Which Tricks are Important for Learning to Rank?

Ivan Lyzhin [1]   **Aleksei Ustimenko** [2][†]   **Andrey Gulin** [1]   **Liudmila Prokhorenkova** [3]

Abstract

Nowadays, state-of-the-art learning-to-rank methods are based on gradient-boosted decision trees (GBDT). The most well-known algorithm is LambdaMART which was proposed more than a decade ago. Recently, several other GBDT-based ranking algorithms were proposed. In this paper, we thoroughly analyze these methods in a unified setup. In particular, we address the following questions. Is direct optimization of a smoothed ranking loss preferable over optimizing a convex surrogate? How to properly construct and smooth surrogate ranking losses? To address these questions, we compare LambdaMART with YetiRank and StochasticRank methods and their modifications. We also propose a simple improvement of the YetiRank approach that allows for optimizing specific ranking loss functions. As a result, we gain insights into learning-to-rank techniques and obtain a new state-of-the-art algorithm.

1. Introduction

Learning to rank (LTR) is a central problem in information retrieval (Burges, 2010). The objective of LTR is to rank a given set of items to optimize the overall utility of the list. In information retrieval, given a query and a list of candidate documents, it is desirable to rank the documents in decreasing order of *relevance* to the query. Usually, LTR methods learn a function computing *scores* for all documents and then sort the documents according to their scores. Thus, a major challenge for LTR is that the sorting operation is non-differentiable, which prevents effective gradient-based optimization.

Despite the recent success of neural approaches in vari-

ous machine learning tasks, gradient-boosted decision trees (GBDT) are still state-of-the-art algorithms for *tabular* datasets containing heterogeneous and noisy features (Gorishniy et al., 2021; Katzir et al., 2021). In particular, GBDT methods outperform other approaches for the learning-to-rank problem. A recent paper by Qin et al. (2021) observes that for standard tabular LTR datasets, the best results are achieved by a classic LambdaMART algorithm (Wu et al., 2010) proposed more than a decade ago and implemented within the LightGBM library (Ke et al., 2017). Thus, our paper focuses on GBDT-based learning-to-rank methods and provides an extensive study of existing and new approaches and their important algorithmic details.

In addition to LambdaMART, we revisit another long-known algorithm called YetiRank (Gulin et al., 2011). YetiRank won The Transfer Learning Track of Yahoo!'s LTR challenge in 2010 but was not much explored after that. The main algorithmic differences of YetiRank compared to LambdaMART are stochastic smoothing used during training to randomize the orderings and a different way to construct an upper bound for the loss function with a tighter approximation. Our empirical evaluation shows that YetiRank outperforms LambdaMART and other competitors in most cases. In addition, we show that YetiRank implicitly optimizes a particular form of the DCG ranking quality function. This observation allows us to modify the YetiRank procedure by re-weighting the gradients according to a proper loss. The proposed modification YetiLoss further improves the performance of YetiRank for specific losses, for instance, MRR and MAP. For MAP, we obtain new state-of-the-art results with YetiLoss. We also explain why YetiLoss is expected to have more stable optimization and better generalization.

A recent work by Ustimenko & Prokhorenkova (2020) shows that the LambdaMART approach, implemented within the CatBoost library (Prokhorenkova et al., 2018), can be outperformed by an algorithm called StochasticRank. StochasticRank directly optimizes any given ranking loss with provable convergence guarantees. For this purpose, it uses stochastic smoothing of the loss function (without upper bounding). The obtained objective is often non-convex, so it is optimized with Stochastic Gradient Langevin Boosting (Ustimenko & Prokhorenkova, 2020) — a boosting algorithm designed to optimize non-convex functions.

[†]The work was done while working at Yandex. [1]Yandex, Moscow, Russia [2]ShareChat, London, UK [3]Yandex Research, Amsterdam, The Netherlands. Correspondence to: Aleksei Ustimenko <research@aleksei.uk>, Liudmila Prokhorenkova <ostroumova-la@yandex-team.ru>.

The main difference between StochasticRank and YetiLoss is that the former optimizes the training loss directly, while the latter constructs special convex bounds on the loss function at each iteration. While minimizing such bounds cannot guarantee reaching the global optima, convexity allows for more efficient optimization, which can often be beneficial. We provide extensive experiments to validate in which scenarios direct optimization is helpful.

To sum up, our main contributions are the following.

- We conduct a thorough analysis of several state-of-the-art GBDT-based LTR algorithms. We provide a theoretical explanation of their differences and extensive empirical evaluation.

- We analyze the effect of several algorithmic details: optimizing the loss function directly versus constructing a convex upper bound, adding stochastic smoothing to the loss function, and different ways of constructing an upper bound.

- We propose a simple extension of the YetiRank algorithm that handles arbitrary loss functions. The obtained algorithm achieves new state-of-the-art results for such loss functions as MRR and MAP that are often overlooked in LTR literature.

In the next section, we give the necessary background: define the learning-to-rank problem and popular quality functions, and then describe several well-known ranking algorithms. In Section 3, we propose our modification of YetiRank that allows for optimizing any given ranking loss and also analyze and compare the properties of all the considered algorithms. In Section 4, we conduct a thorough comparison of existing LTR algorithms on several benchmarks, show that YetiLoss outperforms the competitors for specific ranking quality functions, and analyze the effect of the main algorithmic details on the quality of LTR. Section 5 concludes the paper.

2. Background and Related Work

2.1. Learning-to-rank Problem

Learning to rank is a classic information retrieval problem (Liu, 2009). To formally define the LTR task, consider a query q sampled from a distribution Q. For this query, we are given a set of documents $\{x_1, \ldots, x_{n_q}\}$ and a relevance vector $r = (r_1, \ldots, r_{n_q}) \in \mathbb{R}^{n_q}$. Each document x_i is represented as a vector of features that describes the query-document pair.

A learning-to-rank model is a function $f(x)$ that takes the document's features x_i and returns a value z_i that is the predicted relevance for this document. For a query q, we

compute the vector $z = (f(x_1), \ldots, f(x_{n_q})) \in \mathbb{R}^{n_q}$ called *a vector of scores* and then compute $s = \arg \operatorname{sort}(z) \in \mathbb{R}^{n_q}$ that is the ranking of documents predicted by the model $f(x)$.

A ranking loss function $L(z, r)$ is a function that measures how well the ranking $s = \arg \operatorname{sort}(z)$ agrees with the relevance vector r. To simplify the notation, we also write $L(f, q)$ meaning that $\{x_1, \ldots, x_{n_q}\}$ and r are contained in q and $z = (f(x_1), \ldots, f(x_{n_q}))$ is computed internally.

The ultimate goal of LTR is to find a model $f(x) \in \mathcal{F}$ ($\mathcal{F}$ is a predefined class of models) such that:

$$f = \arg \min_{f \in \mathcal{F}} \mathbb{E}_{q \sim Q} L(f, q). \tag{1}$$

Unfortunately, in practice, we do not know the distribution Q but rather have an i.i.d. sample $q_1, \ldots, q_N \sim Q$ which we call the *training* dataset. Thus, instead of (1), we solve:

$$f = \arg \min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} L(f, q_i). \tag{2}$$

The function $\mathcal{L}_N(f) := \frac{1}{N} \sum_{i=1}^{N} L(f, q_i)$ is called an *empirical loss function*.

The main difficulty of the LTR problem is the fact that the function $\mathcal{L}_N(f)$ is locally constant as a function of z, discontinuous, and non-convex in z, since each $L(z, r)$ is defined by a permutation $s = \arg \operatorname{sort}(z)$. As a result, $\mathcal{L}_N(f)$ does not have a well-defined derivative (derivatives of $L(z, r)$ are everywhere zero due to a local constancy). Thus, one cannot directly optimize this loss with standard methods like gradient boosting or backpropagation in neural networks.

2.2. Ranking Loss Functions

In this section, we define some widely used ranking loss functions. First, recall that all ranking losses depend on the permutation $s = \arg \operatorname{sort}(z)$. However, it can be the case that two documents x_i and x_j have the same score, i.e., $z_i = z_j$. This situation is called a *tie*, and in this case, $\arg \operatorname{sort}(z)$ is not well-defined. Ties may occur due to the model's discrete structure (e.g., if f is a GBDT model) or when two different documents have the same feature vectors. Although ties are often neglected in the LTR literature, they can affect the optimization procedure and lead to an overestimated ranking quality (Ustimenko & Prokhorenkova, 2020). Therefore, Ustimenko & Prokhorenkova (2020) propose resolving the ties as $s = \arg \operatorname{sort}((z, -r))$, meaning that the documents are ordered according to z, but in the case of ties, we order them accordingly to $-r$, i.e., put the less relevant document first. In other words, the *worst* permutation is used for ties. In this paper, we follow this tie-resolution policy.

Let $s = \arg\text{sort}((z, -r))$ and let $n = n_q$ be the length of the vectors z and r. Probably the most well-known ranking quality function is DCG@k:

$$\text{DCG@}k(z, r) = \sum_{i=1}^{\min\{n,k\}} \frac{2^{r_{s_i}} - 1}{2^4 \log_2(i+1)}, \qquad (3)$$

where $r_i \in [0, 4]$ are relevance labels. This quality function is called *Discounted Cumulative Gain*: we divide the *gain* for the relevance by the *discount* for a lower position. Different options for *gain* and *discount* functions are possible, and (3) is the most widely used combination.

NDCG@k is a normalized variant of DCG@k:

$$\text{NDCG@}k(z, r) = \frac{\text{DCG@}k(z, r)}{\max_{z' \in \mathbb{R}^n} \text{DCG@}k(z', r)}.$$

Usually, LTR papers report the values of NDCG@k, while other measures can be of interest in practice. For instance, for binary relevance labels $r_j \in \{0, 1\}$, one can be interested in *Average Precision* (AP):

$$\text{AP}(z, r) = \frac{\sum_{i=1}^n \text{P}(i)\, r_{s_i}}{\sum_{i=1}^n r_i}, \quad \text{where } \text{P}(i) = \frac{1}{i} \sum_{j=1}^i r_{s_j}.$$

Mean Average Precision (MAP) is AP averaged over the queries. Another measure used for binary labels is *Reciprocal Rank* (called MRR after the averaging):

$$\text{RR}(z, r) = \sum_{i=1}^n \frac{r_{s_i}}{i} \prod_{j=1}^{i-1} (1 - r_{s_j}), \quad r_j \in \{0, 1\},$$

which is the inverse rank of the first relevant document. Finally, *Expected Reciprocal Rank* (ERR) is a similar measure applied to non-binary labels:

$$\text{ERR}(z, r) = \sum_{i=1}^n \frac{r_{s_i}}{i} \prod_{j=1}^{i-1} (1 - r_{s_j}), \quad r_j \in [0, 1].$$

The above examples are ranking *quality* functions, i.e., higher values are preferred. When we need to convert a quality function $M(z, r)$ to a loss function, we can just consider $L(z, r) := 1 - M(z, r)$. Note that in our experiments, we report ranking quality values multiplied by 100.

2.3. Ranking Algorithms

This section describes LTR algorithms that are of particular interest to the current research. For more methods and a broader overview, we refer to Liu (2009); Ustimenko & Prokhorenkova (2020); Qin et al. (2021).

As discussed in Section 2.1, the ranking loss function $\mathcal{L}_N(f)$ is non-differentiable. Thus, if we want to apply a convenient

optimization method like gradient boosting or backpropagation, we need to build a differentiable surrogate. One of the most popular approaches is to use the Majorization-Minimization technique, when at each iteration t we build a function $l_t(z, r)$ that is usually convex and differentiable such that:

$$L(z, r) \leq l_t(z, r) \quad \forall z \in \mathbb{R}^{n_q}. \qquad (4)$$

After that, we sum up the upper bounds (4) for all the queries and obtain:[1]

$$\mathcal{L}_N(f) \leq \frac{1}{N} \sum_{i=1}^N l_t(f, q_i) = \mathcal{L}_t^*(f). \qquad (5)$$

The loss function $\mathcal{L}_t^*(f)$ can be optimized directly by gradient boosting or backpropagation.

Many popular learning-to-rank algorithms like Lambda-Rank (Burges et al., 2007), LambdaMART (Wu et al., 2010), or LambdaLoss (Wang et al., 2018) fall into this category. In this section, we mostly follow the reasoning of Wang et al. (2018).

To get a differentiable upper bound, these algorithms first build a bound of the form:

$$L(z, r) \leq \text{const} + \sum_{i,j=1}^{n_q} w_{ij} \cdot \mathbb{1}_{z_j - z_i > 0}. \qquad (6)$$

After that, one can use the inequality $\mathbb{1}_{z_j - z_i > 0} \leq \log_2\left(1 + e^{-(z_i - z_j)}\right)$ to obtain a convex differentiable bound:

$$L(z, r) \leq \text{const} + \sum_{i,j=1}^{n_q} w_{ij} \log_2\left(1 + e^{-(z_i - z_j)}\right). \qquad (7)$$

To be more precise, for NDCG@k LambdaRank and LambdaMART select the weights w_{ij} as:

$$w_{ij} = \Delta_{ij}\text{NDCG@}k(z, r) \cdot \mathbb{1}_{r_i > r_j}, \qquad (8)$$

where $\Delta_{ij}\text{NDCG@}k(z, r)$ denotes the difference in NDCG@$k(z, r)$ caused by permuting i-th and j-th documents in the sorting defined by z. Formally, recall that $s = \arg\text{sort}(z)$ and consider the inverse permutation $p = s^{-1}$. The value p_i is the position of the i-th document in the list ordered by z. Then,

$$\Delta_{ij}\text{NDCG@}k(z, r) = \frac{(2^{r_i} - 2^{r_j})}{2^4 \max_{z'} \text{DCG@}k(z', r)}$$
$$\cdot \left| \frac{1}{\log_2(1 + p_j)} - \frac{1}{\log_2(1 + p_i)} \right|. \qquad (9)$$

---

[1] As before, we denote by $l_t(f, q)$ the values of $l_t(z, r)$, where we set $z = (f(x_1), \ldots, f(x_{n_q}))$.

As proven by Wang et al. (2018), using such w_{ij} indeed gives a desired upper bound. Similarly, one can adapt LambdaMART to any ranking metric by substituting $\Delta_{ij}\text{NDCG}@k(z, r)$ in (8) by the difference of the target loss function. It is important to note that the *upper bound property may not hold* in this case. However, empirical evidence shows that this trick works well in practice.

Now let us describe the YetiRank algorithm. The original formulation given by Gulin et al. (2011) does not specify which ranking metric is optimized, but the pairwise loss is similar to the ones described above. YetiRank uses the following convex differentiable loss:

$$l_t(z, r) = - \sum_{i,j=1}^{n_q} w_{ij} \log \left(1 + e^{-(z_i - z_j)} \right) , \qquad (10)$$

where the weights are chosen as:

$$w_{ij} = (r_i - r_j) \cdot \mathbb{1}_{r_i > r_j} \cdot \mathbb{E}_\epsilon [b^{p_i - 1} \cdot \mathbb{1}_{|p_i - p_j| = 1}] , \quad (11)$$

where $b \in (0, 1)$ is a hyper-parameter and p_i is a random permutation obtained as the inverse of $\arg\text{sort}(z + \epsilon)$ with $\epsilon = \log \frac{u}{1-u}, u \sim U([0, 1]^{n_q})$, i.e., ϵ is distributed according to the Logistic distribution. In practice, to estimate w_{ij}, one adds random noise to the current scores to generate several random permutations. By default, ten permutations are used.[2] Importantly, for YetiRank, we can obtain only stochastic gradient estimates of $\nabla_z l_t(z, r)$ due to the randomness of ϵ. We provide a deeper analysis of YetiRank in Section 3.

The CatBoost library also has a modification of YetiRank called YetiRankPairwise (CatBoost, 2022), which is described in Gulin et al. (2011). It is a GBDT-specific modification of YetiRank, allowing one to get more accurate results on large datasets. However, in our preliminary experiments, we have not observed consistent improvements of YetiRankPairwise over YetiRank. Since YetiRankPairwise is significantly slower than all other algorithms considered in this paper, we do not include it in our empirical analysis.

Finally, we overview the recently proposed StochasticRank algorithm (Ustimenko & Prokhorenkova, 2020). In sharp contrast with the previous approaches that build convex differentiable surrogates to optimize a given non-convex ranking loss, StochasticRank smooths $L(z, r)$ directly using the Gaussian distribution $\mathcal{N}(-\mu r, \sigma I_{n_q})$:

$$l(z, r) = \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I_{n_q})} L(z - \mu r + \sigma \epsilon, r) , \quad (12)$$

where $\mu \geq 0$ and $\sigma > 0$ are hyper-parameters. Such smoothing leads to the loss function $\mathcal{L}_N(f, \sigma, \mu) =$

---

[2]The complexity of computing the gradients is thus increased with the number of permutations, but this does not affect the overall time complexity much since the most time-consuming operation is building a decision tree, and this procedure is not affected.

$\frac{1}{N} \sum_{i=1}^N l(f, q_i)$ which minimum point is arbitrary close to the minimum of $\mathcal{L}_N(f)$, see Ustimenko & Prokhorenkova (2020) for the details. Moreover, the paper proposes novel stochastic gradient estimates to estimate the gradients more effectively compared to the log-derivative trick (Nesterov & Spokoiny, 2017). Finally, StocasticRank relies on Stochastic Gradient Langevin Boosting (SGLB) (Ustimenko & Prokhorenkova, 2021) that guarantees that the optimization would eventually reach the global optimum even for the non-convex loss function $\mathcal{L}_N(f, \mu, \sigma)$.

3. Analysis of LTR Algorithms

In Section 4, we will show that YetiRank achieves state-of-the-art results for most datasets and quality functions. However, it is not specifically optimized for any of these metrics. In this section, we first discuss which loss function is optimized by YetiRank and then propose a simple modification called YetiLoss that can optimize any given ranking quality function. Finally, we discuss how YetiLoss relates to LambdaMART and StochasticRank.

3.1. YetiLoss Algorithm

Let us consider the following ranking quality function for a fixed $b \in (0, 1)$:

$$\text{ExpDCG}(z, r) = \sum_{i=1}^{n_q} b^{i-1} r_{s_i} , \qquad (13)$$

where $s = \arg\text{sort}(z)$. This loss resembles the classic DCG given in (3), but the gain is r_{s_i} instead of $2^{r_{s_i}} - 1$, while the discount is exponential (b^{i-1}) instead of logarithmic ($\log_2(i + 1)$).

It follows from (11) and (13) that up to a constant multiplier, we have

$$w_{ij} = \mathbb{1}_{r_i > r_j} \cdot \mathbb{E}_\epsilon[\Delta_{ij}\text{ExpDCG}(z + \epsilon, r) \cdot \mathbb{1}_{|p_i - p_j| = 1}] ,$$

which means that YetiRank optimizes ExpDCG but in a smoothed form.

This observation suggests that to adapt YetiRank for an arbitrary loss function, we need to replace $\Delta_{ij}\text{ExpDCG}$ by the difference in the desired ranking loss. For $\text{NDCG}@k$, the expression is given in (9). For MRR, we have:

$$\Delta_{ij}\text{MRR}(z, r) = \left| \frac{1}{p_i} - \frac{1}{p_j} \right| \cdot \prod_{k=1}^{\min\{i,j\}-1} (1 - r_{s_j}) \cdot \mathbb{1}_{r_i = 1 - r_j} .$$

Analogously, we obtain the expressions for ERR and MAP. Similarly to LambdaMART, for an arbitrary metric (rather than NDCG), we do not have the upper bound guarantees. However, our experiments show that YetiLoss outperforms standard YetiRank for the MRR and MAP and in half of the cases for ERR.

3.2. Comparison of YetiLoss and LambdaMART

For a given ranking loss function $M(z, r)$, LambdaMART and YetiLoss use the following weights:

$$w_{ij}^{LM} = \Delta_{ij} M(z, r) \cdot \mathbb{1}_{r_i > r_j} \,,$$

$$w_{ij}^{YL} = \mathbb{E}_\epsilon \Delta_{ij} M(z + \epsilon, r) \cdot \mathbb{1}_{r_i > r_j} \cdot \mathbb{1}_{|p_i - p_j| = 1} \,.$$

Even though both weights share the term $\Delta_{ij} M \cdot \mathbb{1}_{r_i > r_j}$, they have principal differences, which we discuss below.

First, let us note that w^{LM} is a discontinuous function of the argument z (model predictions). Hence, it may drastically change when the model f changes only slightly. In contrast, w^{YL} is a smooth function of z due to smoothing with an absolutely continuous distribution. Training with a smooth objective is expected to be more robust; thus, we expect YetiLoss to have a better generalization.

Let us also note that a similar smoothing is proposed in a recent paper by Bruch et al. (2020). The authors use a different smoothing distribution and also add noise into $f_i - f_j$ that appears in the logarithmic term that is multiplied by the weights. The authors confirm that smoothing the loss function improves the performance. While comparing with Bruch et al. (2020) is out of the scope of the current paper since we focus on widely used open-source implementations, we do empirically evaluate the importance of smoothing and different smoothing distributions in Section 4.3.

The second difference is that w^{YL} contains Δ_{ij} only for the neighboring documents with $|p_i - p_j| = 1$. In other words, YetiLoss uses only those pairwise permutations that *can be realized* by small changes in z. In contrast, w^{LM} does not have this restriction and contains the pairwise permutations that *cannot be realized* by small changes of z. This implies that for LambdaMART, the resulting gradients may contain redundant information about the permutations that cannot be achieved within one gradient-boosting step with a small step size. Hence, the obtained upper bound on the ranking loss can be too loose. In Section 4.3, we empirically evaluate the effect of this difference.

3.3. Comparison of YetiLoss and StochasticRank

The approach underlying the StochasticRank algorithm significantly differs from both YetiLoss and LambdaMART (see Section 2.3). However, it turns out that StochasticRank has a structural similarity with YetiLoss but not with LambdaMART. This similarity lies in pairwise permutations that YetiLoss incorporates. We already mentioned that LambdaMART relies on all possible pairwise permutations to build the surrogate loss $\mathcal{L}_t^*$. In contrast, YetiLoss uses only those that permute the neighborhood elements in the sorted list. Similarly, we can show that StochasticRank also

relies only on such permutations. To see that, we write down the formula for $\frac{\partial}{\partial z_i} l(z, r, \sigma, \mu)$ estimate that is introduced by Ustimenko & Prokhorenkova (2020) and is called Coordinate Conditional Sampling (CCS). Let us assume for simplicity that $\mu = 0$:

$$\frac{\partial^{CCS}}{\partial z_i} l(z, r, \sigma) = (2\pi\sigma^2)^{-\frac{1}{2}}$$
$$\cdot \sum_{j=0}^{n_q} \delta_{i,j} L(z + \epsilon, r) e^{-\frac{1}{2\sigma^2}(z_i - z_j + \sigma(\epsilon_i - \epsilon_j))^2} \,,$$

where δ_{ij} corresponds to the difference of the ranking loss if we change only z_i without changing the remaining values, so that the i-th item is placed after the j-th item for $j > 0$. For $j = 0$, it means the difference if we put the i-th item on the first position.

Next, if we assume that all z_i are different, the properties of the super-exponential decay of e^{-y^2} allow us to ensure that for small enough $\sigma \approx 0$ the latter can be approximated as:

$$\frac{\partial^{CCS}}{\partial z_i} l(z, r, \sigma)$$
$$\approx (2\pi\sigma^2)^{-\frac{1}{2}} \cdot \Big(\delta_{i,a} L(z + \epsilon, r) e^{-\frac{1}{2\sigma^2}(z_i - z_a)^2}$$
$$+ \delta_{i,b} L(z + \epsilon, r) e^{-\frac{1}{2\sigma^2}(z_i - z_b)^2}\Big) \,,$$

where a is the index of the largest z_a that is smaller than z_i, and b is the index of the smallest z_b that is larger than z_i, i.e., *that is exactly the pairwise permutation of neighboring items*.

We note that the approximation can be made arbitrarily precise by letting $\sigma \to 0_+$. We also note that $\sigma \approx 0$ is preferred by the algorithm's design to achieve the theoretical guarantees on the optimization of $\mathcal{L}_N$ by StochasticRank.

Such observation suggests that the pairwise permutations of neighboring items are the only permutations that can be achieved by small perturbations of the model we are learning. Hence, $\mathcal{L}_t^*$ that arises from YetiLoss contains more information about *local* behavior of the ranking loss function $\mathcal{L}_N$.

The similarity between YetiLoss and StochasticRank allows us to address the following question in the next section: is direct optimization of the *non-convex* smoothed loss function better than optimization of a *convex* surrogate loss?

4. Experiments

This section empirically compares popular LTR algorithms, evaluates the proposed modification called YetiLoss, and analyzes the importance of several algorithmic details in a unified setup.

Table 1. Learning-to-rank datasets

Dataset	# features	# queries			# documents		
		train	validation	test	train	validation	test
Web10K	136	6,000	2,000	2,000	723,412	235,259	241,521
Web30K	136	18,919	6,306	6,306	2,270,296	747,218	753,611
Yahoo S1	699	19,944	2,944	6,983	473,134	71,083	165,660
Yahoo S2	699	1,266	1,266	3,798	34,815	34,881	103,174
Istella	220	20,307	2,912	9,799	5,497,064	1,828,561	3,129,004
Istella-S	220	19,246	7,211	6,562	2,043,304	684,076	681,250

4.1. Setup

Datasets We use six publicly available datasets. The first two are Web10K and Web30K released by Microsoft (Qin & Liu, 2013). Following previous studies (Qin et al., 2021; Ustimenko & Prokhorenkova, 2020; Wang et al., 2018), we use Fold 1 for these two datasets. We also use two datasets from YAHOO! Learning to Rank Challenge (Chapelle & Chang, 2011). Finally, we take Istella and Istella-S datasets (Dato et al., 2016). All datasets except for Istella are pre-divided into the train, validation, and test sets. For Istella, there is no standard validation set, so we randomly divided the train part into train and validation. Table 1 overviews the datasets used in the current study.

Ranking loss functions We compare the algorithms according to all quality functions described in Section 2.2.

The first one is NDCG@10, which is very popular in LTR research. In some of our experiments, we also use NDCG@1 and NDCG@5. We observe no significant differences in the obtained results, so we omit these losses from the main text. Some results for NDCG@1 and NDCG@5 can be found in Appendix.

The second quality function is MRR, which is a well-known click-based metric. As MRR requires binary labels, we binarize each label as $\widetilde{y}_i := \mathbb{1}_{\{y_i > 0\}}$. MRR is often used in practice while it is much less studied than NDCG@k.

We also consider MAP, for which we binarize the labels in the same way.

Finally, for ERR, we convert the labels to $[0, 1]$ as $\widetilde{y}_i := y_i/4$. Using all these different loss functions is essential for analyzing the generalizability of the approaches to different ranking tasks.

Algorithms All the algorithms are implemented within the CatBoost gradient boosting library (Prokhorenkova et al., 2018).

For YetiRank, we use the official implementation, which we modify to obtain YetiLoss compatible with NDCG, MRR,

ERR, and MAP.

For StochasticRank, we use the official implementation for NDCG, while we also add MRR and ERR loss functions to the optimization. MAP is not supported in StochasticRank out of the box, and Ustimenko & Prokhorenkova (2020) only provide algorithms for efficiently computing ERR-like and DCG-like metrics. Making an efficient and correct gradient estimate for MAP in StochasticRank can be a subject of a separate investigation, which is why we only implemented MRR and ERR.

For LambdaMART, we implement the original algorithm (Wu et al., 2010) within the CatBoost library with all ranking quality functions considered in our analysis.

For completeness of the analysis, we also add the CatBoost trained in the QueryRMSE regime (CatBoost, 2022). In this case, the algorithm optimizes RMSE averaged over the queries. This simple loss function can be considered as a baseline for other methods.

Finally, we also add the implementation of LambdaMART provided by Microsoft within the LightGBM library (Ke et al., 2017). This implementation is known to outperform other open-source versions and to achieve state-of-the-art LTR results for NDCG (Qin et al., 2021), while it does not allow for optimizing MRR, ERR, or MAP. Let us note, however, that LightGBM and CatBoost are different implementations of GBDT. This fact prevents a direct comparison of LTR algorithms. For instance, in contrast to LightGBM, CatBoots uses oblivious decision trees, where the same splitting criterion is used for all tree nodes at a given depth.

Parameter tuning For parameter tuning, we use 20 iterations of random search followed by 80 iterations of Bayesian optimization (Balandat et al., 2020).[3] For all algorithms, we set the maximum number of trees to 1000. We choose the best parameters, including the optimal number of trees, using the value of the desired loss function on the validation set. The list of tuned parameters is given in Appendix A.

---

[3]Small variations of the results compared to the previous papers can be explained by a particular parameter optimization used.

Table 2. Comparison of learning-to-rank algorithms with tuned hyper-parameters

Algorithm	Web10K	Web30K	Yahoo S1	Yahoo S2	Istella	Istella-S
			NDCG@10			
LightGBM	50.39	52.21	78.88	77.25	72.89	76.48
QueryRMSE	50.55	52.19	79.10	**78.01**	69.66	74.83
LambdaMART	49.38	50.93	78.63	77.31	69.83	75.55
StochasticRank	50.34	51.63	78.98	77.77	68.46	74.88
YetiRank	**50.75**	**52.31**	**79.19**	**77.99**	**73.02**	**77.15**
YetiLoss	**50.76**	51.90	78.67	77.77	71.60	76.32
			MRR			
QueryRMSE	83.47	85.38	91.03	92.56	96.23	97.63
LambdaMART	82.59	84.12	90.75	92.34	95.18	96.96
StochasticRank	83.51	85.43	90.72	93.08	96.54	97.83
YetiRank	**84.21**	85.58	**91.20**	93.05	96.70	97.58
YetiLoss	84.17	**86.01**	90.88	**93.21**	**97.16**	**97.97**
			MAP			
QueryRMSE	62.11	63.51	86.06	88.40	78.11	88.59
LambdaMART	57.93	58.58	85.91	89.14	80.80	90.28
YetiRank	62.07	63.39	85.94	88.00	80.01	89.09
YetiLoss	**62.62**	**64.03**	**86.46**	**89.37**	**82.66**	**91.07**
			ERR			
QueryRMSE	57.27	59.30	**66.76**	67.16	85.29	85.55
LambdaMART	56.54	58.76	66.22	66.55	84.71	85.24
StochasticRank	56.94	59.34	66.64	67.17	85.44	85.98
YetiRank	57.41	**59.53**	66.69	**67.27**	**87.29**	**86.97**
YetiLoss	**57.46**	**59.53**	66.71	67.15	86.83	86.59

4.2. Comparison of LTR Algorithms

The results of the comparison are present in Table 2. The best results are highlighted. The differences between the highlighted results and other algorithms are statistically significant according to the paired one-tailed t-test with p-value < 0.05.

We make the following observations. First, we confirm the conclusion from Ustimenko & Prokhorenkova (2020) that StochastiRank outperforms LambdaMART in most cases. However, YetiRank beats both of them, leading to state-of-the-art results, especially for NDCG@10. In turn, the proposed YetiLoss modification of YetiRank achieves the best results for MAP outperforming the competitors by a considerable margin. It is also the best on most datasets for MRR and performs similarly to YetiRank for ERR. For NDCG@10, YetiRank outperforms YetiLoss: this can be explained by the fact that the loss function optimized within YetiRank is similar to NDCG.

Interestingly, the simple QueryRMSE method is very competitive in some cases. For instance, for NDCG@10, it is su-

perior to LightGBM, LambdaMART, and StochasticRank in at least half of the cases. QueryRMSE also achieves the best results on Yahoo S2 for NDCG@10 and on Yahoo S1 for ERR. However, in most cases, YetiLoss is significantly better than QueryRMSE. Our experiments with QueryRMSE show that considering such simple ranking-agnostic baselines is important for a fair analysis.

By comparing YetiLoss with StochasticRank, we can see that convexity of the optimization problem is usually preferred over direct optimization of a non-convex smoothed ranking loss: except for some results on Yahoo datasets, YetiLoss outperforms StochasticRank.

To better understand the differences between LTR algorithms, we also analyze their generalization ability. These results are shown in Appendix B.

Comparison with default hyper-parameters To evaluate the effect of parameter tuning on the obtained experimental result, we additionally evaluate all the algorithms with their default set of hyper-parameters (only the optimal number of trees is chosen on the validation set). The results

Table 3. The effect of stochastic smoothing for YetiLoss

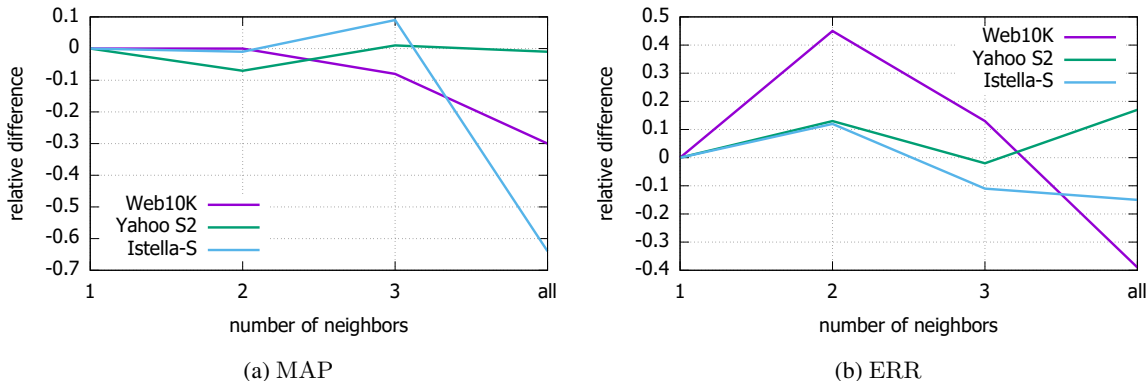| | NDCG@10 | | | MRR | | | MAP | | |
	Web10K	Yahoo S2	Istella-S	Web10K	Yahoo S2	Istella-S	Web10K	Yahoo S2	Istella-S
Logistic	50.76	**77.77**	**76.32**	84.17	93.21	**97.97**	**62.62**	**89.37**	91.07
Gaussian	**50.79**	77.55	76.28	**84.28**	**93.61**	97.83	62.52	**89.37**	**91.08**
No smoothing	48.49	75.29	74.55	82.70	91.94	97.38	61.32	88.13	89.89

(a) MAP

(b) ERR

Figure 1. The effect of the number of neighbors, relative loss compared to YetiLoss

are shown in Table 4. Note that all our observations are also present for this setup. Indeed, YetiLoss is the best with a significant margin for MRR and MAP. Moreover, it wins on half of the datasets for ERR. For NDCG@10, similarly to Table 2, the best results are achieved with YetiRank.

4.3. Analysis of Algorithmic Details

Stochastic treatment of scores First, we analyze the effect of stochastic smoothing of weights. Recall that the presence of such smoothing is the key difference between YetiLoss and LambdaMART. Table 3 confirms that smoothing is essential for YetiLoss. On the other hand, a particular shape of smoothing is not so important: the results obtained with Gaussian and Logistic smoothing are similar.

Number of neighbors Another difference between YetiLoss and LambdaMART is that YetiLoss uses only neighboring documents with $|p_i - p_j| = 1$ while computing the weights. To analyze the effect of this detail, we consider modifications of YetiLoss with $|p_i - p_j| = k$ for $k \in \{1, 2, 3\}$. We also added the LambdaMART version, where all pairs are considered. We can see that there is no stable significant dependence on this parameter. However, choosing $k = 2$ is a good option. Note that smaller values of k lead to more efficient algorithms.

5. Conclusion and Discussion

This paper analyzes several GBDT-based state-of-the-art LTR algorithms and their modifications to determine which algorithmic details are important for better learning to rank. In our experiments, we found that YetiRank is currently the state-of-the-art LTR approach. We make a simple modification of YetiRank and propose an extension called YetiLoss that can optimize arbitrary ranking loss functions. The experiments confirm that YetiLoss is better at optimizing MRR and is considerably better than the competitors on all the datasets for MAP. Another important contribution of our research is a thorough comparison of existing state-of-the-art learning-to-rank algorithms and their algorithmic details.

Note that our work is limited to GBDT-based models. We chose this particular setup as GBDTs are known to outperform other approaches on classic tabular LRT benchmarks. Indeed, Qin et al. (2021) compare the existing neural approaches with GBDT-based methods and show that if fairly evaluated, GBDTs still outperform neural methods.[4] This agrees with the experiments of Gorishniy et al. (2021) comparing GBDTs and neural approaches on non-ranking tabular datasets. Importantly, GBDTs have other advantages useful in some applications: they are fast, easy to use, and do not require much parameter tuning. Also, in production

---

[4]The model proposed by Qin et al. (2021) is comparable with GBDTs, but it uses self-attention, so to score a particular document the neural network uses features of other documents that are not available to GBDT.

Table 4. Comparison of learning-to-rank algorithms with default hyper-parameters

Algorithm	Web10K	Web30K	Yahoo S1	Yahoo S2	Istella	Istella-S
			NDCG@10			
LightGBM	**50.34**	**51.06**	78.34	76.96	67.71	73.67
QueryRMSE	49.82	50.68	78.34	78.03	65.45	72.60
LambdaMART	48.56	49.50	77.44	76.46	66.25	73.37
YetiRank	50.18	50.91	**78.58**	**78.15**	**68.86**	**74.72**
YetiLoss	50.14	50.91	78.03	77.55	68.73	74.62
			MRR			
QueryRMSE	83.44	84.87	90.78	93.25	94.76	96.89
LambdaMART	82.36	83.97	90.72	91.84	94.73	96.81
YetiRank	83.81	85.10	91.02	93.16	95.59	97.09
YetiLoss	**84.50**	**85.98**	**91.14**	**93.76**	**96.40**	**97.71**
			MAP			
QueryRMSE	61.85	63.00	85.66	88.40	72.61	85.95
LambdaMART	56.89	55.81	85.92	89.14	75.21	87.49
YetiRank	61.80	63.03	85.66	88.00	75.36	86.82
YetiLoss	**62.43**	**63.58**	**86.41**	**89.36**	**76.61**	**88.14**
			ERR			
QueryRMSE	56.98	58.64	66.26	67.20	83.06	84.14
LambdaMART	55.92	57.74	65.99	66.41	83.01	84.41
YetiRank	57.12	58.73	**66.54**	**67.28**	85.31	**85.77**
YetiLoss	**57.16**	**58.80**	66.50	67.22	**85.46**	85.71

scenarios (e.g., in search engines), neural networks usually compute some features that are then passed to a GBDT model to be combined with other heterogeneous features. GBDT models are often used on top of neural networks since they are known to be good at dealing with heterogeneous features of different natures and scales. That is why we strongly believe that the analysis of GBDT-based ranking is very important.

On the other hand, a useful direction for future research is to perform a similar systematic analysis for neural network models. All the discussed approaches (YetiLoss, Query-RMSE, and StochasticRank) can be straightforwardly applied to neural rankers. However, it is not guaranteed that the conclusions obtained in the current research would transfer to other types of models and problems, and a separate analysis is required, similarly to what is done for the LambdaLoss framework by Jagerman et al. (2022). In addition, it would be useful to compare the neural rankers in a suitable setup like image retrieval or recommendation systems.

Finally, note that all the algorithms are implemented within the CatBoost open-source gradient boosting library for a fair comparison. We choose this particular library since it contains the original implementations of StochasticRank and YetiRank so that we can compare these methods and their

modifications fairly. While we believe the obtained conclusions would transfer to other libraries, additional analysis is needed to confirm this.

References

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

Bruch, S., Han, S., Bendersky, M., and Najork, M. A stochastic treatment of learning to rank scoring functions. In *Proceedings of the 13th ACM International Conference on Web Search and Data Mining*, pp. 61–69, 2020.

Burges, C. J., Ragno, R., and Le, Q. V. Learning to rank with nonsmooth cost functions. *Proceedings of the Advances in Neural Information Processing Systems*, 19:193–200, 2007.

Burges, C. J. C. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research, 2010.

CatBoost. Ranking: objectives and metrics. `https://catboost.ai/docs/concepts/loss-functions-ranking.html`, 2022.

Chapelle, O. and Chang, Y. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*, pp. 1–24, 2011.

Dato, D., Lucchese, C., Nardini, F. M., Orlando, S., Perego, R., Tonellotto, N., and Venturini, R. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Transactions on Information Systems (TOIS)*, 35(2):1–31, 2016.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems 34*, 2021.

Gulin, A., Kuralenok, I., and Pavlov, D. Winning the transfer learning track of Yahoo!'s learning to rank challenge with YetiRank. In *Proceedings of the Learning to Rank Challenge*, pp. 63–76, 2011.

Jagerman, R., Qin, Z., Wang, X., Bendersky, M., and Najork, M. On optimizing top-k metrics for neural ranking models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2303–2307, 2022.

Katzir, L., Elidan, G., and El-Yaniv, R. Net-DNF: Effective deep modeling of tabular data. In *International Conference on Learning Representations*, 2021.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, pp. 3146–3154, 2017.

Liu, T.-Y. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

Nesterov, Y. and Spokoiny, V. G. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17:527–566, 2017.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pp. 6638–6648, 2018.

Qin, T. and Liu, T. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013.

Qin, Z., Yan, L., Zhuang, H., Tay, Y., Pasumarthi, R. K., Wang, X., Bendersky, M., and Najork, M. Are neural rankers still outperformed by gradient boosted decision trees? In *International Conference on Learning Representations*, 2021.

Ustimenko, A. and Prokhorenkova, L. StochasticRank: Global optimization of scale-free discrete functions. In *International Conference on Machine Learning*, pp. 9669–9679, 2020.

Ustimenko, A. and Prokhorenkova, L. SGLB: Stochastic Gradient Langevin Boosting. In *International Conference on Machine Learning*, pp. 10487–10496, 2021.

Wang, X., Li, C., Golbandi, N., Bendersky, M., and Najork, M. The LambdaLoss framework for ranking metric optimization. In *Proceedings of The 27th ACM International Conference on Information and Knowledge Management*, pp. 1313–1322, 2018.

Wu, Q., Burges, C. J., Svore, K. M., and Gao, J. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.

A. Experimental Setup

We tune the following hyper-parameters:

- *learning-rate*: log-uniform distribution over $[10^{-3}, 1]$ for all models;
- *l2-leaf-reg*: 0 for StochasticRank; log-uniform distribution over $[0.1, 100]$ for LightGBM; over $[0.001, 100]$ for other models;
- *depth:* uniform distribution over $\{6, \ldots, 8\}$ for YetiRank and LambdaMART, over $\{6, \ldots, 10\}$ for StochasticRank, over $\{6, \ldots, 16\}$ for LightGBM;
- *model-shrink-rate*: log-uniform distribution over $[10^{-5}, 10^{-2}]$ for StochasticRank;
- *diffusion-temperature*: log-uniform distribution over $[10^8, 10^{11}]$ for StochasticRank;
- *mu*: log-uniform distribution over $[10^{-2}, 10]$ for StochasticRank;
- *num-leaves*: integer log-uniform distribution over $[16, 256]$ for LightGBM;
- *min-data-in-leaf*: integer log-uniform distribution over $[1, 1000]$ for LightGBM; default value of 10 for other algorithms.

The best hyper-parameters used for Table 2 are listed in Tables 27-30.

B. Additional Experimental Details

Generalization gap   Tables 23-26 provide additional information on the generalization of different algorithms. The column "train" shows the corresponding error on the train set, while "gap" is the difference of the error between the train and test sets. A smaller value in "gap" indicates better generalization.

As expected, YetiLoss usually has larger generalization gaps than YetiRank (though there are some exceptions) since it is trained for a particular target quality function on the train set. Also, YetiLoss usually has better quality on the train set compared to YetiRank. Compared with LightGBM, all CatBoost-based algorithms usually have better generalization (see Table 23). We assume that this is because CatBoost uses oblivious decision trees known to be resistant to overfitting.

Number of neighbors   Extended results on the analysis of the number of neighbors (including additional loss functions NDCG@1 and NDCG@5) can be found in Tables 5-10.

Stochastic smoothing   Extended results on the analysis of stochastic smoothing (including additional loss functions NDCG@1 and NDCG@5) can be found in Tables 11-22.

Table 5. The effect of the number of neighbors for YetiLoss, NDCG@1

	Web	Yahoo	Istella
1	48.24	74.64	70.36
2	49.27	74.60	70.70
all	47.70	74.74	70.93

Table 6. The effect of the number of neighbors for YetiLoss, NDCG@5

	Web	Yahoo	Istella
1	48.61	74.49	70.39
2	49.02	74.29	70.61
all	48.36	74.65	70.44

Table 7. The effect of the number of neighbors for YetiLoss, NDCG@10

	Web	Yahoo	Istella
1	50.76	77.77	76.32
2	50.90	77.68	76.69
3	50.99	77.76	76.74
all	50.56	77.90	76.79

Table 8. The effect of the number of neighbors for YetiLoss, MRR

	Web	Yahoo	Istella
1	84.17	93.21	97.97
2	84.18	93.75	97.99
3	84.39	93.22	97.82
all	84.19	93.84	97.90

Table 9. The effect of the number of neighbors for YetiLoss, MAP

	Web	Yahoo	Istella
1	62.62	89.37	91.07
2	62.62	89.30	91.06
3	62.54	89.38	91.16
all	62.32	89.36	90.43

Table 10. The effect of the number of neighbors for YetiLoss, ERR

	Web	Yahoo	Istella
1	57.46	67.15	86.59
2	57.91	67.28	86.71
3	57.59	67.13	86.48
all	57.07	67.32	86.44

Table 11. The effect of stochastic smoothing for YetiRank, NDCG@1

	Web	Yahoo	Istella
Logistic	48.59	75.34	70.8
Gaussian	49.86	75.19	71.62
No smoothing	48.26	73.7	70.47

Table 17. The effect of stochastic smoothing for YetiLoss, NDCG@1

	Web	Yahoo	Istella
Logistic	48.24	74.64	70.36
Gaussian	49.99	74.36	70.45
No smoothing	48.75	71.89	68.32

Table 12. The effect of stochastic smoothing for YetiRank, NDCG@5

	Web	Yahoo	Istella
Logistic	48.54	74.93	70.81
Gaussuan	49.4	74.99	70.94
No smoothing	47.52	72.14	69.82

Table 18. The effect of stochastic smoothing for YetiLoss, NDCG@5

	Web	Yahoo	Istella
Logistic	48.61	74.49	70.39
Gaussian	49.25	74.17	70.37
No smoothing	46.85	71.51	68.46

Table 13. The effect of stochastic smoothing for YetiRank, NDCG@10

	Web	Yahoo	Istella
Logistic	50.75	77.99	77.15
Gaussian	51	78.07	77.23
No smoothing	49.05	75.57	75.8

Table 19. The effect of stochastic smoothing for YetiLoss, NDCG@10

	Web	Yahoo	Istella
Logistic	50.76	77.77	76.32
Gaussian	50.79	77.55	76.28
No smoothing	48.49	75.29	74.55

Table 14. The effect of stochastic smoothing for YetiRank, MRR

	Web	Yahoo	Istella
Logistic	84.21	93.05	97.58
Gaussian	83.85	93.07	97.47
No smoothing	83.15	92.23	97.13

Table 20. The effect of stochastic smoothing for YetiLoss, MRR

	Web	Yahoo	Istella
Logistic	84.17	93.21	97.97
Gaussian	84.28	93.61	97.83
No smoothing	82.7	91.94	97.38

Table 15. The effect of stochastic smoothing for YetiRank, MAP

	Web	Yahoo	Istella
Logistic	62.07	88	89.09
Gaussian	61.85	87.8	89.02
No smoothing	60.38	86.85	87.47

Table 21. The effect of stochastic smoothing for YetiLoss, MAP

	Web	Yahoo	Istella
Logistic	62.62	89.37	91.07
Gaussian	62.52	89.37	91.08
No smoothing	61.32	88.13	89.89

Table 16. The effect of stochastic smoothing for YetiRank, ERR

	Web	Yahoo	Istella
Logistic	57.41	67.27	86.97
Gaussian	57.76	67.15	86.87

Table 22. The effect of stochastic smoothing for YetiLoss, ERR

	Web	Yahoo	Istella
Logistic	57.46	67.15	86.59
Gaussian	57.55	66.99	86.25

Table 23. Analysis of generalization gap for NDCG@10

Algorithm	Web10K		Web30K		Yahoo S1		Yahoo S2		Istella		Istella-S	
	train	gap	train	gap	train	gap	train	gap	train	gap	train	gap
YetiRank	56.88	6.13	57.43	5.12	86.56	7.37	90.53	12.54	79.86	6.83	85.86	8.71
YetiLoss	57.68	6.92	52.48	3.57	80.46	5.34	94.83	17.06	77.03	5.57	85.23	8.91
StochasticRank	56.97	10.23	54.96	6.33	82.16	6.73	89.66	13.73	72.02	3.71	82.53	8.17
LambdaMART	55.25	5.87	55.72	4.79	83.35	4.72	90.48	13.17	73.38	3.55	80.12	4.57
LightGBM	56.04	9.25	59.58	10.37	84.99	9.67	84.79	9.39	82.36	9.61	87.73	11.77

Table 24. Analysis of generalization gap for MRR

Algorithm	Web10K		Web30K		Yahoo S1		Yahoo S2		Istella		Istella-S	
	train	gap	train	gap	train	gap	train	gap	train	gap	train	gap
YetiRank	87.02	2.80	86.85	1.27	93.06	1.86	96.26	3.21	98.40	1.70	99.48	1.90
YetiLoss	87.82	3.64	87.64	1.63	94.93	4.04	97.88	4.67	98.98	1.82	99.85	1.88
StochasticRank	88.14	4.63	89.20	3.77	93.92	3.20	95.64	2.56	98.24	1.70	99.45	1.62
LambdaMART	86.11	3.52	85.27	1.15	93.26	2.51	96.01	3.67	97.60	2.42	99.66	2.70

Table 25. Analysis of generalization gap for MAP

Algorithm	Web10K		Web30K		Yahoo S1		Yahoo S2		Istella		Istella-S	
	train	gap	train	gap	train	gap	train	gap	train	gap	train	gap
YetiRank	65.38	3.31	65.02	1.63	89.48	3.54	94.16	6.15	84.55	4.53	92.26	3.18
YetiLoss	65.86	3.24	66.22	2.20	90.70	4.24	96.06	6.69	89.40	6.74	96.27	5.20
LambdaMART	62.40	4.46	59.89	1.32	87.13	1.22	96.50	7.36	85.75	4.94	94.02	3.74

Table 26. Analysis of generalization gap for ERR

Algorithm	Web10K		Web30K		Yahoo S1		Yahoo S2		Istella		Istella-S	
	train	gap	train	gap	train	gap	train	gap	train	gap	train	gap
YetiRank	61.80	4.39	62.53	3.00	70.34	3.64	73.40	6.13	91.30	4.02	94.53	7.55
YetiLoss	61.72	4.26	62.11	2.58	70.06	3.34	72.23	5.08	91.00	4.17	94.14	7.55
StochasticRank	62.82	5.88	62.81	3.48	70.00	3.36	73.26	6.09	87.77	2.33	91.82	5.84
LambdaMART	63.85	7.31	63.68	4.92	68.85	2.63	72.04	5.50	87.82	3.11	91.89	6.65

13

Table 27. Chosen parameters for NDCG@10

Algorithm	Parameter	Web10K	Web30K	Yahoo S1	Yahoo S2	Istella	Istella-S
YetiRank	depth	8	8	8	7	8	8
	l2-leaf-reg	0.03343	0.00100	0.00588	0.08913	0.00461	0.00100
	learning-rate	0.05450	0.10085	0.10285	0.04172	0.18317	0.18166
YetiLoss	depth	8	8	8	8	8	8
	l2-leaf-reg	0.00625	0.11205	0.10000	0.00313	0.13938	0.00100
	learning-rate	0.05111	0.11108	0.09002	0.03479	0.25685	0.15083
StochasticRank	depth	10	10	10	8	10	10
	learning-rate	0.18210	0.30549	0.12293	0.05506	1.00000	1.00000
	model-shrink-rate	0.00001	0.00117	0.00049	0.01000	0.00075	0.00125
	diffusion-temperature	675M	63274M	100M	102M	304M	5873M
	mu	0.02034	0.01314	0.06495	0.04142	0.01190	0.01794
LambdaMART	depth	8	8	8	6	8	7
	l2-leaf-reg	0.00394	0.00170	0.00176	0.00100	0.00100	0.00100
	learning-rate	0.07478	0.14310	0.14611	0.10625	0.18117	0.15533
LightGBM	depth	14	14	16	9	14	16
	learning-rate	0.16656	0.12405	0.09355	0.12593	0.22409	0.22599
	num-leaves	97	256	256	16	256	256
	min-data-in-leaf	1	1	2	11	9	1
	l2-leaf-reg	73.06401	0.39895	6.58340	11.54666	0.74747	1.76279

Table 28. Chosen parameters for MRR

Algorithm	Parameter	Web10K	Web30K	Yahoo S1	Yahoo S2	Istella	Istella-S
YetiRank	depth	7	7	8	6	8	8
	l2-leaf-reg	0.23902	0.55462	0.01642	6.24819	0.04491	0.11146
	learning-rate	0.09387	0.22263	0.07333	0.23151	0.14079	0.17564
YetiLoss	depth	6	8	7	8	8	8
	l2-leaf-reg	0.71013	0.32674	0.10000	0.02186	0.10000	0.00356
	learning-rate	0.16364	0.07572	0.15166	0.07400	0.18719	0.06319
StochasticRank	depth	7	10	10	6	10	10
	learning-rate	0.19705	0.34981	0.14603	0.04305	1.00000	0.36481
	model-shrink-rate	0.00001	0.00016	0.01000	0.00001	0.00006	0.00031
	diffusion-temperature	251M	8890M	100M	100M	100M	984M
	mu	0.01120	0.01000	0.03325	0.01689	0.03698	0.01000
LambdaMART	depth	6	6	8	8	6	6
	l2-leaf-reg	0.00769	0.04354	0.00100	0.01220	0.00100	0.00100
	learning-rate	0.23696	0.08981	0.07174	0.12050	0.09162	0.09379

Table 29. Chosen parameters for MAP

Algorithm	Parameter	Web10K	Web30K	Yahoo S1	Yahoo S2	Istella	Istella-S
YetiRank	depth	8	8	8	7	8	8
	l2-leaf-reg	0.01157	0.00320	0.00708	0.20218	0.00287	0.02457
	learning-rate	0.05033	0.09699	0.11392	0.05680	0.18562	0.15196
YetiLoss	depth	8	8	8	8	8	8
	l2-leaf-reg	0.54619	0.04957	0.05314	0.00274	0.00819	0.02790
	learning-rate	0.05031	0.07755	0.06111	0.01103	0.25184	0.25536
LambdaMART	depth	8	8	8	7	8	8
	l2-leaf-reg	0.01119	0.00441	0.00100	0.00879	0.00100	0.00100
	learning-rate	0.01769	0.01267	0.02798	0.04096	0.18745	0.16072

Table 30. Chosen parameters for ERR

Algorithm	Parameter	Web10K	Web30K	Yahoo S1	Yahoo S2	Istella	Istella-S
YetiRank	depth	8	8	8	8	8	8
	l2-leaf-reg	0.06115	0.00100	0.00309	0.01318	0.02537	0.00100
	learning-rate	0.06827	0.09341	0.09871	0.02678	0.17561	0.16226
YetiLoss	depth	8	8	8	6	8	8
	l2-leaf-reg	0.01401	0.00175	0.00966	0.05289	0.00100	0.00160
	learning-rate	0.04484	0.07811	0.06638	0.05021	0.12214	0.09334
StochasticRank	depth	10	10	10	10	10	10
	learning-rate	0.14777	0.30041	0.25101	0.05651	0.54801	0.39642
	model-shrink-rate	0.00042	0.00001	0.00004	0.01000	0.00085	0.00206
	diffusion-temperature	100M	589M	344M	1425M	100B	100M
	mu	0.12610	0.02085	0.12851	0.18189	0.01375	0.01000
LambdaMART	depth	8	8	8	6	6	6
	l2-leaf-reg	0.00100	0.00100	0.00116	0.00100	0.00100	0.00100
	learning-rate	0.05517	0.10631	0.09575	0.04895	0.15040	0.15975