs machine learning techniques to solve ranking problems in IR systems. While traditional machine learning algorithms are mainly designed for classification or regression, they have been adopted for ranking problems in the learning-to-rank setting [23]. The well-known pairwise approaches define loss functions to optimize for preferences among document pairs [3–5, 17, 22], and the listwise approaches define loss functions over the entire document lists to optimize the agreement between predictions and ground truth rankings [6, 39]. The loss functions in these approaches are smooth and convex, and thus efficiently optimized. Also, they are shown to be bounds of ranking metrics [22, 23] and work reasonably well in practice. However, the bounds of these loss functions are usually coarse because they are not designed in a metric-driven manner.

How to directly optimize ranking metrics is an important but challenging problem. The main difficulty lies in the fact that ranking metrics depend on ranks that are usually obtained by sorting documents by their scores. As a result, ranking metrics are either flat or discontinuous everywhere; they are neither smooth nor convex. Direct non-gradient optimization techniques like grid search [24, 31, 34] can be used, but they do not scale well. Thus, prior work has explored three lines of research to tackle the optimization problem in a scalable way.

The first line uses approximation. A common strategy is to approximate ranks in ranking metrics by scores that are output from ranking models [8, 27, 33, 36]. For example, SoftRank [33] first defines a distribution over all possible ranked lists after introducing uncertainty to scores. It then defines SoftNDCG as the expected NDCG over this distribution and uses it as the optimization objective. However, the main drawback of this type of approach is that the objectives are not convex (though smooth), which makes them easily stuck at local optima [23].

The second line casts learning-to-rank as a structured learning problem [21, 25, 41] in which a ranked list is treated as a unit and a loss is defined as its distance to the ideal ranked list that is sorted by relevance labels. The distance metric between ranked lists depends on the ranking metric under consideration [21]. Because there is an exponential number of possible ranked lists, training efficiency is the main bottleneck for this line of work.

The third line uses ranking metrics to dynamically re-weight instances during iterative training procedures [3–5, 29, 40]. For example, AdaRank [40] adapts the AdaBoost idea [12] to ranking problems, and uses the NDCG value of each query to compute a weight for it in the next training iteration. LambdaRank algorithms [4, 5] creatively define a weight ΔNDCG, which is the NDCG difference when a pair of documents is swapped in the current ranked list, and use it to re-weight the pair in the next training iteration. These

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10.
https://doi.org/10.1145/3269206.3271784

methods take metrics into account and are also efficient since a convex optimization problem is solved in each iteration. Different from AdaRank, which has a similar exponential loss as AdaBoost, the underlying loss of LambdaRank remains unknown, despite its empirical success [4, 7].

Our paper is motivated by the desire to understand the theoretical aspects of LambdaRank. What is the underlying loss that LambdaRank optimizes for? Without such knowledge, there are no theoretical guarantees that the LambdaRank iterations will eventually converge. To the best of our knowledge, the best effort so far justifies the convergence through empirical hypothesis testing to show its local optimality [4, 11]. Also, concerns were raised on whether LambdaRank directly optimizes NDCG or not [23]. More importantly, the lack of theoretical justification prevents us from advancing its success by creating new LambdaRank-like learning-to-rank algorithms.

In this paper, we fill this theoretical gap by proposing LambdaLoss, a probabilistic framework for ranking metric optimization. We show that LambdaRank becomes a special configuration in the LambdaLoss framework and a well-defined loss is thus presented for LambdaRank in this paper. The LambdaRank algorithms use a Expectation-Maximization procedure to optimize the loss. More interestingly, our LambdaLoss framework allows us to define metric-driven loss functions conditioning on both ranks and scores and can optimize them efficiently. We show a few cases on how to define LambdaLoss in a metric-driven manner. Along this line, we discover a ranking metric that is bounded by the underlying loss of LambdaRank and show it is more coarse than the one developed in this paper. We validate the LambdaLoss framework on three benchmark LETOR data sets [26]. Our experimental results show that our metric-driven loss functions defined in the LambdaLoss framework can significantly improve the state-of-the-art learning-to-rank algorithms.

The rest of this paper is organized as follows: In Section 2, we review previous related work. We formulate our problem in Section 3. The probabilistic framework is presented in Section 4 and our metric-driven loss functions are described in Section 5. We present our experiments in Section 6 & 7 and conclude in Section 8.

2 RELATED WORK

Learning-to-rank is an extensively studied research field, and multiple optimization algorithms for ranking problems were proposed in prior art (see Liu [23] for a comprehensive survey of the field). In general, learning-to-rank methods fall into three main categories: pointwise, pairwise and listwise methods. Pointwise were the earliest proposed learning-to-rank methods. These methods define loss using regression [14], classification [15] or ordinal regression [9] methods, and were generally found less effective for information retrieval applications [23]. Pairwise learning-to-rank methods [3, 17] model pairwise preferences among documents in the ranked list, and while being more effective than the pointwise methods, they are still less effective compared to the listwise methods [6, 39] that define their loss over the entire document list. All these methods work reasonably well in practice but their connection to ranking metrics are loose.

To optimize ranking metrics, the most straightforward way is through non-gradient techniques like grid search, coordinate search or other exhaustive search techniques [24, 31, 34]. However, since IR metrics are neither smooth nor convex, these methods provide no guarantees regarding reaching true global maxima. In addition, since these methods require search over the entire parameter space and evaluation over the entire data set for each search, they do not scale very well as the number of features or training instances increases. Approximation-based methods such as SoftRank [33], SmoothRank [8] and ApproxNDCG [27], instead focus on optimizing a continuous and differentiable approximation of the target metric, but suffer from being easily stuck at local optima. Structural learning algorithms [21, 25, 41] optimize ranking metrics based on structural SVM algorithms [35] but are computationally costly due to the exponential number of constraints in their formulations. AdaRank [40] and LambdaRank [5] are two iterative methods and use ranking metrics to dynamically re-weight training examples after each iteration.

Our work is motivated by LambdaRank [5]. The initial version of LambdaRank was based on neural network models. A later version using the same procedure, but based on gradient boosting decision tree models is known as LambdaMART [4]. LambdaMART was empirically shown to be a state-of-the-art model (e.g., it was the winning submission at the Yahoo! learning-to-rank challenge [7]), however there are no theoretical guarantees that it indeed optimizes NDCG over the training data. This paper attempts to address this gap and extends it to a framework that can define and optimize metric-driven loss functions.

3 PROBLEM FORMULATION

We formulate our problem in the learning-to-rank framework in this section.

3.1 Learning-to-Rank

In the learning-to-rank setting, the training data T contains a set of queries Q with each having a list of result documents. For each document, we have a feature vector as well as a relevance label; and thus each query has a list $\mathbf{x}_q$ of document feature vectors and a list $\mathbf{y}_q$ of document relevance labels. Thus the training data is represented as

$$T = \{(\mathbf{x}_q, \mathbf{y}_q) | q \in Q\}.$$

In the following discussion, we drop the subscript q and use $\mathbf{x}$ and $\mathbf{y}$ for conciseness.

A learning-to-rank algorithm is to find a ranking model Φ that can predict the relevance scores $\mathbf{s}$ for all documents in a query:

$$\mathbf{s} = \Phi(\mathbf{x})$$

We take a generic form of the ranking model Φ in our paper. In practice, Φ can be implemented in different forms. For example, RankingSVM [17] uses a linear model; LambdaMART [4] uses a tree-based model; RankNet [3] uses a neural network model. Though the implementation can be different, all the learning-to-rank algorithms learn Φ using loss functions as their objectives. A loss function is in general defined based on relevance labels $\mathbf{y}$ and predicted scores

$\mathbf{s} = \Phi(\mathbf{x})$:

$$\mathcal{L}(\Phi) = \frac{1}{|T|} \sum_{(\mathbf{x},\mathbf{y})\in T} l(\mathbf{y}, \Phi(\mathbf{x})) \quad (1)$$

where l is the loss function for a single query that takes the labels and scores as input and output a real value as the loss.

$$l : (\mathbf{y}, \mathbf{s}) \rightarrow \mathbb{R} \quad (2)$$

A learning-to-rank algorithm is to find the optimal Φ that minimizes the overall loss:

$$\hat{\Phi} = \arg\min_{R(\Phi)} \mathcal{L}(\Phi)$$

in the space of ranking models $R(\Phi)$.

3.2 Simple Loss Functions

How to define the loss l in Eq 2 is an important factor for learning-to-rank algorithms. Suppose we have n documents and we use y_i and s_i to represent the label and score for the i-th document. The loss l can be defined in a pointwise manner, like a mean square error

$$l(\mathbf{y}, \mathbf{s}) = \sum_{i=1}^{n} (y_i - s_i)^2$$

Such a loss function is commonly used in regression problems and it is not specific for ranking problems. The introduction of pairwise and listwise loss functions incubates the learning-to-rank research area [3, 6, 17], where the loss is defined to measure the correctness of relative document orders, instead of error in absolute relevance prediction. A commonly used pairwise loss function is the logistic loss

$$\begin{aligned} l(\mathbf{y}, \mathbf{s}) &= \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{I}_{y_i > y_j} \log_2(1 + e^{-\sigma(s_i - s_j)}) \quad (3) \\ &= \sum_{y_i > y_j} \log_2(1 + e^{-\sigma(s_i - s_j)}) \end{aligned}$$

where $\mathbb{I}$ is the indicator function and σ is a hyper-parameter. This loss is based on cross entropy and used in the RankNet algorithms [3–5]. The intuition is to apply a penalty on the out-of-order pair (i, j) that has $y_i > y_j$ but $s_i < s_j$. Note that we use $\log_2$ instead of log purposely to facilitate the discussion in Section 5.

3.3 Ranking Metrics

There are many existing ranking metrics such as NDCG and MAP used in IR problems. A common property of these metrics is that they are rank-dependent and place more emphasis on performance of the top ranked documents. For example, the commonly adopted NDCG metric for a single query over the document list ranked by decreasing scores $\mathbf{s}$ is defined as

$$\text{NDCG} = \frac{1}{\text{maxDCG}} \sum_{i=1}^{n} \frac{2^{y_i} - 1}{\log_2(1 + i)} = \sum_{i=1}^{n} \frac{G_i}{D_i} \quad (4)$$

where

$$G_i = \frac{2^{y_i} - 1}{\text{maxDCG}}, \; D_i = \log_2(1 + i)$$

are gain and discount functions respectively and maxDCG is a normalization factor per query and computed as the DCG for the list ranked by decreasing relevance labels $\mathbf{y}$ of the query. Please note that maxDCG is a *constant* factor per query in NDCG.

Ideally, learning-to-rank algorithms should use ranking metrics as learning objectives. However, it is easy to see that sorting by scores is needed to obtain ranks. This makes ranking metrics either flat or discontinuous everywhere, so they cannot be directly optimized efficiently.

3.4 LambdaRank

Bridging the gap between evaluation metrics and loss functions has been studied actively in the past [23]. Among them, LambdaRank or its tree-based variant LambdaMART has been one of the most effective algorithms to incorporate ranking metrics in the learning procedure. The basic idea is to dynamically adjust the loss during the training based on ranking metrics. Using NDCG as an example, ΔNDCG is defined as the absolute difference between the NDCG values when two documents i and j are swapped

$$\Delta\text{NDCG}(i, j) = |G_i - G_j||\frac{1}{D_i} - \frac{1}{D_j}| \quad (5)$$

LambdaRank uses the logistic loss in Eq 3 and adapts it by re-weighing each document pair by ΔNDCG in each iteration

$$l(\mathbf{y}, \mathbf{s}) = \sum_{y_i > y_j} \Delta\text{NDCG}(i, j) \log_2(1 + e^{-\sigma(s_i - s_j)}) \quad (6)$$

3.5 Research Problems

LambdaRank has been shown empirically to improve ranking quality and looks very related to the NDCG metric. However, there are a few outstanding questions about it: (1) From the theoretical perspective, does the iterative procedure employed by LambdaRank converge? (2) Is there an underlying global loss function in Eq 1 for LambdaRank and how is it related to the NDCG metric? These questions, especially the second one, have puzzled researchers for a while. For example, in Chapter 3.3.7 of Liu's book [23], the author raised the concern on claiming that LambdaRank directly optimizes NDCG. Broadly, we are more interested in closing the gap between learning loss functions and ranking metrics.

4 THE LAMBDALOSS FRAMEWORK

In this section, we present the LambdaLoss framework and provide an underlying loss for LambdaRank by formulating it as a special configuration in the LambdaLoss framework.

4.1 Loss Function

We formulate the loss function in a probabilistic manner. Similar to previous work [33], we assume that scores of documents $\mathbf{s}$ determine a distribution over all possible ranked lists or permutations. Let π denote a ranked list and we use $\{P(\pi|\mathbf{s}) : \pi \in \Pi\}$ to denote the distribution. In our framework, we treat the ranked list π as a hidden variable and define the loss based on the likelihood of observing relevance $\mathbf{y}$ given $\mathbf{s}$ (or equivalently Φ and $\mathbf{x}$) using a mixture model over Π:

$$P(\mathbf{y}|\mathbf{s}) = \sum_{\pi\in\Pi} P(\mathbf{y}|\mathbf{s}, \pi) P(\pi|\mathbf{s})$$

where $\mathbf{s} = \Phi(\mathbf{x})$ and we use them interchangeably. Given a set of training instances $T = \{(\mathbf{x}, \mathbf{y})\}$, we define the loss $l(\mathbf{y}, \mathbf{s})$ in Eq 2

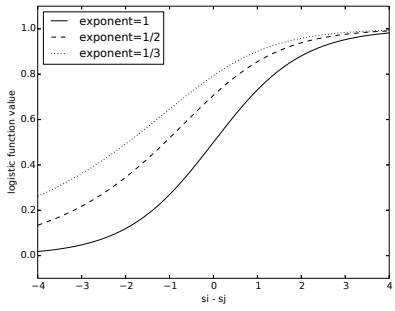


Figure 1: Generalized logistic distributions with different exponents.

as the negative log likelihood based on the maximum likelihood principle

$$l(\mathbf{y}, \mathbf{s}) = -\log_2 P(\mathbf{y}|\mathbf{s}) = -\log_2 \sum_{\pi \in \Pi} P(\mathbf{y}|\mathbf{s}, \pi) P(\pi|\mathbf{s}) \quad (7)$$

There are two main terms in the loss function: likelihood $P(\mathbf{y}|\mathbf{s}, \pi)$ and ranked list distribution $P(\pi|\mathbf{s})$. Clearly, different formulations of these two terms give different loss functions. We show a few ones in the following.

4.1.1 Likelihood $P(\mathbf{y}|\mathbf{s}, \pi)$. We start with the basic Bradley-Terry model [2] as our formulation for likelihood. In the basic Bradley-Terry model, the probability that document i is more relevant than j, i.e., $y_i > y_j$, is only based on their scores s_i and s_j, regardless of π:

$$P(y_i > y_j | s_i, s_j) = \frac{1}{1 + e^{-\sigma(s_i - s_j)}}$$

Such a model makes intuitive sense: $P(y_i > y_j|s_i, s_j) > 0.5$ when $s_i > s_j$ and vice versa. By treating the relevance labels $\mathbf{y}$ as a set of pairwise preference observations, the log likelihood in Eq 7 becomes

$$l(\mathbf{y}, \mathbf{s}) = -\log_2 P(\mathbf{y}|\mathbf{s}) = \sum_{y_i > y_j} \log_2(1 + e^{-\sigma(s_i - s_j)})$$

because of the independency of π, i.e., $P(\mathbf{y}|\mathbf{s}, \pi) = P(\mathbf{y}|\mathbf{s})$. Thus our loss function boils down to the logistic loss used in RankNet which was also shown in [30].

Our loss in Eq 7 can be more sophisticated because it allows the dependence on π in the likelihood term $P(\mathbf{y}|\mathbf{s}, \pi)$. For example, assuming that π_i and π_j are ranks in π for document i and j, we define

$$P(y_i > y_j | s_i, s_j, \pi_i, \pi_j) = [\frac{1}{1 + e^{-\sigma(s_i - s_j)}}]^{|G_i - G_j||\frac{1}{D(\pi_i)} - \frac{1}{D(\pi_j)}|}$$

where $D(\pi_i)$ and $D(\pi_j)$ are the position discount functions and G_i and G_j are the gain functions used in NDCG in Eq 4. Such a definition is very similar to the basic Bradley-Terry model but is now rank-sensitive [30]. Intuitively, the chance of observing document i is more relevant than j depends on both their difference in relevance labels and their positions in a ranked list. Mathematically, it belongs to the family of generalized logistic distributions [19] and a few examples with different exponents are shown in Figure 1. When the exponent becomes larger, the curve is more sharp and the loss is thus more sensitive to the score difference. Thus, we have the loss

$$l(\mathbf{y}, \mathbf{s}) = -\sum_{y_i > y_j} \log_2 \sum_{\pi} P(y_i > y_j | s_i, s_j, \pi_i, \pi_j) P(\pi|\mathbf{s}) \quad (8)$$

Such a loss function cannot be optimized directly by a conventional learning-to-rank algorithm like RankNet. We will present how to optimize this type of loss function in Section 4.2.

4.1.2 Ranked List Distribution $P(\pi|\mathbf{s})$. The term $P(\pi|\mathbf{s})$ can take different forms. We discuss two and compare their approach with ours in this section.

In SoftRank [33], uncertainty was introduced to each score s_i using a normal distribution $\mathcal{N}(s_i, \epsilon)$ with Gaussian noise ϵ. Ranked lists π are still determined by sorting scores but now becomes uncertain due to Gaussian noise. Thus a distribution $\mathcal{N}(\pi|\mathbf{s})$ with soft assignment over Π is formed. Such a distribution is not easy to write analytically, but easy to draw a sample by randomly sampling a score from the normal distribution of each score and then sorting to form a ranked list. The objective used in SoftRank is the expected NDCG over $\mathcal{N}(\pi|\mathbf{s})$:

$$\text{SoftNDCG} = \sum_{\pi} \text{NDCG}(\pi, \mathbf{y}) \mathcal{N}(\pi|\mathbf{s})$$

where the NDCG is computed based on ranking π and labels $\mathbf{y}$. The SoftNDCG can be thought as the negative of a loss function $l(\mathbf{y}, \mathbf{s})$. It is smooth with respect to the scores $\mathbf{s}$ but not convex. Gradient descent was used and it is easy to get stuck at a local optimum.

In the ListNet approach [6], $P(\pi|\mathbf{s})$ is formed as the Plackett-Luce model $PL(\pi|\mathbf{s})$. It also defines a distribution $PL(\pi|\mathbf{y})$ by treating $\mathbf{y}$ as scores in the Plackett-Luce model. The cross entropy between these two distributions is used as the loss

$$l(\mathbf{y}, \mathbf{s}) = \text{CrossEntropy}(PL(\pi|\mathbf{y}), PL(\pi|\mathbf{s}))$$

Though theoretically interesting, such a loss is hard to use in practice due to the huge size of Π. The ListNet approach reduces the problem by using the marginal distribution of the first position and the cross entropy then becomes

$$l(\mathbf{y}, \mathbf{s}) = -\sum_i \frac{e^{y_i}}{\sum_j e^{y_j}} \log_2 \frac{e^{s_i}}{\sum_j e^{s_j}}$$

Gradient descent was also used in the ListNet optimization.

The loss in our LambdaLoss framework is different from these two and we discuss our optimization method next.

4.2 Optimization by EM Algorithm

The loss l in Eq 7 is not convex in general. Fortunately, such a loss can be minimized by the well-known Expectation-Maximization (EM) algorithm [10]. EM is an iterative procedure and can be proven to converge to a local minimum. It starts from a random guess of model $\Phi^{(0)}$ and iterates over E-step and M-step to update the model. At iteration t, in the E-step, the distribution of hidden variable π is estimated based on the current ranking model $\Phi^{(t)}$. After the distribution of hidden variable $P^{(t)}(\pi|\mathbf{s})$ is estimated, the M-step is to re-estimate the ranking model $\Phi^{(t+1)}$ to minimize the negative likelihood, i.e., likelihood loss, using complete data

$C = \{(\mathbf{y}, \mathbf{x}, \pi)|\pi \sim P^{(t)}(\pi|\mathbf{s})\}$ where π follows the distribution $P^{(t)}(\pi|\mathbf{s})$.

$$\Phi^{(t+1)} = \arg\min \mathcal{L}_C(\Phi) = \arg\min \frac{1}{|T|} \sum_{(\mathbf{y},\mathbf{x})\in T} l_C(\mathbf{y}, \Phi(\mathbf{x}))$$

where

$$\begin{aligned} l_C(\mathbf{y}, \mathbf{s}) &= -\sum_{\pi} P^{(t)}(\pi|\mathbf{s}) \log_2 P(\mathbf{y}|\mathbf{s}, \pi) \\ &\approx \frac{1}{K} \sum_{k=1}^{K} \log_2 P(\mathbf{y}|\mathbf{s}, \pi^k) \end{aligned} \tag{9}$$

where π^k is the k-th sample of ranked list from $P^{(t)}(\pi|\mathbf{s})$ and $\mathcal{L}_C(\Phi)$ is the likelihood loss over the complete data. Since $P^{(t)}(\pi|\mathbf{s})$ is known, computing $\Phi^{(t+1)}$ is efficient when $\log_2 P(\mathbf{y}|\mathbf{s}, \pi)$ is convex. However, the M-step is still computationally prohibitive because the $P^{(t)}(\pi|\mathbf{s})$ for all $\pi \in \Pi$ are needed in theory. In practice, A few samples of ranked lists are good enough to give a good estimation of $\Phi^{(t+1)}$ and this is a common practice of Monte Carlo integration [28].

In particular, we can use a hard assignment distribution $H(\pi|\mathbf{s})$ to reduce the computational cost. That is,

$$H(\hat{\pi}|\mathbf{s}) = 1 \text{ and } H(\pi|\mathbf{s}) = 0 \text{ for all } \pi \neq \hat{\pi}$$

where $\hat{\pi}$ is the one in which all documents are sorted by decreasing scores $\mathbf{s}$. In fact, such a distribution is the limit distribution when $\epsilon \to 0$ in the normal distribution. The EM algorithm is similar to the one used in the k-means clustering algorithm.

4.3 LambdaRank in the Framework

In this section, we use the hard assignment distribution $H(\pi|\mathbf{s})$ in the loss in Eq 8 and show that LambdaRank is an EM procedure to optimize this loss

$$l(\mathbf{y}, \mathbf{s}) = -\sum_{y_i > y_j} \log_2 \sum_{\pi} P(y_i > y_j|s_i, s_j, \pi_i, \pi_j) H(\pi|\mathbf{s}) \tag{10}$$

In the E-step, we compute scores for documents to obtain $\mathbf{s}$ and then rank all documents per query to obtain $\hat{\pi}$. In the M-step, the loss over complete data becomes

$$\begin{aligned} l_C(\mathbf{y}, \mathbf{s}) &= -\sum_{\pi} P^{(t)}(\pi|\mathbf{s}) \log_2 P(\mathbf{y}|\mathbf{s}, \pi) \\ &= -\sum_{\pi} H^{(t)}(\pi|\mathbf{s}) \log_2 P(\mathbf{y}|\mathbf{s}, \pi) \\ &= -\log_2 P(\mathbf{y}|\mathbf{s}, \hat{\pi}) \end{aligned}$$

Then

$$\begin{aligned} l_C(\mathbf{y}, \mathbf{s}) &= -\sum_{y_i > y_j} \log_2([\frac{1}{1+e^{-\sigma(s_i - s_j)}}]^{|G_i - G_j||\frac{1}{D(\hat{\pi}_i)} - \frac{1}{D(\hat{\pi}_j)}|}) \\ &= \sum_{y_i > y_j} |G_i - G_j||\frac{1}{D(\hat{\pi}_i)} - \frac{1}{D(\hat{\pi}_j)}| \log_2(1 + e^{-\sigma(s_i - s_j)}) \\ &= \sum_{y_i > y_j} \Delta\text{NDCG} \log_2(1 + e^{-\sigma(s_i - s_j)}) \end{aligned} \tag{11}$$

Thus, ΔNDCG can be computed for every pair of documents and used as weights for each pair in the M-step to compute $\Phi^{(t+1)}$ in LambdaRank.

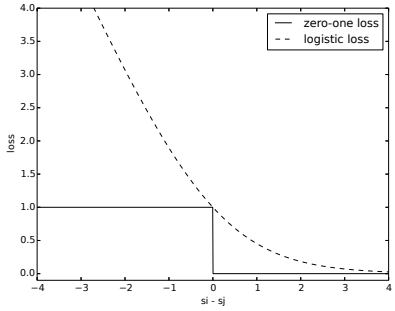


Figure 2: Zero-one loss is bounded by logistic loss.

Such an EM procedure can be implemented differently depending on the underline model structure. For example, in LambdaMART [4], the model Φ is implemented as gradient boosting decision trees and $\Phi^{(t+1)}$ is built upon $\Phi^{(t)}$ by adding a new boosting tree that regresses to the gradient of the loss in Eq 11. From the EM perspective, such a technique is viewed as a heuristic in practice because $\Phi^{(t+1)}$ can be built from scratch, without necessarily relying on $\Phi^{(t)}$. The heuristic of basing $\Phi^{(t+1)}$ on $\Phi^{(t)}$, however, can make the convergence faster and a similar strategy was also used in [38].

Next, we turn to the problem of how to define likelihood $P(\mathbf{y}|\mathbf{s}, \pi)$ or equivalently $\log_2 P(\mathbf{y}|\mathbf{s}, \pi)$.

5 METRIC-DRIVEN LOSS FUNCTIONS

One appealing property of the LambdaLoss framework is that $P(\mathbf{y}|\mathbf{s}, \pi)$ allows us to take both ranks and scores into our loss definitions. This provides a natural bridge between ranking metrics that purely rely on ranks and the traditional loss functions that purely rely on scores. Thus our approach is different from previous work on ranking metrc optimization where they replace ranks fully by scores [8, 27, 33].

In this section, we show a few cases to define loss functions in a metric-driven manner. The main upper bound function we use is

$$\mathbb{I}_{s_i < s_j} \leq \log_2(1 + e^{-\sigma(s_i - s_j)}) \tag{12}$$

where $\mathbb{I}_{s_i < s_j}$ is the zero-one loss and it is bounded by the logistic loss. The relationship is depicted in Figure 2 with $\sigma = 1$.

In the following, we assume to use the hard assignment distribution $H(\pi|\mathbf{s})$ and discuss how to handle the soft assignment at the end of this section. We use $\hat{\pi}$ to denote the mode of $H(\pi|\mathbf{s})$ and i and j to denote the ranks in $\hat{\pi}$ that is sorted by scores $\mathbf{s}$.

5.1 Average Relevance Position

The first metric is the Average Relevance Position (ARP) proposed in [18]. The ARP was originally proposed for binary relevance and we extend it to graded relevance here. For a single query,

$$\text{ARP} = \sum_{i=1}^{n} y_i \cdot i \tag{13}$$

where i is the rank and y_i is the graded relevance for the document at rank i in $\hat{\pi}$. Let s_i be the score, then $i = \sum_{j=1}^{n} \mathbb{I}_{s_i < s_j} + 1$ and we

have

$$\begin{aligned}\text{ARP} &= \sum_{i=1}^{n} y_i (\sum_{j=1}^{n} \mathbb{I}_{s_i<s_j} + 1) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} y_i \mathbb{I}_{s_i<s_j} + C_1 \\ &\le \sum_{i=1}^{n}\sum_{j=1}^{n} y_i \log_2(1+e^{-\sigma(s_i-s_j)}) + C_1\end{aligned}$$

where $C_1 = \sum_{i=1}^{n} y_i$ is a constant and the last step is based on Eq 12. The corresponding LambdaLoss is

$$l(\mathbf{y},\mathbf{s}) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \log_2 \sum_{\pi} (\frac{1}{1+e^{-\sigma(s_i-s_j)}})^{y_i} H(\pi|\mathbf{s})$$

and we denote this as **ARP-Loss1**. Such a formulation does not depend on ranks, but only on scores. It is an upper-bound for the ARP metric and can be minimized by standard pairwise loss algorithms where y_i is the weight for pair (s_i, s_j) and y_j is the weight for pair (s_j, s_i). For ARP, we can actually have an alternative formulation

$$\begin{aligned}\text{ARP} - C_1 &= \frac{1}{2}(\sum_{i=1}^{n}\sum_{j=1}^{n}(y_i \mathbb{I}_{s_i<s_j} + y_j \mathbb{I}_{s_j<s_i})) \\ &= \sum_{i=1}^{n}(\sum_{j:y_j<y_i} y_i \mathbb{I}_{s_i<s_j} + y_j \mathbb{I}_{s_j<s_i}) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j:y_j=y_i} y_j \\ &= \sum_{i=1}^{n}(\sum_{j:y_j<y_i} y_i \mathbb{I}_{s_i<s_j} + y_j \mathbb{I}_{s_j<s_i}) + C_2 \\ &= \sum_{i=1}^{n}\sum_{j:y_j<y_i} |y_i - y_j| \mathbb{I}_{s_i<s_j} + C_3 + C_2 \\ &\le \sum_{i=1}^{n}\sum_{j:y_j<y_i} |y_i - y_j| \log_2(1+e^{-\sigma(s_i-s_j)}) + C_3 + C_2 \\ &= \sum_{y_i>y_j} |y_i - y_j| \log_2(1+e^{-\sigma(s_i-s_j)}) + C_3 + C_2\end{aligned}$$

where $C_2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{j:y_j=y_i} y_j$ and $C_3 = \sum_{i=1}^{n}\sum_{j:y_j<y_i} y_j$ are constant. We denote this as **ARP-Loss2**. This loss is closely related to the RankNet loss in Eq 3, where $|y_i - y_j|$ is replaced by $\mathbb{I}_{y_i>y_j}$. The ARP-Loss2 gives a theoretical justification of using the difference between relevance labels for pairwise loss.

5.2 NDCG

Different from ARP that is a cost-based function, NDCG in Eq 4 is a gain-based function. To derive its loss function, we define:

$$\text{NDCGcost} = \sum_{i=1}^{n} G_i - \sum_{i=1}^{n} \frac{G_i}{D_i} = \sum_{i=1}^{n} G_i(1 - \frac{1}{D_i}) \tag{14}$$

Based on $D_i - 1 = \log_2(1+i) - 1 \le i - 1$, we have

$$\begin{aligned}\text{NDCGcost} &= \sum_{i=1}^{n} \frac{G_i}{D_i}(D_i - 1) \le \sum_{i=1}^{n} \frac{G_i}{D_i}(i-1) \\ &= \sum_{i=1}^{n} \frac{G_i}{D_i} \sum_{j=1}^{n} \mathbb{I}_{s_i<s_j} \le \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{G_i}{D_i} \log_2(1+e^{-\sigma(s_i-s_j)})\end{aligned}$$

The bound is a combination of scores and ranks and can be optimized by our EM algorithm with LambdaLoss

$$l(\mathbf{y},\mathbf{s}) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \log_2 \sum_{\pi} (\frac{1}{1+e^{-\sigma(s_i-s_j)}})^{\frac{G_i}{D_i}} H(\pi|\mathbf{s})$$

We name it **NDCG-Loss1**. In the M-step, the weight for a pair (s_i, s_j) is set to be $\frac{G_i}{D_i}$.

The bound used in NDCG-Loss1 may be too loose when i become larger. We define the second loss for NDCG based on the chain rule:

$$1 - \frac{1}{D_i} = \sum_{j=1}^{i-1} |\frac{1}{D_{|i-j|}} - \frac{1}{D_{|i-j|+1}}| = \sum_{j=1}^{i-1} \delta_{ij}$$

where

$$\delta_{ij} = |\frac{1}{D_{|i-j|}} - \frac{1}{D_{|i-j|+1}}| \tag{15}$$

It is easy to show $\delta_{ij} = \delta_{ji}$ and we also define $\delta_{ii} = 0$. Thus,

$$\begin{aligned}\text{NDCGcost} &= \sum_{i=1}^{n} G_i \sum_{j=1}^{i-1} \delta_{ij} = \sum_{i=1}^{n} G_i \sum_{j=1}^{n} \delta_{ij} \mathbb{I}_{s_i<s_j} \\ &= \sum_{i=1}^{n}\sum_{j:y_j<y_i} \delta_{ij}[G_i \mathbb{I}_{s_i<s_j} + G_j \mathbb{I}_{s_j<s_i}] + C_2 \\ &= \sum_{i=1}^{n}\sum_{j:y_j<y_i} \delta_{ij}|G_i - G_j| \mathbb{I}_{s_i<s_j} + C_3 + C_2 \qquad (16)\end{aligned}$$

where $C_2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{j:G_j=G_i} \delta_{ij} G_j$ and $C_3 = \sum_{i=1}^{n}\sum_{j:G_j<G_i} \delta_{ij} G_j$. Both C_2 and C_3 depend on the current ranking $\hat{\pi}$ only. They are constant in the M-step since the ranked lists or their distribution are determined. The corresponding LambdaLoss of Eq 16 is

$$l(\mathbf{y},\mathbf{s}) = -\sum_{y_i>y_j} \log_2 \sum_{\pi} (\frac{1}{1+e^{-\sigma(s_i-s_j)}})^{\delta_{ij}|G_i-G_j|} H(\pi|\mathbf{s})$$

We denote this as **NDCG-Loss2**.

NDCG-Loss2 is a potentially tighter bound than NDCG-Loss1 and can be extended to all NDCG-like metrics by redefining G_i, D_i, and subsequently δ_{ij}. For example, using $G_i = y_i$ and $D_i = \frac{1}{i}$, NDCG-Loss2 can be used to optimize the MRR-like metrics for binary cases.

5.3 Metric for LambdaRank

We have derived loss functions for metrics and thus connect metrics with different loss functions. It is intriguing to ask the question from the opposite direction: What is the metric that LambdaRank loss function optimizes for? In fact, we have the following proposition.

Proposition 5.1. *The LambdaRank loss is a bound for the following metric:*

$$\text{Metric}_{\text{LambdaRank}} = \sum_{i=1}^{n} G_i \sum_{j=1}^{i-1} |\frac{1}{D_i} - \frac{1}{D_j}|. \tag{17}$$

Proof. Let $\rho_{ij} = |\frac{1}{D_i} - \frac{1}{D_j}|$. Then $\rho_{ij} = \rho_{ji}$ and we have

$$\begin{aligned}\text{Metric}_{\text{LambdaRank}} &= \sum_{i=1}^{n} G_i \sum_{j=1}^{n} \mathbb{I}_{s_i<s_j}\rho_{ij} \\ &= \sum_{i=1}^{n} \sum_{j:y_j<y_i} \rho_{ij}[G_i\mathbb{I}_{s_i<s_j} + G_j\mathbb{I}_{s_j<s_i}] + C_2 \\ &= \sum_{i=1}^{n} \sum_{j:y_j<y_i} \rho_{ij}|G_i - G_j|\mathbb{I}_{s_i<s_j} + C_3 + C_2\end{aligned}$$

where C_2 and C_3 can be defined similarly as NDCG-Loss2. It can be seen that $\rho_{ij}|G_i - G_j|$ is the ΔNDCG and we get the LambdaRank loss in Eq 6 using Eq 12. □

Interestingly, NDCGCost $\leq \text{Metric}_{\text{LambdaRank}}$ since $\sum_{j=1}^{i-1}\delta_{ij} = \rho_{i1} \leq \sum_{j=1}^{i-1}\rho_{ij}$. Thus LambdaRank optimizes a coarse upper bound of NDCG. Also it is interesting to notice the structural similarity between NDCG-Loss2 and LambdaRank loss and a hybrid loss can be obtained by a linear combination of ρ_{ij} and δ_{ij}, as shown in our experiments.

5.4 Remarks

We primarily worked with the hard assignment $\hat{\pi}$ in this section. In the soft assignment setting, we have $\pi \neq \hat{\pi}$ and the document at rank i in $\hat{\pi}$ can have a different rank i' in π. In this case, $i' \neq i = 1 + \sum_{j=1}^{n}\mathbb{I}_{s_i<s_j}$. To reuse our above derivation, we can apply a scale trick. For example, the NDCGcost for π becomes

$$\sum_{i=1}^{n} G_i(1 - \frac{1}{D_{i'}}) = \sum_{i=1}^{n} G_i \frac{1 - \frac{1}{D_{i'}}}{1 - \frac{1}{D_i}}(1 - \frac{1}{D_i}) = \sum_{i=1}^{n} G'_i(1 - \frac{1}{D_i})$$

where $G'_i = G_i \frac{1-\frac{1}{D_{i'}}}{1-\frac{1}{D_i}}$. We can thus define a LambdaLoss that is an upper-bound of NDCGcost in the soft assignment setting based on G'_i. Due to space limit, we leave this study as our future work.

6 EXPERIMENT SETUP

Our evaluation uses a standard supervised learning-to-rank framework [23]. This section describes our evaluation data sets, evaluation metrics, and competing methods used in our experiments.

6.1 Data Sets

The three data sets we used in our experiments come from a YAHOO LTRC data set and the benchmark LETOR data sets [26]. These data sets are publicly available for the research community. All of them are data sets for web search and the largest data sets publicly available for learning-to-rank algorithms. The relevance labels of documents for each query are rated by human in the form of multi-level graded relevance.

The first dataset is YAHOO LTRC that is from YAHOO! Learning to Rank Challenge [7]. It contains two sets with each being divided into three partitions for training, validation, and testing. We use the Set1 of this data set. The other two datasets, WEB10K and WEB30K, were released by Microsoft [26]. Each one of them contains five folds with every fold being divided into three partitions for training, validation, and testing. We use the Fold1 in these two data sets respectively. The statistics for these three data sets are displayed in Table 1.

6.2 Evaluation Metrics

We evaluate different methods using NDCG of the top k ranked documents (i.e., NDCG@k). They are standard metrics for evaluating the quality of ranked lists with multi-level graded relevance [32]. The difference between NDCG@k and the NDCG in Eq 4 is that only the top k documents are used: The top k documents ranked by scores $\mathbf{s}$ are used to compute DCG and the top k documents ranked by labels $\mathbf{y}$ are used to compute maxDCG in Eq 4. In our experiments, we use NDCG@5 as our primary metric to compare different models and study different k values separately.

For our metric-driven loss functions, we adapt them for NDCG@k by truncating the loss. For example, we have NDCG-Loss2 variant for NDCG@k as

$$\sum_{i=1}^{n} \sum_{j:y_j<y_i} \mathbb{I}_{i\leq k \text{ or } j\leq k}\delta_{ij}|G_i - G_j|\log_2(1 + e^{-\sigma(s_i-s_j)}) \tag{18}$$

for the formulation in Eq 16. Other loss functions are adapted similarly.

In our experiments, when NDCG@k is used as the evaluation metric, we use the corresponding truncated LambdaLoss.

6.3 Competing Methods

Our methods are implemented based on both the RankLib library[1] and the LightGBM library [20] to allow for reproducibility. RankLib implemented a list of the most popular and representative algorithms such as RankNet [3], Multiple Additive Regression Trees (MART) [13], and LambdaMART [4]. We compared all of these baselines and found that LambdaMART performs the best. So we only report our comparison with the LambdaMART implementation in RankLib. The LightGBM library [20] is a more recently released package that focuses on Gradient Boosting methods. It provides a new implementation of LambdaMART based on more sophisticated boosting techniques and achieves the state-of-the-art performance on multiple public benchmark datasets. We compare our methods with the LambdaMART implementation in LightGBM.

For our LambdaLoss methods, we implemented them in both RankLib and LightGBM. The loss functions we proposed are general, and can work with any ranking models (e.g., neural networks or tree-based models). In this paper, we focus our study on tree-based methods. In addition to ARP-Loss1, ARP-Loss2, NDCG-Loss1, and NDCG-Loss2, we also study a hybrid model that linearly combines ρ_{ij} in LambdaRank and δ_{ij} in NDCG-Loss2:

$$\rho_{ij} + \mu\delta_{ij}. \tag{19}$$

where μ is the the weight coefficient on NDCG-Loss2 and we name such a loss NDCG-Loss2++.

[1] https://sourceforge.net/p/lemur/wiki/RankLib/

Table 1: Statistics of the three data sets used in our experiments.

Data sets	YAHOO			WEB10K			WEB30K		
	Train	Valid.	Test	Train	Valid.	Test	Train	Valid.	Test
Queries	19,944	2,994	6,983	6,000	2,000	2,000	18,919	6,306	6,306
Docs	473,134	71,083	165,660	723,412	235,259	241,521	2,270,296	747,218	753,611
Features	700			136			136		

Table 2: Hyper-parameter settings of representative methods in LightGBM based on validation where leaves is num-leaves, hessian is the min-sum-hessian-in-leaf, and mindata is min-data-in-leaf respectively.

	Methods	leaves	hessian	mindata
YAHOO	LambdaMART	200	10	100
	NDCG-Loss2	100	50	5
	NDCG-Loss2++	255	10	5
WEB10K	LambdaMART	255	50	50
	NDCG-Loss2	20	100	20
	NDCG-Loss2++	100	1	5
WEB30K	LambdaMART	200	50	50
	NDCG-Loss2	50	100	50
	NDCG-Loss2++	255	50	20

7 EXPERIMENTAL RESULTS

We compare different learning-to-rank methods and report our experimental results in this section.

7.1 Overall Comparison

Table 3 shows the results of comparing our methods against LambdaMART in RankLib. We tune the number of trees on the validation set while using the default settings for other hyper-parameters (we will show the results with tuned hyper-parameters in LightGBM). In this table, our proposed methods NDCG-Loss2 and NDCG-Loss2++ achieve the best results. Compared with LambdaMART, both methods achieve significant improvement over all the Train, Validation and Test data sets. For example, NDCG-Loss2 and NDCG-Loss2++ achieve 1.86% and 1.58% over LambdaMART on the WEB30K Test data set. Furthermore, NDCG-Loss2 is better than NDCG-Loss1 across all the data sets, confirming that NDCG-Loss2 is a better bound than NDCG-Loss1 for the NDCG metric. NDCG-Loss2 is also better than ARP-Loss1 and ARP-Loss2, but NDCG-Loss1 is not. This shows it is beneficial to have NDCG-driven loss functions but the benefit can be realized only with a proper bound. Though ARP-Loss1 is derived from ARP, it is very competitive and this shows that there is a good correlation between the ARP and NDCG metrics.

Table 4 shows the comparison in LightGBM. In this comparison, the hyper-parameters specific to each learning-to-rank algorithm are tuned to optimal on the validation sets using NDCG@5 as the metric. The most important hyper-parameters used are shown in Table 2. The default early stopping is applied when there is no improvement on validation set for consecutive 30 iterations. Our results in Table 4 are comparable with results reported in early work [20]. For NDCG-Loss2++, we set the weight μ to 5. From this table, we can see that NDCG-Loss2++ is significantly better than the LambdaMART in LightGBM, the state-of-the-art implementation, on both YAHOO and WEB30K data sets. This shows that our LambdaLoss can improve over the state-of-the-art learning-to-rank algorithms.

However, we also observe that the NDCG-Loss2 tuned using validation data set does not perform well in LightGBM. This is because NDCG-Loss2 can easily overfit the training data in the LightGBM setting. As shown in Table 2, the number of leaves per tree for NDCG-Loss2 is much smaller than that of the LambdaMART baseline (e.g., 50 vs 200 on WEB30K). We verify the overfitting by using the optimal hyper-parameter of LambdaMART on NDCG-Loss2. The results on the Train data sets are shown in Table 5 and we can see severe overfitting of NDCG-Loss2. On one hand, this could mean that NDCG-Loss2 is a tighter bound for NDCG than LambdaMART. On the other hand, similar to the observation in [33], a tighter bound can easily lead to overfitting without proper regularization. Our proposed NDCG-Loss2++ regularize NDCG-Loss2 using LambdaMART formulation and achieve the best results. This result is inspiring in designing regularization techniques when directly optimizing ranking metrics.

The results for LightGBM is much better than the results in RankLib, part of the reason is that LightGBM uses much complex trees with a large number of leaves per tree. When we limit the number of leaves to 10, the default value of RankLib, in LightGBM, we observe sightly better results for LightGBM but the comparison among methods is similar to RankLib: NDCG-Loss2 outperforms LambdaMART. We omit such results due to space limitation.

7.2 Parameter Study

7.2.1 Parameter μ. We study the sensitivity of parameter μ in Eq. 19 by varying it between $[0.1, 10]$, which controls the importance of NDCG-Loss2 in the hybrid model NDCG-Loss2++. Due to space limitation, we show the results only on the WEB30K Test data set, as similar trends are observed on other data sets. Figure 3 shows the NDCG@5 metric in both the RankLib and LightGBM implementations of NDCG-Loss2++. Results from both implementations are relatively stable, meaning that the hybrid model is not very sensitive to the change of μ. On the other hand, NDCG-Loss2++ performs better when μ is larger, suggesting that NDCG-Loss2 plays a more important role than LambdaRank.

7.2.2 Parameter k in NDCG@k. Figure 4 shows the results of NDCG@k with k selected from $\{1, 3, 5, 10\}$ on the WEB30K test data set in RankLib. Our proposed methods can outperform LambdaMART on all the k values and this shows the flexibility of designing LambdaLoss to optimize different metrics.

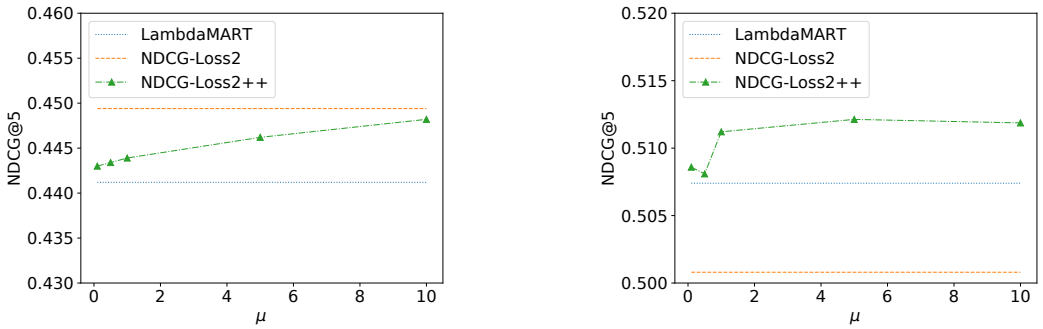
7.2.3 Convergence. Figure 5 shows the NDCG@5 values of NDCG-Loss2, NDCG-Loss2++, and LambdaMART methods along the EM

Table 3: Overall comparison in RankLib based on NDCG@5 in percentage. * indicates that a method is statistically significantly better than LambdaMART, according to t-test at level of 0.05. Boldface is the best column wise.

Data sets	YAHOO			WEB10K			WEB30K		
	Train	Valid.	Test	Train	Valid.	Test	Train	Valid.	Test
LambdaMART	72.21	69.07	70.36	44.99	43.40	42.19	45.28	43.81	44.12
ARP-Loss1	**74.11***	**69.70***	70.58	47.24*	44.01*	42.88*	47.02*	44.59*	44.69*
ARP-Loss2	71.74	69.24	70.38	46.06*	43.69	42.70	45.25	43.86	44.13
NDCG-Loss1	71.45	68.95	70.26	45.49*	43.80	42.56	45.03	43.60	44.04
NDCG-Loss2	73.07*	69.57*	70.63*	47.33*	43.98*	**43.19***	**47.35***	**44.71***	**44.94***
NDCG-Loss2++	73.65*	69.41*	**70.72***	**48.58***	**44.03***	42.99*	46.89*	44.48*	44.82*

Table 4: Overall comparison in LightGBM based on NDCG@5 in percentage. * indicates that a method is statistically significantly better than LambdaMART, according to t-test at level of 0.05. Boldface is the best column wise.

Data sets	YAHOO			WEB10K			WEB30K		
	Train	Valid.	Test	Train	Valid.	Test	Train	Valid.	Test
LambdaMART	91.84	74.93	75.40	**70.24**	49.05	49.00	64.23	50.23	50.74
NDCG-Loss2	77.96	73.77	74.78	54.75	**49.40**	49.22	52.60	49.82	50.08
NDCG-Loss2++	**92.82***	**75.00***	**75.70***	62.75	49.03	**49.25**	**66.78***	**50.97***	**51.21***



(a) RankLib implementation.

(b) LightGBM implementation.

Figure 3: Impact of μ on WEB30K.

Table 5: Overfitting on Training data in LightGBM.

Data sets	YAHOO	WEB10K	WEB30K
LambdaMART	91.84	70.24	64.23
NDCG-Loss2	92.62	88.09	98.74

iterations. They all show convergence as the number of trees increases and this empirically verifies the correctness of our EM algorithm in optimizing LambdaLoss.

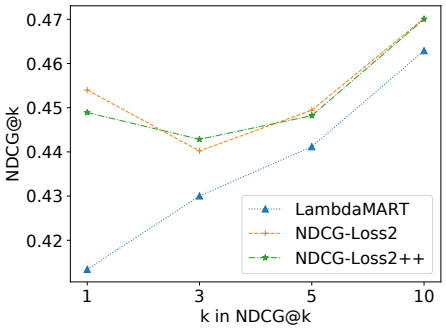


Figure 4: NDCG@k along with k on WEB30K.

8 CONCLUSIONS

In this paper, we presented the LambdaLoss framework for ranking metric optimization. Our framework is motivated by the LambdaRank algorithms that incorporate ranking metrics in their learning procedures. We showed that LambdaRank can be formulated as a special configuration in our LambdaLoss framework, and can be theoretically justified by a novel loss function. Furthermore, we introduced several other metric-driven loss functions in our framework that depend on both ranks and scores. We empirically validated the proposed framework on three publicly available learning-to-rank data sets, and demonstrated that our framework can significantly improve the state-of-the-art learning-to-rank algorithms.

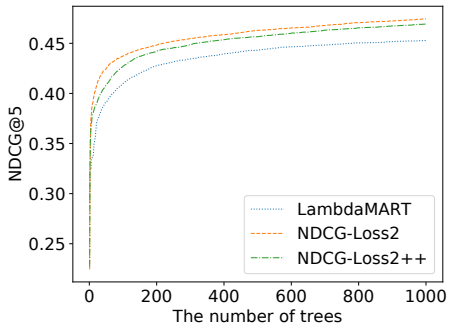


Figure 5: Convergence of the EM procedure in LambdaLoss.

Our work opens up several interesting research directions for future work. (1) While we mainly focus on NDCG in this paper, it would be interesting to extend it to other popular ranking metrics such as MAP. (2) We mainly worked with the hard assignment of $H(\pi|\mathbf{s})$ in this paper; the effect of the soft assignment setting in LambdaLoss can be explored. (3) We implemented our framework on tree-based models; other models such as deep neural networks can be studied in the future. (4) Our results show that tighter bound for metrics may lead to overfitting and thus developing regularization techniques in LambdaLoss is another interesting direction.

REFERENCES

[1] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc.

[2] Ralph A. Bradley and Milton E. Terry. 1952. The Rank Analysis of Incomplete Block Designs — I. The Method of Paired Comparisons. *Biometrika* 39 (1952), 324–345.

[3] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proc. of the 22nd International Conference on Machine Learning (ICML)*. 89–96.

[4] Christopher J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report MSR-TR-2010-82. Microsoft Research.

[5] Christopher J. C. Burges, Robert Ragno, and Quoc Viet Le. 2006. Learning to Rank with Nonsmooth Cost Functions. In *Proc. of the 20th Annual Conference on Neural Information Processing Systems (NIPS)*. 193–200.

[6] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proc. of the 24th International Conference on Machine Learning (ICML)*. 129–136.

[7] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge*. 1–24.

[8] Olivier Chapelle and Mingrui Wu. 2010. Gradient descent optimization of smoothed information retrieval metrics. *Inf. Retr.* 13, 3 (2010), 216–235.

[9] Wei Chu and Zoubin Ghahramani. 2005. Preference learning with Gaussian processes. In *Proc. of the 22nd International Conference on Machine Learning (ICML)*. 137–144.

[10] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B (Methodological)* 39, 1 (1977), 1–38.

[11] Pinar Donmez, Krysta M. Svore, and Christopher J.C. Burges. 2009. On the Local Optimality of LambdaRank. In *Proc. of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 460–467.

[12] Yoav Freund and Robert E. Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. Syst. Sci.* 55, 1 (1997), 119–139.

[13] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of Statistics* 29, 5 (2001), 1189–1232.

[14] Norbert Fuhr. 1989. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems (TOIS)* 7, 3 (1989), 183–204.

[15] Fredric C Gey. 1994. Inferring probability of relevance using the method of logistic regression. In *Proc. of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 222–231.

[16] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.

[17] Thorsten Joachims. 2002. Optimizing Search Engines Using Clickthrough Data. In *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 133–142.

[18] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *Proc. of the 10th ACM International Conference on Web Search and Data Mining (WSDM)*. 781–789.

[19] N.L. Johnson, S. Kotz, and N. Balakrishnan. 1995. *Continuous univariate distributions.* Number v. 2 in Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley & Sons.

[20] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*. 3149–3157.

[21] Quoc V. Le and Alexander J. Smola. 2007. Direct Optimization of Ranking Measures. *CoRR* abs/0704.3359 (2007).

[22] Ping Li, Christopher J. C. Burges, and Qiang Wu. 2007. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In *Proc. of the 21st Annual Conference on Neural Information Processing Systems (NIPS)*. 897–904.

[23] Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval.* Springer.

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

[25] Pritish Mohapatra, Michal Rolinek, C. V. Jawahar, Vladimir Kolmogorov, and M. Pawan Kumar. 2018. Efficient Optimization for Rank-based Loss Functions. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[26] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013).

[27] Tao Qin, Tie-Yan Liu, and Hang Li. 2010. A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval* 13, 4 (2010), 375–397.

[28] Christian P. Robert and George Casella. 2005. *Monte Carlo Statistical Methods (Springer Texts in Statistics).* Springer-Verlag.

[29] Zhengya Sun, Tao Qin, Qing Tao, and Jue Wang. 2009. Robust Sparse Rank Learning for Non-smooth Ranking Measures. In *Proc. of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 259–266.

[30] Martin Szummer and Emine Yilmaz. 2011. Semi-supervised Learning to Rank with Preference Regularization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM)*. 269–278.

[31] Ming Tan, Tian Xia, Lily Guo, and Shaojun Wang. 2013. Direct Optimization of Ranking Measures for Learning to Rank Models. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. 856–864.

[32] Niek Tax, Sander Bockting, and Djoerd Hiemstra. 2015. A cross-benchmark comparison of 87 learning to rank methods. *Information Processing & Management* 51, 6 (2015), 757–772.

[33] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. Softrank: optimizing non-smooth rank metrics. In *Proc. of the 2008 International Conference on Web Search and Data Mining (WSDM)*. 77–86.

[34] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. 2006. Optimisation methods for ranking functions with multiple parameters. In *Proc. of the 15th ACM International Conference on Information and Knowledge Management (CIKM)*. 585–593.

[35] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support Vector Machine Learning for Interdependent and Structured Output Spaces. In *Proc. of the 21st International Conference on Machine Learning (ICML)*. 104.

[36] Hamed Valizadegan, Rong Jin, Ruofei Zhang, and Jianchang Mao. 2009. Learning to Rank by Optimizing NDCG Measure. In *Proc. of the 22nd International Conference on Neural Information Processing Systems (NIPS)*. 1883–1891.

[37] Ellen M. Voorhees and Donna K. Harman (Eds.). 1999. *Proc. of The Eighth Text REtrieval Conference, TREC.* Vol. Special Publication 500-246. National Institute of Standards and Technology (NIST).

[38] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. In *Proc. of the 11th Conference on Web Search and Data Mining (WSDM)*. 610–618.

[39] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise Approach to Learning to Rank: Theory and Algorithm. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. 1192–1199.

[40] Jun Xu and Hang Li. 2007. AdaRank: A Boosting Algorithm for Information Retrieval. In *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 391–398.

[41] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. 2007. A support vector method for optimizing average precision. In *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 271–278.