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

Xinyi Dai\textsuperscript{1}, Jiawei Hou\textsuperscript{1}, Qing Liu\textsuperscript{2}, Yunjia Xi\textsuperscript{1}, Ruiming Tang\textsuperscript{2}, Weinan Zhang\textsuperscript{1}, Xiuqiang He\textsuperscript{2}, Jun Wang\textsuperscript{3}, Yong Yu\textsuperscript{1}

\textsuperscript{1}Shanghai Jiao Tong University, \textsuperscript{2}Huawei Noah’s Ark Lab, \textsuperscript{3}University 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 LambdaMART 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

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 FRP based ranking framework does not necessarily bring the highest utility in reality. To be more specific, FRP 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
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 if it is visually attractive content, like a vertical search result or an item with a fancy thumbnail.

The scalar $\delta_i$ 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 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.

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.

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.

A different type of counterfactual method, referred to as contrastive method, utilizes the concept of “do-experiments” to study the relative order of the items [28]. The idea is to change the order of items and compare the users’ behavior with the original order. However, the downside of this method is that it requires human observation, which is time-consuming and expensive.

3 PROBLEM FORMULATION

When a user issues a new request $q$, the system delivers a ranked list $(f_i, b_i)_{i=1}^n$ 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 $S = \{ (f_i, k_i, b_i, c_{i,k_i})_{i=1}^{n_q} \}$, 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 the current ranking model, in the
The main difficulty of existing methods of learning to rank via presentation at the 10th position in the final ranking, then its utility position. For example, if one item is ranked first in the click logs but relevance or the attention bias explicitly. Instead, we have to deal bias (or position bias), since we do not observe them directly from.

4.1 Unbiased Estimation of the Utility

In this section, we present a general ranking framework, i.e., \( U\text{-rank} \), 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.

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

where \( \mathbb{P}(c_{i,k_i} = 1) \) 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 \( \mathbb{P}(c_{i,k_i} = 1) \cdot b_i \) 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., \( U\text{-rank} \), 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 \( q_{\theta}(f_i, k_i) \).

Then we can estimate the parameter \( \theta \) via the standard cross-entropy minimization:

\[
\mathcal{L}_c(\theta) = \sum_{q \in Q} \sum_{i=1}^{n_q} l \left( c_{i,k_i}, q_{\theta}(f_i, k_i) \right) ,
\]

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

\[
U_q' = \sum_{i=1}^{n_q} c_{i,k_i} \cdot \frac{P(c_{i,k_i} = 1)}{P(c_{i,k_i} = 1)} \cdot b_i .
\]

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

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

\[
= \sum_{i=1}^{n_q} \mathbb{P}(c_{i,k_i} = 1) \cdot \mathbb{P}(c_{i,k_i} = 1) \cdot b_i
\]

\[
= \sum_{i=1}^{n_q} \mathbb{P}(c_{i,k_i} = 1) \cdot b_i = U_q .
\]

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 \( q_{\theta}(f_i, k_i) \). 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 \( k \) as \( u(i, k) = \mathbb{P}(c_{i,k_i} = 1) \cdot \frac{P(c_{i,k_i} = 1)}{P(c_{i,k_i} = 1)} \cdot b_i \). With \( k_i^* \) being the
optimal position assigned to item $i$ by the graph matching algorithm, the regret of the utility is defined as
\[
L_r(\Phi, q) = \sum_{i=1}^{n_q} u(i, k_i^*) - \sum_{i=1}^{n_q} u(i, k_i).
\] (5)

Minimizing the regret of the utility $L_r(\Phi, q)$ 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 $\Phi^{(t)}$. Inspired by the re-weighting technique used in LambdaRank [6], we compute the difference between the unbiased utility $\Delta U\text{til}(i, j)$ when the positions of two items $i$ and $j$ are swapped, as
\[
\Delta U\text{til}(i, j) = u(i, k_j) + u(j, k_i) - u(i, k_i) - u(j, k_j).
\] (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
\[
L_r(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j=k_j<i} \Delta U\text{til}(i, j) \log \left( 1 + e^{-\sigma(u_i - u_j)} \right),
\] (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
\[
L_\Delta(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j:y_j > y_i} \lvert \text{NDCG}(i, j) \rvert \log \left( 1 + e^{-\sigma(u_i - u_j)} \right).
\] (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 $\Delta$. 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 $L_r^*(\Phi, q)$ is an upper bound of the utility regret $L_r(\Phi, q)$. To make the proof easier to understand, we construct a function:
\[
L_r^*(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j=k_j<i} |u(i, k_j) - u(i, k_i)|.
\] (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 $g(x) = \log(1 + e^{-\alpha x})$ where $\alpha$ 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 $g(x) = \max(\log(1 + e^{\alpha C}), 2) - \log(1 + e^{-\alpha x})$ where $\alpha$ 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 $C_1 = \max\{\log(1 + \exp(2\alpha C)), 2\}$ and $C_2 = C_1 \sum_{j=1}^{n_q} \sum_{i,k_j>k_i} u(j, k_j) - u(j, k_i)$. Then we have $L_r^*(\Phi, q) \leq L_r(\Phi, q) + C_2$.

**Proof.**
\[
L_r^*(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j=k_j<i} |u(i, k_j) - u(i, k_i)| \mathbb{I}(s_i < s_j)
\]
\[
\leq \sum_{i=1}^{n_q} \sum_{j=k_j<i} (u(i, k_j) - u(i, k_i)) \mathbb{I}(s_i < s_j) + \sum_{i=1}^{n_q} \sum_{j=k_j<i} (u(j, k_j) - u(j, k_i)) \mathbb{I}(s_i < s_j) + C_1
\]
\[
= \sum_{i=1}^{n_q} \sum_{j=k_j<i} \Delta U\text{til}(i, j) \log \left( 1 + e^{-\sigma(u_i - u_j)} \right) + C_2
\] (10)
\[
= L_r(\Phi, q) + C_2
\] (11)

where the inequality holds due to Lemma 4.1 and Lemma 4.2. □

Theorem 4.4 states that $L_r^*(\Phi, q)$ is upper bounded by our objective $L_r^*(\Phi, q)$ 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 $L_r(\Phi, q)$ is upper bounded by $L_r^*(\Phi, q)$.

**Proof.**
\[
L_r(\Phi, q) = \sum_{i=1}^{n_q} \sum_{j=k_j<i} |u(i, k_j) - u(i, k_i)| = \sum_{i=1}^{n_q} \sum_{k_j=1}^{k_i-1} |u(i, k) - u(i, k_j)|
\]
\[
= \sum_{i=1}^{n_q} \sum_{k_j=1}^{k_i-1} (u(i, k) - u(i, k_j)) \geq \sum_{i=1}^{n_q} u(i, 1) - \sum_{i=1}^{n_q} u(i, k_i)
\]
\[
\geq \sum_{i=1}^{n_q} u(i, k_i^*) - \sum_{i=1}^{n_q} u(i, k_i) = L_r(\Phi, q)
\] (12)

where the first inequality holds due to Lemma 4.3. □
Table 1: Comparison of different unbiased learning to rank models on three benchmark datasets.

| Ranking model | Yahoo! LETOR set 1 | MSLR-WE10K | Istella-S LETOR |
|---------------|-------------------|------------|----------------|
|               | MAP   | nDCG | # Click | CTR | MAP   | nDCG | # Click | CTR | MAP   | nDCG | # Click | CTR |
| SVMRank       | None  | 0.072 | 0.845 | 0.399 | 0.0641 | 0.496 | 0.735 | 0.854 | 0.0827 | 0.773 | 0.808 | 0.931 | 0.0939 |
|               | Randomization | 0.639 | 0.787 | 0.544 | 0.0574 | 0.433 | 0.666 | 0.820 | 0.0799 | 0.742 | 0.787 | 0.909 | 0.0910 |
|               | CBPM  | 0.701 | 0.843 | 0.594 | 0.0635 | 0.477 | 0.721 | 0.751 | 0.0746 | 0.752 | 0.793 | 0.923 | 0.0919 |
|               | Groundtruth | 0.718 | 0.859 | 0.612 | 0.0656 | 0.515 | 0.748 | 0.872 | 0.0870 | 0.775 | 0.816 | 0.958 | 0.0952 |
| 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 |
|               | CBPM  | 0.718 | 0.857 | 0.613 | 0.0651 | 0.514 | 0.744 | 0.836 | 0.0836 | 0.779 | 0.813 | 0.941 | 0.0941 |
|               | Groundtruth | 0.719 | 0.859 | 0.618 | 0.0657 | 0.521 | 0.748 | 0.882 | 0.0853 | 0.781 | 0.815 | 0.948 | 0.0948 |
| DNN           | None  | 0.639 | 0.782 | 0.553 | 0.0589 | 0.436 | 0.682 | 0.830 | 0.0839 | 0.676 | 0.703 | 0.828 | 0.0824 |
|               | U-rank | 0.697 | 0.792 | 0.551 | 0.0577 | 0.477 | 0.722 | 0.829 | 0.0814 | 0.733 | 0.771 | 0.894 | 0.0895 |
|               | KM (oracle model) | 0.719 | 0.861 | 0.618 | 0.0659 | 0.492 | 0.725 | 0.903 | 0.0915 | 0.783 | 0.816 | 0.959 | 0.0953 |

* 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 $L_s^*$, the utility of the query-document pair $(q, x)$, is bounded by our proposed objective $L_q^*$ plus a constant. It implies that minimizing 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.

5.1 Datasets

- **Yahoo! LETOR set 1**: 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-WE10K**: 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**: 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 $P(a_i = 1|k_i, x_i) = 1/k_i^\max(w^WWW x_i^1, 1.0)$. In our setting, $x_i$ is the set of items 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 $\sum_{j=1}^n w_j = 0$. Following Hu et al. [14], the relevance probability is defined as $P(r_i = 1|x_i) = e + (1 - e)w_{\max} - 1$, where $y_i$ denotes the relevance label of $x_i$ and $y_{\max}$ is the highest level of relevance. $\epsilon$ is set to 0.1, which denotes the CTR of irrelevant documents. The overall CTR is calculated by $P(r_i = 1) = P(a_i = 1) \cdot P(r_i = 1)$.

5.3 Baselines

We implement eight datasets 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 inference 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-WE10K on CTR comparing to the best baseline methods. Secondly, $U-rank$...
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.

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     |
|               | Groundtruth | 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     |
|               | Groundtruth | 1.826 | 0.781    | 1.429    | 1.640   | 3.637     | 1.009     | 2.245     | 2.837     |
| DLA           | None    | 1.665    | 0.624    | 1.338    | 1.520   | 3.631     | 0.958     | 2.233     | 2.827     |
| DeepFM        | None    | 1.790    | 0.762    | 1.379    | 1.593   | 3.753     | 1.131     | 2.289     | 2.881     |
| U-rank        | None    | 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.

**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 \( \text{category}_i \) 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 = \( \sum_{K_k=1}^{K} Q(o_i=1|\text{category}_i, k_i) \cdot c_{i, k_i} \). 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., \( \text{Revenue}@K = \sum_{K_k=1}^{K} Q(o_i=1|\text{category}_i, k_i) \cdot c_{i, k_i} \cdot b_i \) 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%,
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 = \( \frac{\# \text{ downloads}}{\# \text{ impressions}} \) and Conversion rate: CVR = \( \frac{\# \text{ downloads}}{\# \text{ users}} \), 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 Weina 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 non-smooth 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 Prowoswalski. 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, Yunnings Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In IJCAI.

[13] Huifeng Guo, Jinjia 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, Qi 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] Shubhika Kanti Karmaker Santu, Pankshun Sondbhi, 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, Hoinn Brynjardottir, Ling Xia, Thorsten Joachims, Geri Gay, Laura Granka, Fabio Fellaccia, and Bing Fan. 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, Sebastian Lahaye, and Justin Rao. 2014. Whole Page Optimization: How Page Elements Interact with the Position Auction. In SIGIR.

[25] James Munkres. 1957. Algorithms for the assignment and transportation problems. Journal of the society for industrial and applied mathematics (1957).

[26] Yanru Qu, Boabu 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] Yuhihiro 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, Duan Hu, Liangjie Hong, and Huan Liu. 2018. Turning clicks into purchases: Revenue optimization for product search in e-commerce. In SIGIR.

[34] Weina Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In ECR.

[35] Zhe Zhao, Lichan Hong, Li Ji, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask system. In RecSys.