

U-rank: Utility-oriented Learning to Rank with Implicit Feedback

Xinyi Dai¹, Jiawei Hou¹, Qing Liu², Yunjia Xi¹, Ruiming Tang²,
Weinan Zhang¹, Xiuqiang He², Jun Wang³, Yong Yu¹

¹Shanghai Jiao Tong University, ²Huawei Noah's Ark Lab, ³University College London
{daixinyi,hjw99868,wnzhang,yyu}@sjtu.edu.cn,{liuqing48,tangruiming,hexiuqiang1}@huawei.com,jun.wang@cs.ucl.ac.uk

ABSTRACT

Learning to rank with implicit feedback is one of the most important tasks in many real-world information systems where the objective is some specific utility, e.g., clicks and revenue. However, we point out that existing methods based on probabilistic ranking principle do not necessarily achieve the highest utility. To this end, we propose a novel ranking framework called *U-rank* that directly optimizes the expected utility of the ranking list. With a position-aware deep click-through rate prediction model, we address the attention bias considering both query-level and item-level features. Due to the item-specific attention bias modeling, the optimization for expected utility corresponds to a maximum weight matching on the item-position bipartite graph. We base the optimization of this objective in an efficient LambdaLoss framework, which is supported by both theoretical and empirical analysis. We conduct extensive experiments for both web search and recommender systems over three benchmark datasets and two proprietary datasets, where the performance gain of *U-rank* over state-of-the-arts is demonstrated. Moreover, our proposed *U-rank* has been deployed on a large-scale commercial recommender and a large improvement over the production baseline has been observed in an online A/B testing.

CCS CONCEPTS

• Information systems → Learning to rank.

KEYWORDS

Learning to Rank, Utility Maximization, Position Bias, Implicit Feedback

ACM Reference Format:

Xinyi Dai, Jiawei Hou, Qing Liu, Yunjia Xi, Ruiming Tang, Weinan Zhang, Xiuqiang He, Jun Wang and Yong Yu. 2020. *U-rank: Utility-oriented Learning to Rank with Implicit Feedback*. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, October 19–23, 2020, Virtual Event, Ireland. ACM, NY, NY, USA, 8 pages. <https://doi.org/10.1145/3340531.3412756>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).
CIKM '20, October 19–23, 2020, Virtual Event, Ireland

© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6859-9/20/10...\$15.00
<https://doi.org/10.1145/3340531.3412756>

1 INTRODUCTION

Ranking is the core of information retrieval. In the traditional web search scenario, learning to rank (LETOR) methods are proposed to optimize the ranked list based on human-annotated relevance labels [20]. Typically these methods sort the documents according to their probability of relevance in descending order, according to the famous probabilistic ranking principle (PRP) [28]. Due to the lack of annotated labels, recently, many works have focused on learning to rank via implicit feedback, such as user's click data, which is timely and personalized. These works are also based on PRP, where the relevance is estimated from implicit feedback through counterfactual methods [2, 14, 16, 30, 31]. Besides the traditional web search scenario, nowadays, ranking is also an important part of many real-world applications, including recommender systems [17], online advertising [29] and product search [18]. In these applications, specific utility metrics (such as clicks, conversions and revenue, etc.) are proposed, by which the quality of a ranked list is evaluated.

In many existing works, LETOR algorithms are derived on the basis of PRP, and then evaluated on some utility metrics [18, 35]. However, we find that the PRP based ranking framework does not necessarily bring the highest utility in reality. To be more specific, PRP is basically correct for items with large differences in relevance estimation. However, for two items with close relevance estimation, putting the item which is more sensitive to position change at the higher position will bring a higher expected utility, even if it is slightly less relevant. To provide a persuasive example, we show the average click curve of five popular apps from a mainstream App Store in the right panel of Figure 1. Consider a non-personalized case for simplicity that we recommend App 1, App 2, and App 3 to one user. If the apps are sorted by PRP, the ranked list will be App 1, App 2, and App 3 by their relevance in terms of click-through rate (CTR). However, the optimal ranked list with the maximum utility should be App 1, App 3, and App 2, since the utility gain of promoting App 3 from the 3rd to 2nd position is 0.019, which is larger than the utility loss 0.010 of dragging App 2 from the 2nd to 3rd position. As can be seen, sorting items by relevance may fail to achieve the highest utility in some situations, which is actually quite common in industrial scenarios. Therefore, we aim to optimize the objectives that are directly related to the utility based on user's implicit feedback.

Optimizing the utility metric like CTR of the whole ranked list is not as easy as it seems. One direct solution might be estimating *only one unique* CTR of each query-item pair and then optimizing the certain metric w.r.t. the whole list using the estimated CTR [33]. However, bias will be introduced in this solution since user's CTR is not a static property like relevance. To be clear, for the same query-item pair, the CTR might change with its presented position. As

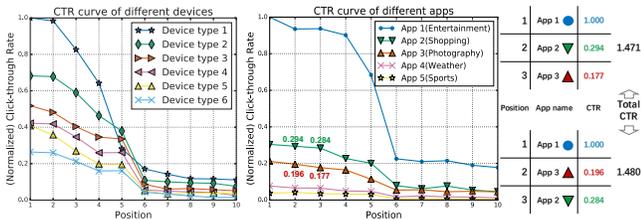


Figure 1: The CTR analysis w.r.t. query/item features. The data is collected through a 120 days’ click log on random recommendation traffic in a mainstream App Store.

shown in Figure 1, the CTR decreases as the presented position goes from top to bottom, and moreover, the magnitude of the decrease is different among items and device types.

In order to design effective utility-oriented algorithms, we need to figure out why this phenomenon happens and then investigate how to deal with it. The decrease of user’s CTR mainly results from the decrease of user attention, which is supported by eye-tracking studies [21, 22]. Most existing works have treated such attention bias as position bias [16, 30], i.e., more attention is paid to the top positions than the bottom ones. In the literature, position bias is considered to be decorrelated with the ranked items, i.e., makes the same effect on all items [8, 27], which is generally correct in the traditional *10 blue links* scenario. Under such assumption, following PRP achieves the goal of the highest expected utility since the click curves of different positions across different items have the same shape despite the different scales.

However, we argue that in many real-world applications, a user’s attention on items does not only depend on the positions but also the item attributes and the user contexts. For the App recommendation case as demonstrated in Figure 1, visual difference in the thumbnail of a product or the preview frame of videos leads to item-specific attention bias. In web search, an example of item-specific attention bias is vertical bias, commonly observed when the page contains vertical search results (such as images, videos, maps, etc.). For example, Metrikov et al. [24] found that an image in search result can raise CTR and flatten the click curve at the same time. A visually attractive content, like a vertical search result or an item with a fancy thumbnail, can still attract user’s attention even it is placed at a lower position, leading to a flatter click curve. In other words, such visually attractive results are less sensitive to the position change. In the example above, CTR of App 2 is less sensitive in whether placing it in position 2 or position 3 compared to App 3. Placing items of which CTR is more sensitive to position change at top positions often leads to a higher utility. Besides, query-level features like device type, as shown in Figure 1, also leads to different attention biases. Hence, to obtain unbiased CTR estimation, we need to exploit both the item-level and the query-level features to model the dependency between click and position.

Based on these considerations, in this work, we propose a ranking framework called *U-rank* that directly optimizes expected utility from implicit feedback. Instead of ranking according to PRP, we first derive a new list-wise learning objective, of which the expectation is the utility metric we want to maximize. Then to obtain an unbiased estimation of the expected utility, we address the attention bias considering both the query-level and item-level features with

a position-aware deep CTR model. Finally, to efficiently optimize the expected utility, we formulate it as an item-position matching problem as shown in Figure 2, and learn a scoring function towards the best matching through pairwise permutations inspired by Lambdaloss framework [32], which reduces the complexity in inference stage from $O(N^3)$ to $O(N)$. Theoretical analysis demonstrates that we solve an upper bound problem of the matching problem.

We conduct thorough experiments on three benchmark LETOR datasets and a large-scale real-world commercial recommendation dataset to verify the effectiveness of *U-rank*. Further, *U-rank* has been deployed on the recommender system of a mainstream App Store, where a 10-day online A/B test shows that *U-rank* achieves an average improvement of 19.2% on CTR and 20.8% on conversion rate over the production baseline.

2 RELATED WORK

Generative click models are introduced to study user browsing behavior and extract unbiased relevance feedbacks from click data. For example, Position-based model (PBM) [27] assumes that a click only depends on the position and the relevance of the document. Cascade model [8] assumes that user browses a search web page sequentially from top to bottom until a relevant document is found. Following these two classical click models, more sophisticated ones (e.g., UBM [9], DBN [7], CCM [11] and NCM [4]) have been proposed. These click models estimate the relevance of each item in a point-wise manner instead of considering the relative order of the items as in pairwise or listwise approaches. Recently, a new line of research, referred to as counterfactual methods, utilizes inverse propensity score (IPS) weighting to address position bias in a learning to rank framework. Wang et al. [30] and Joachims et al. [16] proposed the IPW-based framework of debiasing click data in a learning to rank framework. In both works, the propensity estimation relies on randomizing search results displayed to users, which obviously degrades users’ search experience. Considering this, Agarwal et al. [1] proposed PBM to estimate propensity without Intrusive Interventions. CPBM [10], on the basis of PBM, learns a query-dependent propensity estimation. However, multiple rankers are required to learn, which makes them inconvenient to deploy in real-world applications. Besides, another branch of unbiased learning to rank works [2, 14, 31] jointly learn the propensity model with a relevance model, which results in biased estimation of propensity unless the relevance estimation is very accurate.

3 PROBLEM FORMULATION

When a user issues a new request q , the system delivers a ranked list $(f_i, b_i)_{i=1}^{n_q}$ of n_q items to the user according to a ranking model over all the candidate items. The feature vector f_i of each item i consists of item features, context features, and user/query features. The scalar b_i denotes the utility value related to each item i , e.g., the watch time of each video in video recommendation, or the bid price of each ad in sponsored search.

The users’ click logs are a set $S = \{(f_i, k_i, b_i, c_{i,k_i})_{i=1}^{n_q}\}_{q \in Q}$, where k_i is the position of item i , and c_{i,k_i} is the users’ implicit feedback on item i when displaying at position k_i , i.e., $c_{i,k_i} = 1$ for click and $c_{i,k_i} = 0$ for non-click. To distinguish between the position of item i in users’ click logs and in the current ranking model, in the

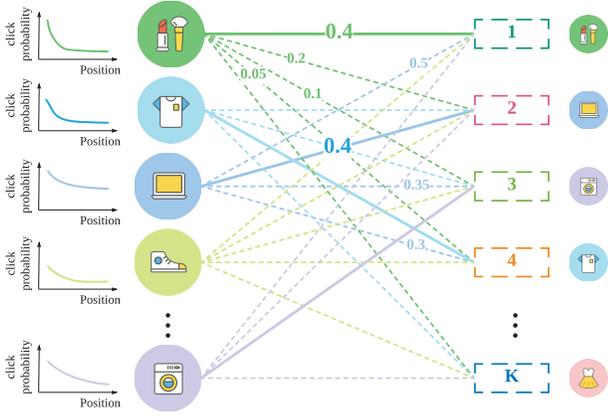


Figure 2: The maximization of the utility can be seen as solving the maximum-weight matching on the item-position bipartite graph, where the edge weight between an item and a position denotes the utility of placing the item at this position, i.e., the product of item’s CTR at this position and the utility value of this item.

following parts, we use k_i^h to denote the position of item i in click logs and keep k_i as the position of item i in the current ranking model.

The ultimate goal of this system is to find the best permutation of candidate items for each query q to maximize the utility. The utility is defined as the expected sum of weighted clicks of each item over the whole ranked list, as follows,

$$U_q = \mathbb{E} \left[\sum_{i=1}^{n_q} c_{i,k_i} \cdot b_i \right] = \sum_{i=1}^{n_q} P(c_{i,k_i} = 1) \cdot b_i, \quad (1)$$

where $P(c_{i,k_i} = 1)$ is the probability of the item i being clicked if displayed at position k_i . Maximizing utility U_q is equivalent to solving a maximum weight matching problem on the item-position bipartite graph, where $P(c_{i,k_i} = 1) \cdot b_i$ is the edge weight between the item i and the position k_i in the graph, as shown in Figure 2.

4 MODEL FRAMEWORK

In this section, we present a general ranking framework, i.e., U -rank, to maximize the utility U_q in Eq. (1) directly. Firstly, we derive an unbiased metric of utility from click logs, the expectation of which is the utility U_q . Secondly, we design an efficient learning to rank method to optimize this metric, of which the loss function is an upper bound of the utility regret.

4.1 Unbiased Estimation of the Utility

The main difficulty of existing methods of learning to rank via implicit feedback lies in the estimation of the underlying attention bias (or position bias), since we do not observe them directly from the data. With the new learning objective, we do not need to infer relevance or the attention bias explicitly. Instead, we have to deal with another mismatch problem, which is between the CTR of the historical presented position and that of the final presented position. For example, if one item is ranked first in the click logs but presented at the 10th position in the final ranking, then its utility

is overestimated. To correct this bias, we need an accurate model of user’s CTR on different positions.

The estimation of CTR on different position $P(c_{i,k_i} = 1)$ refers to one of the most well-studied tasks in recommender systems, i.e., CTR estimation. Deep CTR models [26, 34] can take position and the rich query-item features as input, to model the complex user interaction in feature space from the click logs. It is pointed out that position k_i is a very important feature in CTR estimation [3, 13]. However, if we directly used a CTR estimation model as the ranking model, the position feature is vacant at the inference stage. Therefore, we design a position-aware deep CTR model as a debiasing module instead of directly using it as a ranking model. Assume that the probability function of item i displayed at position k_i is a function g_θ of item feature f_i and position k_i , i.e., $P(c_{i,k_i} = 1) = g_\theta(f_i, k_i)$. Then we can estimate the parameter θ via the standard cross-entropy minimization:

$$\mathcal{L}_c(\theta) = \sum_{q \in Q} \sum_{i=1}^{n_q} l(c_{i,k_i}, g_\theta(f_i, k_i)), \quad (2)$$

where $l(p, q) = -p \log q - (1 - p) \log(1 - q)$ is the cross-entropy loss.

Based on users’ click logs and the estimated CTR, we derive an unbiased metric of utility U_q as

$$U'_q = \sum_{i=1}^{n_q} c_{i,k_i^h} \cdot \frac{P(c_{i,k_i} = 1)}{P(c_{i,k_i^h} = 1)} \cdot b_i. \quad (3)$$

We prove that our derived utility U'_q is unbiased w.r.t. U_q in Eq. (1), by showing that the expectation of U'_q is equivalent to U_q , as

$$\begin{aligned} \mathbb{E}[U'_q] &= \mathbb{E} \left[\sum_{i=1}^{n_q} c_{i,k_i^h} \cdot \frac{P(c_{i,k_i} = 1)}{P(c_{i,k_i^h} = 1)} \cdot b_i \right] \\ &= \sum_{i=1}^{n_q} P(c_{i,k_i^h} = 1) \cdot \frac{P(c_{i,k_i} = 1)}{P(c_{i,k_i^h} = 1)} \cdot b_i \\ &= \sum_{i=1}^{n_q} P(c_{i,k_i} = 1) \cdot b_i = U_q. \end{aligned} \quad (4)$$

4.2 Learning to Optimize the Utility

One straightforward way to optimize U_q is to perform maximum-weight matching algorithm (e.g., Kuhn-Munkres algorithm [19, 25]) on the bipartite graph directly (each query corresponds to one graph), given $g_\theta(f_i, k_i)$. However, the complexity for such a graph matching algorithm to run in the inference stage is $O(N^3)$ (N denotes the number of candidate items), which is unacceptable in a production system. Therefore, in this section, we propose a parameterized scoring function $\Phi(\cdot)$ to approximate the maximum-weight matching procedure on each query, still aiming at maximizing the utility, so that the complexity at the inference stage can be reduced to $O(N)$. For each item i , the scoring function $\Phi(\cdot)$ gives a ranking score as $s_i = \Phi(f_i, b_i)$. For each query q , we compute the score s_i of each item i , and the result list is generated by sorting their scores in descending order.

According to Eq. (3), we define the utility of displaying item i at position k_i as $u(i, k_i) = c_{i,k_i^h} \cdot \frac{P(c_{i,k_i} = 1)}{P(c_{i,k_i^h} = 1)} \cdot b_i$. With k_i^* being the

optimal position assigned to item i by the graph matching algorithm, the regret of the utility is defined as

$$\mathcal{L}_r(\Phi, q) = \sum_{i=1}^{n_q} u(i, k_i^*) - \sum_{i=1}^{n_q} u(i, k_i). \quad (5)$$

Minimizing the regret of the utility $\mathcal{L}_r(\Phi, q)$ directly is infeasible as k_i 's are discrete values. Therefore, we adapt the LambdaLoss framework [32] to learn a ranking model towards the optimal ranking by optimizing our proposed loss function (which will be presented in Eq. (7)) with iterative pairwise permutation. Like in LambdaLoss we follow an EM procedure that in E step we obtain the ranked list based on current scoring function $\Phi^{(t)}$ and in M step we re-estimate the scoring function $\Phi^{(t+1)}$ to minimize our loss function. The learning procedure of U -rank is as follows.

We first initialize the scoring function with random initialization of $\Phi^{(0)}$. Inspired by the re-weighting technique used in LambdaRank [6], we compute the difference between the unbiased utility $\Delta Util(i, j)$ when the positions of two items i and j are swapped, as

$$\Delta Util(i, j) = u(i, k_j) + u(j, k_i) - u(i, k_i) - u(j, k_j). \quad (6)$$

Then this difference value is used as the weight in the pairwise loss for each pair of items. Following [5, 6], we design our loss function in the form of logistic loss, as

$$\mathcal{L}'_r(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j:k_j < k_i} \Delta Util(i, j) \log \left(1 + e^{-\sigma(s_i - s_j)} \right), \quad (7)$$

where k_i and k_j denote the position assigned to item i and j by ranking model at the last step (by the scoring function $\Phi^{(t)}$). This loss is minimized, so that we get a new scoring function $\Phi^{(t+1)}$. Then the process is repeated until convergence.

Notice that in a standard LambdaLoss framework, the LambdaLoss is defined as

$$\mathcal{L}_\lambda(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j:y_i > y_j} |\Delta NDCG(i, j)| \log \left(1 + e^{-\sigma(s_i - s_j)} \right). \quad (8)$$

Note that the differences between our objective (7) and the LambdaLoss objective (8) lie in (i) the subscript of the summation symbol and (ii) the absolute value symbol of the difference term Δ . In LambdaLoss framework, the pairwise label of each item pair (i, j) is determined. The optimal ranking order is known to us by ranking all the items according to relevance label or click label (denoted by y_i for item i), in descending order. However, in our framework, we cannot obtain the explicit label y_i for item i . An item is treated as the positive item if it is placed at a lower position by scoring function $\Phi^{(t)}$ and the swap of the item pair brings utility gain, and vice versa. We cannot access the optimal ranking order in each query beforehand, where the optimal order is achieved through iterative pairwise permutation.

4.3 Theoretical Analysis

In this section, we theoretically prove that our proposed training objective $\mathcal{L}'_r(\Phi, q)$ is an upper bound of the utility regret $\mathcal{L}_r(\Phi, q)$. To make the proof easier to understand, we construct a function:

$$\mathcal{L}''_r(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j:s_i < s_j} |u(i, k_j) - u(i, k_i)|. \quad (9)$$

We start with several lemmas which will be used in our proof.

LEMMA 4.1. *Given an indicator function $f(x) = \mathbb{I}(x \leq 0)$ and a function $g(x) = \log(1 + e^{-\sigma x})$ where σ is a constant in \mathbb{R} , it holds that $f(x) \leq g(x)$ for all $x \in \mathbb{R}$.*

LEMMA 4.2. *Given an indicator function $f(x) = \mathbb{I}(x \geq 0)$ and a function $g(x) = \max\{\log(1 + e^{\sigma C}), 2\} - \log(1 + e^{-\sigma x})$ where σ is a constant in \mathbb{R} , it holds that $f(x) \leq g(x)$ for $x \in [-C, C]$.*

LEMMA 4.3. *Given a sum function $f(x) = \sum_i x_i$ and a max function $g(x) = \max_i x_i$, it holds that $f(x) \geq g(x)$ for $x_i \geq 0, \forall i$.*

Now we are ready to derive the main theoretical result.

THEOREM 4.4. *Assume $u(i, k_i)$ is a monotonic decreasing function w.r.t k_i and the ranking score s_i is bounded in the range of $[-C, C]$. Let $C_1 = \max\{\log(1 + \exp(2\sigma C)), 2\}$ and $C_2 = C_1 \cdot \sum_{j=1}^{n_q} \sum_{i:k_i > k_j} (u(j, k_j) - u(j, k_i))$. Then we have $\mathcal{L}''_r(\Phi, q) \leq \mathcal{L}'_r(\Phi, q) + C_2$.*

PROOF.

$$\begin{aligned} \mathcal{L}''_r(\Phi, q) &= \sum_{i=1}^{n_q} \sum_{j=1}^{n_q} |u(i, k_j) - u(i, k_i)| \mathbb{I}(s_i < s_j) \\ &= \sum_{i=1}^{n_q} \sum_{j:k_j < k_i} (u(i, k_j) - u(i, k_i)) \mathbb{I}(s_i < s_j) + \sum_{j=1}^{n_q} \sum_{i:k_i > k_j} (u(j, k_j) - u(j, k_i)) \mathbb{I}(s_j < s_i) \\ &\leq \sum_{i=1}^{n_q} \sum_{j:k_j < k_i} (u(i, k_j) - u(i, k_i)) \log \left(1 + e^{-\sigma(s_i - s_j)} \right) \\ &\quad - \sum_{i=1}^{n_q} \sum_{j:k_j < k_i} (u(j, k_j) - u(j, k_i)) [\log \left(1 + e^{-\sigma(s_i - s_j)} \right) + C_1] \\ &= \sum_{i=1}^{n_q} \sum_{j:k_j < k_i} \Delta Util(i, j) \log \left(1 + e^{-\sigma(s_i - s_j)} \right) + C_2 \quad (10) \\ &= \mathcal{L}'_r(\Phi, q) + C_2 \quad (11) \end{aligned}$$

where the inequality holds due to Lemma 4.1 and Lemma 4.2. \square

Theorem 4.4 states that $\mathcal{L}''_r(\Phi, q)$ is upper bounded by our objective $\mathcal{L}'_r(\Phi, q)$ plus C_2 . C_2 is a constant since in the M step C_2 only depends on the current scoring function $\Phi^{(t)}$. Notice that the assumptions in the theorem are not restrictive in practice. As illustrated in Figure 1, the real utility basically satisfies the monotonic decreasing assumption. Moreover, the ranking score is often clipped in implementation to avoid explosion in exponential function.

THEOREM 4.5. *Assume the utility $u(i, k_i)$ is a monotonic decreasing function w.r.t k_i . Then $\mathcal{L}_r(\Phi, q)$ is upper bounded by $\mathcal{L}''_r(\Phi, q)$.*

PROOF.

$$\begin{aligned} \mathcal{L}''_r(\Phi, q) &= \sum_{i=1}^{n_q} \sum_{j:s_i < s_j} |u(i, k_j) - u(i, k_i)| = \sum_{i=1}^{n_q} \sum_{k=1}^{k_i-1} |u(i, k) - u(i, k_i)| \\ &= \sum_{i=1}^{n_q} \sum_{k=1}^{k_i-1} (u(i, k) - u(i, k_i)) \geq \sum_{i=1}^{n_q} u(i, 1) - \sum_{i=1}^{n_q} u(i, k_i) \\ &\geq \sum_{i=1}^{n_q} u(i, k_i^*) - \sum_{i=1}^{n_q} u(i, k_i) = \mathcal{L}_r(\Phi, q) \quad (12) \end{aligned}$$

where the first inequality holds due to Lemma 4.3. \square

Table 1: Comparison of different unbiased learning to rank models on three benchmark datasets.

Ranking model		Yahoo! LETOR set 1				MSLR-WEB10K				Istella-S LETOR			
		MAP	nDCG	# Click	CTR	MAP	nDCG	# Click	CTR	MAP	nDCG	# Click	CTR
SVMRank	None	0.702	0.845	0.599	0.0641	0.498	0.735	0.834	0.0827	0.773	0.808	0.931	0.0939
	Randomization	0.639	0.787	0.544	0.0574	0.433	0.686	0.820	0.0799	0.742	0.787	0.909	0.0910
	CPBM	0.701	0.843	0.594	0.0637	0.477	0.721	0.751	0.0746	0.752	0.793	0.923	0.0919
	<i>Groundtruth</i>	<i>0.718</i>	<i>0.859</i>	<i>0.612</i>	<i>0.0656</i>	<i>0.515</i>	<i>0.748</i>	<i>0.872</i>	<i>0.0870</i>	<i>0.775</i>	<i>0.816</i>	<i>0.958</i>	<i>0.0952</i>
LambdaRank	None	0.700	0.847	0.606	0.0641	0.498	0.736	0.834	0.0828	0.776	0.810	0.945	0.0941
	Randomization	0.680	0.828	0.582	0.0621	0.451	0.700	0.813	0.0808	0.748	0.793	0.924	0.0923
	CPBM	0.718	0.857	0.613	0.0651	0.514	0.744	0.836	0.0836	0.779	0.813	0.941	0.0941
	<i>Groundtruth</i>	<i>0.719</i>	<i>0.859</i>	<i>0.618</i>	<i>0.0657</i>	<i>0.521</i>	<i>0.748</i>	<i>0.882</i>	<i>0.0883</i>	<i>0.781</i>	<i>0.815</i>	<i>0.948</i>	<i>0.0948</i>
DNN	DLA	0.639	0.782	0.553	0.0589	0.430	0.682	0.830	0.0839	0.676	0.703	0.828	0.0824
	CTR-1	0.647	0.792	0.551	0.0577	0.477	0.722	0.829	0.0814	0.733	0.771	0.894	0.0895
U-rank		0.719	0.861*	0.618*	0.0659*	0.492	0.725	0.903*	0.0915*	0.783*	0.816*	0.959*	0.0953*
<i>KM (oracle model)</i>		<i>0.935</i>	<i>0.987</i>	<i>0.684</i>	<i>0.0737</i>	<i>0.710</i>	<i>0.723</i>	<i>0.976</i>	<i>0.0969</i>	<i>0.993</i>	<i>0.995</i>	<i>1.132</i>	<i>0.1126</i>

* denotes statistically significant improvement (measured by t-test with p-value < 0.05) over all baselines. Note: baselines in italic is not included.

THEOREM 4.6. *Under the assumption of Theorem 4.4 and Theorem 4.5, we have that $\mathcal{L}_r(\Phi, q) \leq \mathcal{L}_r^t(\Phi, q) + C_2$.*

The proof of Theorem 4.6 is trivial due to Theorem 4.4 and Theorem 4.5. Theorem 4.6 demonstrates that the utility regret $\mathcal{L}_r(\Phi, q)$ is bounded by our proposed objective $\mathcal{L}_r^t(\Phi, q)$ plus a constant. It implies that optimizing our proposed objective is actually minimizing the upper bound of the utility regret, which guarantees the effectiveness of *U-rank* theoretically.

5 SEMI-SYNTHETIC EXPERIMENTS

The semi-synthetic setup is widely applied in unbiased learning to rank [10] which allows us to explore different settings ¹.

5.1 Datasets

- **Yahoo! LETOR set 1**² is used in Yahoo! Learning-to-Rank Challenge. The dataset consists of 700 features normalized in [0, 1], which are extracted from query-document pairs.
- **MSLR-WEB10K**³ is a large-scale dataset released by Microsoft Research. It contains 10,000 queries and 1,200,193 documents. There are 136 features extracted from query-document pairs.
- **Istella-S LETOR**⁴ [23] is one of the largest public available datasets. Istella-S is composed of 33,018 queries, where for each query-document pair there are 220 features.

5.2 Click Data Generation

We mainly follow Fang et al. [10] to generate synthetic click data with item-specific attention bias for the three datasets. In the following part, *oracle model* refers to this click generation model. Similar to [10], the attention bias which is related to both position and the item feature is calculated by $P(o_i = 1 | k_i, x_i) = 1/k_i^{\max(w^\top x_i + 1, 0)}$. In our setting, x_i is the set of item features, while in the setting of [10], x_i is the set of query features which is shared among all the items for a same query. The parameter vector w is drawn from a uniform distribution over $[-\eta, \eta]$ and is normalized such that $\sum_{j=1} w_j = 0$. Following Hu et al. [14], the relevance probability is defined as $P(r_i = 1) = \epsilon + (1 - \epsilon) \frac{2^{y_i} - 1}{2^{y_{\max}} - 1}$, where y_i denotes the relevance label of x_i and y_{\max} is the highest level of relevance. ϵ

is set to 0.1, which denotes the CTR of irrelevant documents. The overall CTR is calculated by $P(c_i = 1) = P(o_i = 1) \cdot P(r_i = 1)$. The maximal position k_{\max} is set to be 10.

5.3 Baselines

We implement eight baselines that explore the performance of two standard learning to rank methods (i.e., **SVMRank** [15] and **LambdaRank** [6]), with four propensity estimation methods, which are detailed as follows. (1) **None** uses the original click data without debiasing. (2) **Randomization** [16] estimates propensity with online randomized experiments. (3) **CPBM** [10] estimates examination probability w.r.t different queries based on intervention harvesting. (4) **Groundtruth** uses the groundtruth examination probability for *oracle model* as propensity. The result of this method is the upper bound of the results of other IPS approaches based on the same ranking model. Other methods we implement include (5) **CTR-1**, the position-aware click model used in our framework which assigns position 1 to each item during online inference. (6) **DLA** [2] is a dual learning algorithm that jointly learns an unbiased ranker and an unbiased propensity model. We also explore the performance of **KM (oracle model)** which solves the maximum-weight graph matching problem via Kuhn-Munkres (KM) algorithm [19, 25], given the groundtruth CTR. It is supposed to produce the best utility we can achieve on the test data.

5.4 Overall Performance

In this section, we assume the utility value of each item to be 1 in order to consistently and fairly compare *U-rank* with existing (unbiased) learning to rank methods. We evaluate the performance of the baseline approaches and *U-rank* in terms of the relevance based metrics, i.e., *MAP* and *nDCG* (*nDCG* denotes *nDCG@10*), and utility based metrics, i.e., *# Click* and *CTR*. Here, The *# Click* and *CTR* are utility metrics based on oracle click model denoting *clicks per query* and *CTR per document*, respectively. The overall performance on the three benchmark datasets is shown in Table 1. *Firstly*, our method *U-rank* achieves consistently the best performance over the state-of-the-art baseline approaches on the utility-based metrics, i.e., *#Click* and *CTR*. For example, *U-rank* achieves 1% improvement in Yahoo LETOR set 1 and 8.3% improvement in MSLR-WEB10K on *CTR* comparing to the best baseline methods ⁵. *Secondly*, *U-rank*

¹Code for our experiments is available at <https://github.com/xydaisjtu/U-rank>

²<https://webscope.sandbox.yahoo.com>

³<https://www.microsoft.com/en-us/research/project/mslr/>

⁴<http://quickrank.isti.cnr.it/istella-dataset/>

⁵The baseline in italic use the information from oracle click model, so they are not included for comparison.

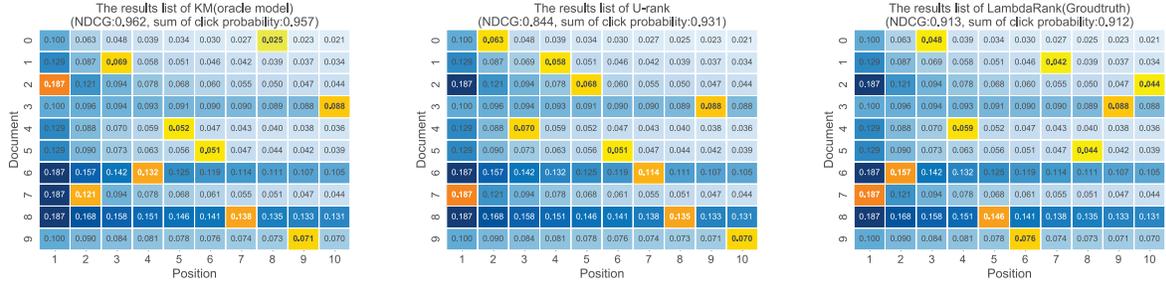


Figure 3: Comparison of the result lists of different methods on the first query of MSLR-WEB10K.

also outperforms most of the baselines in terms of the relevance based metrics, i.e., MAP and $nDCG$, though it does not always perform the best especially on MSLR-WEB10K dataset where the disagreement between utility-based metric and relevance-based metric is larger than that on the other two datasets. *Thirdly*, the method *Groundtruth* achieves the best utility among the counterfactual learning approaches, demonstrating the effectiveness of the IPS-based framework when the propensity estimation is accurate. Randomization fails to perform well because it assumes that the user’s attention only relates to the position which is not true in our setting where the user’s attention relies on both the position and the item feature. CPBM achieves the second-best utility among the IPS-based methods since it models the propensity of each query by taking query features into consideration. *Lastly*, *U-rank* and CTR-1 share the same click model. However, *U-rank* outperforms CTR-1 mainly because CTR-1 ranks items by their estimated CTR at position 1, which is suboptimal in case of item-specific attention bias. *U-rank* also outperforms DLA since DLA relies heavily on the accuracy of the estimated propensity, which is hard to achieve.

5.5 Empirical Analysis

RQ1: How does our model achieve higher CTR? In Figure 4, we show the average CTR on each position of *U-rank* and LambdaRank(Groundtruth), the upper bound of counterfactual learning methods in Table 1. We also plot the results of *KM (oracle model)* for reference.

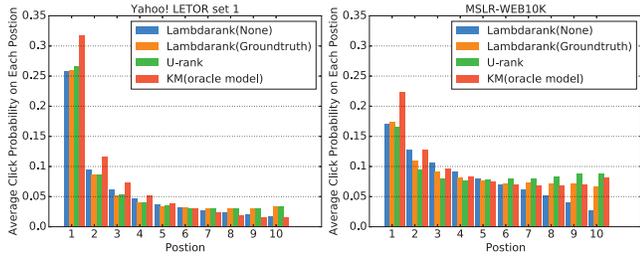


Figure 4: Average CTR on each position.

Firstly, comparing the results of the two datasets in Figure 4, we observe a steeper decline of average CTR to positions of the *KM* (oracle model) method on Yahoo dataset than that on the MSLR dataset. This suggests that on this dataset, positions have a very strong impact on users’ click. Thus, to optimize the utility, a well-performed approach should put more relevant items at higher positions. In MSLR-WEB10K, on the other hand, the average CTR of the optimal matching tends to be equally distributed on the positions, compared to the Yahoo dataset. We find that *our method is adaptive to different*

Table 2: Comparison of two different architecture of the position aware click estimation.

	AUC			# click	CTR
	train	test	validation		
A1	0.695	0.684	0.687	0.903	0.915
A2	0.701	0.693	0.692	0.852	0.847

severity of position bias. In Yahoo dataset, our model focuses more on top positions than LambdaRank, while in the MSLR dataset, our model learns a flatter distribution. Notably, on both datasets, our model achieves a larger sum of click probabilities over all the positions than LambdaRank.

Secondly, we analyze the result of a single query in detail. The experiment is conducted on the first query of the MSLR-WEB10K dataset. Figure 3 shows the click probabilities of the ten items for this query and their click probabilities if placed at each position according to our oracle click data generation model. The position of each item assigned by different methods is denoted in orange color. We can see that although LambdaRank performs better in $nDCG$ with a groundtruth propensity. It, however, achieves a lower CTR than our method *U-rank*. This is because, similar to the *KM* (oracle model), *U-rank* will take the position sensitivity of different items into consideration. For example, document 6 is of high relevance and relatively not sensitive to the position change. LambdaRank displays it at the second position while our method and *KM* both display it at a lower position, so that the second position is kept for an item that is more sensitive to the position change.

RQ2: What kind of architecture should we use to implement the position-aware click estimation? We implement two kinds of architecture for the position-aware click estimation. A1 is a neural network, with the item features as input and a K -dim vector as output, where the k -th dimension denotes the CTR of the item at position k , and K denotes the number of positions. A2 is also a neural network, with the concatenation of item feature and position in one-hot encoding as its input and a single value as output, representing the CTR of the item at the given position. The result on MSLR-WEB10K is presented in Table 2. Although A2 achieves better AUC, we utilize A1 as the click model to pursue higher utility.

6 REAL-WORLD DEPLOYMENT

In order to verify the effectiveness of our proposed model in real-world applications, we conduct experiments in two recommendation scenarios in Company X’s App Store. This App Store has hundreds of millions of daily active users who create hundreds of billions of user logs every day in the form of implicit feedback such as browsing, clicking, and downloading behaviors.

Table 3: Comparison of different unbiased learning to rank models on real-world recommendation scenarios.

Ranking model		Scenario 1 (without bid)				Scenario 2 (with bid)			
		# click	# click@1	# click@3	# click@5	Revenue	Revenue@1	Revenue@3	Revenue@5
SVMRank	None	1.586	0.500	1.107	1.348	3.602	0.959	2.177	2.788
	<i>Groundtruth</i>	1.617	0.536	1.154	1.386	3.619	1.015	2.229	2.825
LambdaRank	None	1.750	0.701	1.327	1.556	3.586	0.964	2.178	2.774
	<i>Groundtruth</i>	1.826	0.781	1.429	1.640	3.637	1.009	2.245	2.837
DLA		1.665	0.624	1.338	1.520	3.631	0.958	2.233	2.827
DeepFM		1.790	0.762	1.379	1.593	3.753	1.131	2.289	2.881
U-rank		1.859*	0.841*	1.474*	1.676*	3.966*	1.264*	2.607*	3.214*

* denotes statistically significant improvement (measured by t-test with p-value<0.05) over all baselines.

6.1 Offline Evaluation

Setups. We conduct offline experiments based on two recommendation scenarios with different utility settings. In Scenario 1, we only consider the downloads of the Apps as the utility, while in Scenario 2, the bid price of each App download needs to be considered. In both scenarios, we use seven consecutive days’ data for training and the eighth day’s data for testing. As in the semi-synthetic experiments, we also implement two LETOR methods, *i.e.*, SVMrank and LambdaRank as baselines. The propensity estimation method *Randomization* is not applicable here since we are not allowed to randomly swap two items of a ranked list in a live recommender system. Similarly, CPBM is not applicable either since in practice we cannot obtain the ranking results of the same user from multiple rankers at the same time. Thus, we only compare *U-rank* with the ranker learned with biased click data, *i.e.*, *None*, and the ranker with groundtruth propensity, *i.e.*, *Groundtruth*. The groundtruth propensity is the same as the propensity that we use in evaluation in the next paragraph. *DeepFM* is included as a baseline as it is the production baseline in this App recommendation online system. It trains with position feature and takes default position 1 in the inference stage. To make a fair comparison, we perform *DeepFM* architecture in both click model and ranking model in *U-rank*.

Metrics. Unlike in the semi-synthetic experiments, here we do not know the underlying user click model. Thus, we have to debias the click data generated by a historical ranker with a pre-estimated propensity to obtain the click signals on the new positions. We estimate the propensity for each category of items from 120 days’ click data on random traffic. This category-wise propensity estimation is a coarse approximation of the groundtruth propensity, which is not available. The propensity is defined as $Q(o_i = 1 | \text{category}_i, k_i)$, where category_i denotes the category of item i . This propensity is only used for evaluation except in the *Groundtruth* methods in Table 3, where this propensity is used for debiasing.

The utility in Scenario 1 is defined as the expected number of debiased clicks at the top- K positions in a session, *i.e.*, $\#click@K = \sum_{k_i=1}^K \frac{Q(o_i=1|\text{category}_i, k_i)}{Q(o_i=1|\text{category}_i, k_i^h)} c_{i, k_i^h}$. We use $\#click$ to denote the case when $K = n_q$. The utility in Scenario 2 is defined as the expected revenue at top- K positions in a session after debiasing, *i.e.*, $revenue@K = \sum_{k_i=1}^K \frac{Q(o_i=1|\text{category}_i, k_i)}{Q(o_i=1|\text{category}_i, k_i^h)} c_{i, k_i^h} \cdot b_i$ where b_i is the bid of item i . We use *Revenue* to denote the case when $K = n_q$.

Results. The overall performance on the two real-world datasets is shown in Table 3. We have the following observations. *Firstly*, *U-rank* achieves the best performance comparing to the state-of-the-art baselines. Specifically, in Scenario 1, *U-rank* achieves 1.8%, 7.7%,

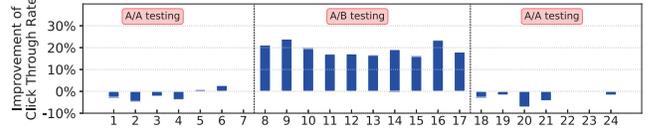


Figure 5: Online experimental results of click through rate.

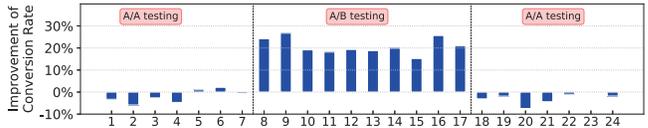


Figure 6: Online experimental results of conversion rate.

3.1% and 2.2% improvement over the best baseline method in terms of $\#click$, $\#click@1$, $\#click@3$ and $\#click@5$, respectively. In the experiment with bid in Scenario 2, the improvements are 2.5%, 13%, 6.7% and 3.9% in terms of *Revenue*, *Revenue@1*, *Revenue@3* and *Revenue@5*, respectively. These results demonstrate the superiority of our approach over the baselines in optimizing the utility, which motivates us to deploy *U-rank* in the live recommender system. *Secondly*, *U-rank* performs better than *DeepFM* because *U-rank* considers item-specific attention bias while *DeepFM* learns from biased data. We do not elaborate on the other results since they are consistent with the results in the semi-synthetic experiments.

6.2 Online Evaluation

Setups. We conduct A/B testing in a recommendation scenario in Company X’s App store, comparing the proposed model *U-rank* with the current production baseline *DeepFM* [12] that supports multiple scenarios such as “Must-have Apps” and “Novel and Fun”. The whole online experiment lasts 24 days, from May 6, 2020 to May 29, 2020. We monitor the results of A/A testing for the first seven days, conduct A/B testing for the following ten days, and conduct A/A testing again in the last seven days. 15% of the users are randomly selected as the experimental group and another 15% of the users are in the control group. During A/A testing, all the users are served by *DeepFM* model [12]. During A/B testing, users in the control group are presented with recommendation by *DeepFM*, while users in the experimental group are presented with the recommendation by our proposed model *U-rank*. Note that the click model of *U-rank* shares the same network architecture and parameter complexity with *DeepFM* in order to verify whether the improvement is brought by the objective function design of the ranker in *U-rank*.

To deploy *U-rank*, we utilize a single node with 48 core Intel Xeon CPU E5-2670 (2.30 GHZ), 400 GB RAM and as well as 2 NVIDIA TESLA V100 GPU cards, which is the same as the training environment of the baseline DeepFM. For model training, *U-rank* requires minor changes to the current training procedure due to the pairwise loss function. For model inference, *U-rank* shares the same pipeline as DeepFM, which means there is no extra engineering work needed in model inference, to upgrade DeepFM model (or other similar deep models) to *U-rank*.

Metrics. We examine two metrics in the online evaluation. They are *Click-through rate*: $CTR = \frac{\#downloads}{\#impressions}$ and *Conversion rate*: $CVR = \frac{\#downloads}{\#users}$, where # downloads, # impressions and #users are the number of downloads, impressions and visited users, respectively.

Results. Figure 5 and Figure 6 show the improvement of the experimental group over the control group with respective to CTR and CVR, respectively. We can see that the system is rather stable where both CTR and CVR fluctuated within 8% during the A/A testing. Our *U-rank* model is launched to the live system on Day 8. From Day 8, we observe a significant improvement over the baseline model with respect to both CTR and CVR. The average improvement of CTR is 19.2% and the average improvement of CVR is 20.8% over the ten days of A/B testing. These results clearly demonstrate the high effectiveness of our proposed model in improving the total utility which refers to the number of downloads in this scenario. From Day 18, we conduct A/A testing again to replace our *U-rank* model with the baseline model in the experimental group. We observe a sharp drop in the performance of the experimental group, which once more verify that the improvement of online performance in the experimental group is indeed introduced by our proposed model.

7 CONCLUSION

In this paper, we propose a novel framework *U-rank*, which directly optimizes the expected utility of the ranked list without any extra assumption on relevance nor examination. Specifically, *U-rank* first uses a position-aware deep CTR model to perform an unbiased estimation of the expected utility, and then optimizes the objective with an efficient algorithm based on a LambdaRank-like objective. Extensive studies on three benchmark datasets and two real-world datasets based on different scenarios have shown the effectiveness of our work. We also deploy this ranking framework on a commercial recommender system and observe a large utility improvement over the production baseline via online A/B testing. In future work, we plan to consider other biases like selection bias and propose a more general debiasing framework.

ACKNOWLEDGEMENT

The corresponding author Weinan Zhang thanks the support of NSFC (61702327, 61772333, 61632017). The work is also sponsored by Huawei Innovation Research Program.

REFERENCES

- [1] Aman Agarwal, Ivan Zaitsev, Xuanhui Wang, Cheng Li, Marc Najork, and Thorsten Joachims. 2019. Estimating Position Bias without Intrusive Interventions. In *WSDM*.
- [2] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W. Bruce Croft. 2018. Unbiased Learning to Rank with Unbiased Propensity Estimation. In *SIGIR*.
- [3] Xiao Bai, Reza Abasi, Bora Edizel, and Amin Mantrach. 2019. Position-aware deep character-level CTR prediction for sponsored search. *TKDE* (2019).
- [4] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A Neural Click Model for Web Search. In *WWW*.
- [5] Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N Hullender. 2005. Learning to rank using gradient descent. In *ICML*.
- [6] Christopher J Burges, Robert Ragno, and Quoc V Le. 2007. Learning to rank with nonsmooth cost functions. In *NeurIPS*.
- [7] Olivier Chapelle and Ya Zhang. 2009. A Dynamic Bayesian Network Click Model for Web Search Ranking. In *WWW*.
- [8] Craswell, Nick, Zoeter, Onno, Taylor, Michael Lyu, Ramsey, and Bill. 2008. An experimental comparison of click position-bias models. In *WSDM*.
- [9] Georges E. Dupret and Benjamin Piwowarski. 2008. A User Browsing Model to Predict Search Engine Click Data from Past Observations. In *SIGIR*.
- [10] Zhichong Fang, Aman Agarwal, and Thorsten Joachims. 2019. Intervention Harvesting for Context-Dependent Examination-Bias Estimation. In *SIGIR*.
- [11] Fan Guo, Chao Liu, Anitha Kannan, Tom Minka, Michael Taylor, Yi-Min Wang, and Christos Faloutsos. 2009. Click Chain Model in Web Search. In *WWW*.
- [12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *IJCAI*.
- [13] Huifeng Guo, Jinkai Yu, Qing Liu, Ruiming Tang, and Yuzhou Zhang. 2019. PAL: A Position-Bias Aware Learning Framework for CTR Prediction in Live Recommender Systems. In *Recsys*.
- [14] Ziniu Hu, Yang Wang, Qu Peng, and Hang Li. 2019. Unbiased LambdaMART: An Unbiased Pairwise Learning-to-Rank Algorithm. In *WWW*.
- [15] Thorsten Joachims. 2006. Training Linear SVMs in Linear Time. In *KDD*.
- [16] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased Learning-to-Rank with Biased Feedback. In *WSDM*.
- [17] Alexandros Karatzoglou, Linas Baltrunas, and Yue Shi. 2013. Learning to rank for recommender systems. In *Recsys*.
- [18] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On application of learning to rank for e-commerce search. In *SIGIR*.
- [19] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* (1955).
- [20] Tie-Yan Liu. 2011. *Learning to rank for information retrieval*. Springer Science & Business Media.
- [21] Lori Lorigo, Maya Haridasan, Hrönn Brynjarsdóttir, Ling Xia, Thorsten Joachims, Geri Gay, Laura Granka, Fabio Pellacini, and Bing Pan. 2008. Eye tracking and online search: Lessons learned and challenges ahead. *Journal of the American Society for Information Science and Technology* (2008).
- [22] Lori Lorigo, Bing Pan, Helene Hembrooke, Thorsten Joachims, Laura Granka, and Geri Gay. 2006. The influence of task and gender on search and evaluation behavior using Google. *Information Processing & Management* (2006).
- [23] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Salvatore Trani. 2016. Post-Learning Optimization of Tree Ensembles for Efficient Ranking. In *SIGIR*.
- [24] Pavel Metrikov, Fernando Diaz, Sebastien Lahaie, and Justin Rao. 2014. Whole Page Optimization: How Page Elements Interact with the Position Auction.
- [25] James Munkres. 1957. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics* (1957).
- [26] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-based neural networks for user response prediction over multi-field categorical data. *TOIS* (2018).
- [27] Richardson, Matthew, Dominowska, Ewa, Ragno, and Robert. 2007. Predicting clicks: Estimating the click-through rate for new ads. In *WWW*.
- [28] Stephen E Robertson. 1977. The probability ranking principle in IR. *Journal of documentation* (1977).
- [29] Yukihiro Tagami, Shingo Ono, Koji Yamamoto, Koji Tsukamoto, and Akira Tajima. 2013. Ctr prediction for contextual advertising: Learning-to-rank approach. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*.
- [30] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *SIGIR*.
- [31] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. In *WSDM*.
- [32] Xuanhui Wang, Cheng Li, Nadav Golbandi, Mike Bendersky, and Marc Najork. 2018. The LambdaLoss Framework for Ranking Metric Optimization. In *CIKM*.
- [33] Liang Wu, Diane Hu, Liangjie Hong, and Huan Liu. 2018. Turning clicks into purchases: Revenue optimization for product search in e-commerce. In *SIGIR*.
- [34] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *ECIR*.
- [35] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumbhakar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *Recsys*.