Learning-To-Rank with Context-Aware Position Debiasing

Keyi Xiao, Xuezhi Cao, Peihao Huang, Sheng Chen, Xiang Zhou, Yunsen Xian
{xiaokeyi,caoxuezhi,huangpeihao,chensheng19,zhouxiang02,xianyunsen}@meituan.com
Meituan-Dianping Group

ABSTRACT

Learning-to-rank technologies have been widely used for click-through rate prediction tasks and successfully applied in scenarios including web search, advertising, etc. However, users' tendency to click items with higher positions leads to bias problems in traditional learning-to-rank algorithms. There are two crucial problems caused by using such biased data for training and validation, i.e. i) inconsistency between offline evaluation and online performance, and ii) performance dropping due to biased training. For evaluation consistency problem, we conduct analysis to demonstrate its relation towards the position bias problem and alleviate it by using randomized data as validation data. To deal with biased training data, most existing debiasing approaches calculate the bias at each position based on the inverse propensity weighting techniques. However, in complex search scenario where queries have different numbers of retrieved items, click distributions and product styles, the bias distribution over position is quite different for each individual search request, hence can not be precisely captured using simple position-based debiasing methods. In this paper, we propose the Context-Aware Position Debiasing (CAPD) technique to predict accurate bias (both positive bias and negative bias) of each request and do pairwise debiasing on LambdaRank. Experiments demonstrate that our CAPD technique outperforms existing debiasing methods according to NDCG on unbiased randomized validation data and click-through rate in online A/B test.

KEYWORDS

Position Bias Estimation, Inverse Propensity Weighting, Context-Aware Position Debiasing

ACM Reference Format:
Keyi Xiao, Xuezhi Cao, Peihao Huang, Sheng Chen, Xiang Zhou, Yunsen Xian. 2018. Learning-To-Rank with Context-Aware Position Debiasing. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Learning-to-rank [13] (LTR) has been widely used in various domains including search engines, online advertising, recommender systems etc. Existing algorithms of learning-to-rank include pointwise approaches [14], pairwise approaches [2] and list-wise approaches [4]. Among the proposed algorithms, LambdaRank [3, 18] is an effective method to optimize the Normalized Discounted Cumulative Gain (NDCG) metric [10], which is widely used for evaluating the performance of a ranking system. Despite their training methodologies, most LTR methods are affected by the position bias problem introduced by real online user behaviors, e.g., users tend to click items presented in top positions, thus causing the position bias problem [11]. Therefore, study of the position bias problem serves as a fundamental research direction in learning-to-rank.

There are two key problems caused by using such biased data for training and validation. The first one is the evaluation inconsistency problem. When biased data is used for offline validation, which is widely adopted in practice, the bias will lead to the inconsistency between offline evaluation and online performance. The second is the problem of biased training. A satisfying learning-to-rank model should be able to distinguish between the position bias and actual user preferences among the user behaviors. Otherwise, the trained model would be a biased one which can not accurately capture user preferences precisely.

In this paper, we analyze how position bias causes the inconsistency between offline evaluation and online performance. As a result, traditional LambdaRank algorithm to optimize the offline ranking metrics of NDCG might leads to a biased model which is not optimal online. And we show that using randomized data to evaluate could achieve the consistency between NDCG offline and online performance. However, result randomization intuitively degrades the users' search experience. So we only use limited randomized data as our validation data.

To tackle the problem of biased training, researchers proposed plenty works on training unbiased models with biased data. Most existing works are based on the inverse propensity weighting (IPW[7]) method, which treats the bias as a counterfactual effect. Such works include IPW on SVM-Rank [12], selection bias estimation [17] and unbiased LambdaMart [9]. All of these approaches conduct bias estimation and assign a single constant bias for each position regardless of the search request, assuming that position bias does not change according to search requests. However, such assumption does not always hold.

Context-aware position debiasing is further required when dealing complex search scenarios. Specifically, complex search context can be consist of target search domains/scenarios, different user groups, display templates etc. Taking scenarios as an example, user may pay more attention to each displayed item when search for holiday trip hotels , while only browsing the first few candidates when search for nearby fast foods. Moreover, there are plenty application-related display variations which can also affect the actual position bias that user experiences, e.g. display advertisements, promotion campaign cards, related search blocks, etc. (demonstrated in Figure

DLP-KDD 2020, August 24, 2020, San Diego, California, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

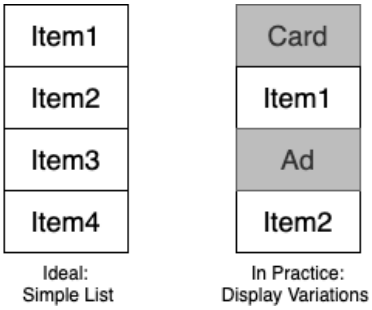

Figure 1: Demonstration of Display Variations in Search.

1). Therefore, the position bias distribution may be quite different under various search context, and context-aware position debiasing is required for better conducting the unbiased training in complex search scenarios.

In this paper, we propose Context-Aware Position Debiasing (CAPD) by following the basic idea of Inverse Propensity Weighting and considering the contextual information for conducting position debiasing. Learning from the pairwise debiasing method in unbiased LambdaMart [9], in which introduce the positive bias of clicked data and negative bias of unclicked data, we propose two bias estimation models in CAPD, one is for positive bias estimation and the other is for negative bias estimation. Specifically, we conduct a query-position-dependent model with pairwise loss training on randomized data to predict bias distribution on each request. And then we conduct the pairwise debiasing on LambdaRank with bias estimated by models. Extensive experiments demonstrate the effectiveness of our CAPD method, which out-performs all comparing bias estimation methods. CAPD has also been successfully adopted for Meituan search, one of the leading online-to-offline (O2O) e-commerce platform in China, achieving significant improvement on online click-through rates.

Our primary contributions of this paper can be summarized as follows:

- We analyze how position bias problem causes the inconsistency between offline evaluation and online performance and propose the unbiased validation method with randomized data.
- We propose a Context-Aware Position Debiasing (CAPD) technique to conduct unbiased training in complex search scenarios. CAPD includes two steps: i) First we conduct two query-position-dependent models with pairwise loss (one is for positive bias estimation and the other is for negative bias estimation), where bias distribution could be estimated with different context pattern. ii) Then we conduct the pairwise debiasing on LambdaRank with positive bias and negative bias.
- We verify the effectiveness of the Context-Aware Position Debiasing (CAPD) technique through rigorous offline experiments and online experiments. And performance online is consistent with offline evaluation on randomized data but inconsistent with offline evaluation on regular data, which is correspond with our analysis of inconsistency problem.

For the rest of this paper, we first introduce the related works in Sec. 2 and formally define the problem in Sec. 3. In Sec. 4, we present the details of CAPD proposed in this paper. Extensive experiments using real search data is explained in Sec. 5. Finally, conclusions and future works are discussed in Sec. 6.

2 RELATED WORK

In search systems, a ranker is usually defined as a function of feature vector based on a query item pair. Given a query, the retrieved items are ranked base on their scores given by the ranker. There are many algorithms proposed in previous work for learning-to-rank, including point-wise algorithm[14], pairwise algorithm[2] and list-wise algorithm[4]. The key issue of ranking in search system is to determine the orders of items in each request, so that the score distribution in each request is much more important than the global score distribution. Thus, pairwise loss and list-wise loss works much better than point-wise loss in learning-to-rank.

In most learning-to-rank algorithms, the training data is collected and labeled by users' click behaviour online. Clicked items are labeled as positive samples and unclicked items are labeled as negative samples in each request. Previous work of Joachims et al. [11] and Yue et al. [19] studies users' tendency of clicking items with good positions and the consequence of position bias problem. Several click models have been developed to solve the bias problem including Position Based Model(PBM) [15], Cascade Model(CM) [6], User Browsing Model(UBM) [8], Dynamic Bayesian Network Model(DBN) [5], etc.

Performance of learning-to-rank approaches is measured by ranking metrics such as Normalized Discounted Cumulative Gain (NDCG) [10], Mean Average Precision(MAP)[1], Mean Reciprocal Rank(MRR)[16], etc. In search systems, users care more about the top results, so that we usually use NDCG as our offline ranking metric. Previous work of LambdaRank algorithm[3, 18] is efficient to optimize NDCG. Thus our work is based on LambdaRank and use NDCG as our ranking metric.

Hu et al. [9] proposed an unbiased pairwise learning-to-rank algorithm based on LambdaRank, in which introduce the positive bias of clicked data and negative bias of unclicked data. The positive bias t_i^+ and negative bias t_j^- are defined as

$$\begin{aligned} P(c_i^+|x_i) &= t_i^+ P(r_i^+|x_i) \\ P(c_j^-|x_j) &= t_j^- P(r_j^-|x_j) \end{aligned} \tag{1}$$

at a clicked position i and an unclicked position j in Unbiased LambdaRank. And we notice that a reasonable estimation result should be the t_i^+ decreasing with i and the t_j^- increasing with j, which is contradictory to the result in Unbiased LambdaMart[9] on t_j^-. We verifies the effectiveness of pairwise debiasing on LambdaRank, and fix the negative bias problem in this paper.

Wang et al. [17] proposed several bias estimation methods including generalized bias model of training n logistic regression models, each for a single position. However, in complicate search scenario, such as Meituan Search, traditional bias estimation methods do not work well because of the significant difference on bias distribution between queries and the discontinuous item positions of each request. In such search scenario, bias distribution in each request is much more important than the global bias distribution.

Thus, in this paper, two context-aware bias estimation models with pairwise loss are proposed, one is for positive bias estimation and the other is for the negative bias estimation, in which focus more on the bias distribution of each request. This model predicts the bias of each position in a request, and it fits well when request has discontinuous item positions because the discontinuity appears in training data and predicting data both. Experiments show that these two bias estimation models achieve significant improvement of online click-through rate.

3 PROBLEM FORMULATION

In this section, we review the general setting of pairwise learning-to-rank algorithm LambdaRank. We then propose the inaccuracy of evaluation when we train the model with online data, which is biased without considering the show position of items. Furthermore, we propose some unbiased validation methods.

3.1 Pairwise Learning-to-Rank: LambdaRank

Let $Q = (q, \{x_1, ..., x_n\})$ denote a request q and its set of result items. Let D denote the set of query-wise data Q. The goal of learning-to-rank is to find a ranking model $f(x)$ to minimize the loss function defined as:

$$L(f) = \sum_{Q \in D} l(Q, f) \tag{2}$$

where $l(Q, f)$ is the loss of $f(x)$ applied to request Q. We then use the y_i as the label and $\hat{y_i} = f(x_i)$ as the score for the i-th item x_i. A commonly used pairwise loss function is the logistic loss

$$l(Q, f) = \sum_{y_i > y_j} log_2(1 + e^{-\sigma(\hat{y_i} - \hat{y_j})}) \tag{3}$$

where σ is a hyper-parameter.

For ranking effectiveness, the evaluation metric in this paper is NDCG, which is commonly used in learning to rank tasks. The NDCG metric for a single query over the item list ranked by decreasing scores $\hat{y_i}$ is defined as

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

LambdaRank uses the logistic loss and adapts it by reweighing each item pair by $\Delta NDCG$ so that the lambdaLoss is defined as

$$l(d, f) = \sum_{y_i > y_j} \Delta NDCG(i, j) log_2(1 + e^{-\sigma(\hat{y_i} - \hat{y_j})}) \tag{5}$$

3.2 Inconsistency Between Offline Evaluation and Online Performance

An observation is that the data set D collected from the online click log is biased as users tend to click items ranked at higher positions.

When there is a fixed ranking strategy online, the model trained from D tends to maintain the ranking of that strategy. A model performing better online may get a smaller NDCG on the validation data from D. Actually, in Meituan Search Scenario, inconsistency between offline evaluation and online performance appears often when doing a model upgrade (e.g. Manual rules to GBDT models, GBDT models to deep models) or introducing some new features.

Such inconsistency makes the offline evaluation useless. With the inconsistency, one have to evaluate all ranking models online, which is very costly.

We use an example to better explain the observation of inconsistency between offline evaluation and online performance. Assume that we have online strategy (x_1, x_2), in which x_1 presented before x_2, and offline strategy (x_2, x_1). Let r_i^+ represent that item x_i is relevant. Let c_i^+ represent that item x_i is clicked. We assume that the click probability is proportional to the relevance probability at each position, where the ratio t_i^+ is the bias at a click position i. [9] So we have

$$P(c_i^+|x_i) = t_i^+ P(r_i^+|x_i) \tag{6}$$

Assuming there are n exposure data online with online strategy (x_1, x_2), we have $t_1^+ P(r_1^+|x_1)n$ clicks of item x_1 and $t_2^+ P(r_2^+|x_2)n$ clicks of item x_2. And we have $m = n(t_1^+ P(r_1^+|x_1) + t_2^+ P(r_2^+|x_2))$ of clicked requests. So that the NDCG of online strategy (x_1, x_2) is

$$NDCG(x1, x2) = \frac{1}{m}(t_1^+ P(r_1^+|x_1) + \frac{t_2^+ P(r_2^+|x_2)}{\log_2 3}) \tag{7}$$

For the offline strategy (x_2, x_1), because the validation data is from online strategy, the click number of item x_i does not change. So we have NDCG of offline strategy (x_2, x_1) is

$$NDCG(x2, x1) = \frac{1}{m}(t_2^+ P(r_2^+|x_2) + \frac{t_1^+ P(r_1^+|x_1)}{\log_2 3}) \tag{8}$$

And we assume that $P(r_2^+|x_2) > P(r_1^+|x_1)$ which means offline strategy (x_2, x_1) is a better strategy. Once $\frac{t_1^+}{t_2^+} > \frac{P(r_2^+|x_2)}{P(r_1^+|x_1)}$, we could get $NDCG(x2, x1) < NDCG(x1, x2)$, which could have an opposite performance when we use strategy (x_2, x_1) online.

3.3 Unbiased Validation Methods

We propose two unbiased validation methods and show how they fix the inconsistency problem.

3.3.1 Weighted NDCG. Inverse Propensity Weighting[7] approach can be adopted to help overcome position bias problem and fix the inconsistency between online and offline. We define a weighted NDCG using the bias ratio of each request. For the clicked position i of request data Q, we define the weight w_Q as the normalization of bias t_i^+. and we define the weighted NDCG of a single request as: $\frac{1}{w_Q} NDCG_Q$. And weighted NDCG of the strategy would be:

$$weightedNDCG = \frac{\sum_{Q \in D} \frac{1}{w_Q} NDCG_Q}{\sum_{Q \in D} \frac{1}{w_Q}} \tag{9}$$

Reviewing the example before, we could get the weighted NDCG of online and offline strategy as follows:

$$\begin{aligned} weightedNDCG(x1, x2) &= \frac{1}{m}(P(r_1^+|x_1) + \frac{P(r_2^+|x_2)}{\log_2 3}) \\ weightedNDCG(x2, x1) &= \frac{1}{m}(P(r_2^+|x_2) + \frac{P(r_1^+|x_1)}{\log_2 3}) \end{aligned} \tag{10}$$

And we have $weightedNDCG(x1, x2) < weightedNDCG(x2, x1)$ which would be consistent with online performance.

However, the accuracy of weighted NDCG depends heavily on t_i^+ of the bias estimation. A model which fits the weighted NDCG

would get a higher weighted NDCG in validation. Thus we need NDCG of randomized data for double check.

3.3.2 Result Randomization. We have explained that the inconsistency between online and offline is due to the biased display online. In the example of Figure 1, if we show the random ranking of half (x1,x2) and half (x2,x1) online, the NDCG of two strategies on the randomized data would also be

$$\begin{aligned} NDCG(x1,x2) &= \frac{1}{m}(P(r_1^+|x_1) + \frac{P(r_2^+|x_2)}{\log_2 3}) \\ NDCG(x2,x1) &= \frac{1}{m}(P(r_2^+|x_2) + \frac{P(r_1^+|x_1)}{\log_2 3}) \end{aligned} \tag{11}$$

Thus in this paper, we use a randomized top-N data set $\mathfrak{R}$ as the validation data set and do the unbiased training on the regular data set D.

4 UNBIASED LEARNING-TO-RANK WITH CONTEXT-AWARE

In this section, we show how to conduct pairwise debiasing on the LambdaRank. What's more, our search scenario is quite complicated so that for different request, even at the same position, the bias could be very different. Thus we propose some effective methods to estimate bias with context-aware.

4.1 Inverse Propensity Weighting on LambdaRank

Reviewing the LamdaRank algorithm [18] with the lambdaLoss which is defined as Eq.5, the lambda gradient λ_i of item x_i is calculated using all pairs of the other items with respect to the request Q

$$\lambda_i = \sum_{j:(x_i,x_j)\in Q} \lambda_{ij} - \sum_{j:(x_j,x_i)\in Q} \lambda_{ji} \tag{12}$$

$$\lambda_{ij} = \frac{-\sigma}{1+e^{\sigma(\hat{y}_i-\hat{y}_j)}}|\Delta NDCG(i,j)| \tag{13}$$

where λ_{ij} is the lambda gradient defined on a pair of item x_i and x_j.

We use the weightedNDCG(Eq.9) instead of NDCG(Eq.4) to be our optimization target, which infers the query-level debiasing on lambda gradient.

4.1.1 Query-level debiasing. We defined the bias ratio t_i^+ as Eq.6. Thus the query-level debiasing of LambdaRank is to weight the lambda gradient λ_i as

$$\hat{\lambda_i} = \frac{1}{t_i^+}(\sum_{j:(x_i,x_j)\in Q} \lambda_{ij} - \sum_{j:(x_j,x_i)\in Q} \lambda_{ji}) \tag{14}$$

when there is a click on the $item_i$ of request Q.

This method fits well when there is a single click per request. But in Meituan Search, requests with multiple clicked items account for half of clicked requests. Thus we need item-level debiasing.

4.1.2 Item-level debiasing. Consider the pair-wise lambda gradient λ_{ij} defined on the pair of clicked item x_i and unclicked item x_j. For the clicked item x_i, the weight of λ_{ij} should be different with different unclicked item x_j. Reviewing the positive bias ratio t_i^+ in Eq.6, we define the negative bias ratio t_j^- as

$$P(c_j^-|x_j) = t_j^- P(r_j^-|x_j) \tag{15}$$

Thus we have the pairwise weighted lambda gradient [9]

$$\hat{\lambda_i} = \sum_{j:(x_i,x_j)\in Q} \hat{\lambda_{ij}} - \sum_{j:(x_j,x_i)\in Q} \hat{\lambda_{ji}} \tag{16}$$

$$\hat{\lambda_{ij}} = \frac{\lambda_{ij}}{t_i^+ t_j^-} \tag{17}$$

We notice that t_i^+ is decreasing with i and the t_j^- is increasing with j. The difference between our work and Unbiased LambdaMart [9] is that we use randomized data to estimate t_i^+ and t_j^-. The negative bias t_j^- learned from the algorithm in Unbiased LambdaMart [9] is decreasing with j, which is contradict to the definition of negative bias.

4.2 Bias Estimation

Our application Meituan Search Engine serves a very complicated scenario and the bias distribution is quite different for each request. For example, different queries has different number of recall items. And for the query words of brand with exact intent, the items are similar and the clicks are focus on the head positions. For the query words of address, the clicks distribution are more dispersed. What's more, devices and display styles also affect bias distributions a lot.

We do the bias estimation on a randomized top30 data because clicks after position 30 is very rare. The distribution of clicks on randomized data is credible for bias estimation while the clicks on regular data is more concentrated on head positions because they have more relative items.

4.2.1 Bias Estimation in Complex Search Scenario. The basic idea of doing bias estimation in complex search scenario is to partition requests into segments and estimate the click distributions for each segment.

In our application, the most important feature affecting the bias distribution is the view-depth of each request. An other important feature is the query intention, which affects the item distribution. In Figure2 and Figure3 we could see how these two factors influence the click distribution.

Thus we calculate the click distribution on each view-depth (e.g. view-depth = 2,3,4..,29,30+) and on each query intention (e.g. brand, address, hotel, etc.). And we have the bias t_i^+ and t_j^- at specific view-depth and query intention as

$$\begin{aligned} t_i^+ &\propto P(click|pos=i, view-depth, queryIntent) \\ t_j^- &\propto 1 - P(click|pos=i, view-depth, queryIntent) \end{aligned} \tag{18}$$

What's more, in our training data set, some queries have advertisement or card insertions before items and in our ranking systems we only rank the items(e.g. Figure 1). Thus we need to do the query-wise normalization instead of normalization of each segment. And

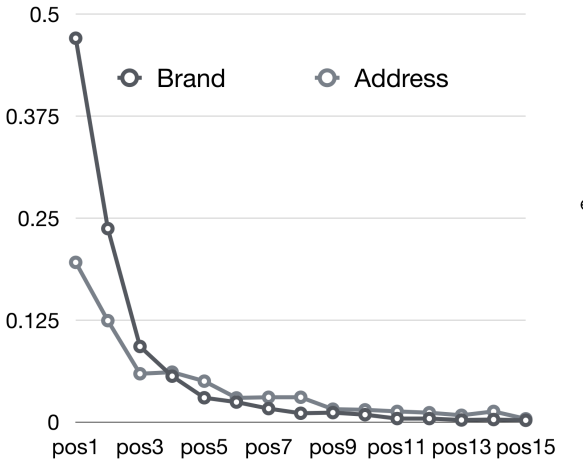

Figure 2: Illustration of CTR at each position of different query intention.

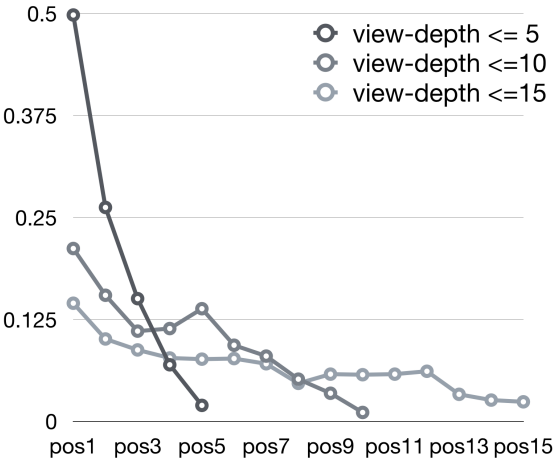

Figure 3: Illustration of CTR at each position of different view-depth.

we define the normalized bias as

$$w_i^+ = \frac{t_i^+}{\max_{x_i \in Q}(t_i^+)}$$
$$w_j^- = \frac{t_j^-}{\max_{x_j \in Q}(t_j^-)} \tag{19}$$

And we do the pairwise inverse propensity weighting on lambda gradient as

$$\hat{\lambda_{ij}} = \frac{\lambda_{ij}}{w_i^+ w_j^-} \tag{20}$$

4.2.2 Bias Estimation Model. An other challenge in Meituan Search is that training data in our search scenario might not be continuous with positions. Card blocks and advertisements are commonly inserted in Meituan Search and the ranking models only rank the items of each request. Figure 1 illustrate the difference between traditional training data and Meituan Search training data. The insertions have some regular patterns for different query words. Generalized bias estimation models [17] of independent model at each position cannot learn about the display pattern of each request. Thus, we propose a query-position-dependent bias estimation model which also consider the display pattern of each request. We defined the bias prediction problem as follows:

Definition 4.1. Position Bias Prediction. Given a request $Q = (q, \{x_1, ..., x_n\}$ and the position i of each item x_i, the problem of Position Bias Prediction is to estimate the click probability and unclick probability of the item at position i when we show the set of items in a random order in each request Q.

We use the queries in the randomized data $\mathfrak{R}$ as our training data. And different from training n models for n positions in [17], we train one model with pairwise loss for all positions to estimate the click probability. Because in our scenario, we concerned more about the bias distribution in each single request. For each request, positions of exposure are quite different in Meituan Search, so one model for one position is unable to satisfy the accurate estimation of bias. Thus we include the positions of items to our features and describe our approach as follows:

- **Data formulation:** For each request $Q = (q, \{x_1, ..., x_n\}$ in randomized data set $\mathfrak{R}$, we define the y_i as the label of each item x_i and the $y_i = 1$ when x_i is clicked and $y_i = 0$ when x_i is not clicked.
- **Features:** For each request Q, we construct a feature vector $v(Q)$. In our setting, the feature can be query-dependent(e.g. query intention, length of query) or user-dependent(e.g. devices, average click position of the user) or item-set-dependent(e.g. recall-num of the item list, view-depth of the item list). And the feature vector of each item x_i in request Q is the concatenation of $(v(Q), i)$, which is only depend on the request and item position.
- **Training:** The goal of our bias estimation model $f(x)$ is to minimize the pairwise loss function (logistic loss)

$$l(Q, f) = \sum_{y_i > y_j} log_2(1 + e^{-\sigma(\hat{y}_i - \hat{y}_j)}) \tag{21}$$

  And we train a nn model with logistic gradient for each pair

$$Grad = \frac{-\sigma}{1 + e^{\sigma(\hat{y}_i - \hat{y}_j)}} \tag{22}$$

- **Prediction:** The output of the model passes through a sigmoid layer so that the predict value is between 0 and 1. We define the $P(ct_i^+)$ of request Q as $t_i^+ = f(v(Q), i)$, and the normalized bias $w_i^+ = \frac{t_i^+}{\max_{x_i \in Q}(t_i^+)}$

Similarly to train the model of positive bias t_i^+, we also train a model of negative bias t_j^-, in which just exchange the labels as the $y_j = 0$ when x_j is clicked and $y_j = 1$ when x_j is not clicked. We define this model as $f^-(v(Q), j)$ and we have the negative bias $t_j^- = f^-(v(Q), j)$ of request Q at position j. Thus we have the normalized negative bias $w_j^- = \frac{t_j^-}{\max_{x_j \in Q}(t_j^-)}$.

Finally we use the bias estimated from models to do the pairwise inverse propensity weighting on lambda gradient as Eq.20

Table 1: Features used in CAPD

Feature	Description
totalCount	total count of recall items in each request
queryIntent	type id of the query(brand=0, address=1,...)
queryClickPosAvg	average click position of the query
queryLength	length of query
viewdepth	user's view depth of the request
device	iPhone, iPad, Android...
itemPosition	position of the item
offset	page offset of the item

5 EXPERIMENTS

In this section, we conduct experiments on two steps of Context-Aware Position Debiasing technique. One is the experiment set of bias estimation methods, in which we compare the effectiveness of global bias estimation method, statistic bias estimation method on different segment, and a query-position-dependent bias estimation model. The other is the experiment set of debiasing methods including query-level debiasing method, item-level debiasing method with only positive bias, and item-level debiasing method with positive bias and negative bias.

Randomized data is used as our validation data and we also pay attention to the NDCG on regular data and CTR online, which shows that the NDCG in randomized data is positively correlated with the online click-through rate while the NDCG in regular data is not.

5.1 Experimental Design

We conduct the experiments using search log from Meituan search. The dataset D covers 15M users, 80M items, and 200M impressions among them. The data is segmented into training data D_t and evaluation set D_e according to date.

To conduct the unbiased validation and to estimate bias in CAPD, we also collect a randomized data set $\mathfrak{R}$. For each request, we give users a random ranking of top 30 items with a probability of one in a thousand. And we collected 300K clicked queries of random ranking in same time period of regular data set.

Both regular and randomized validation sets D_e and $\mathfrak{R}$ are used during evaluation to demonstrate the correlation/inconsistency between different offline evaluation settings and the actual online performance.

We perform two sets of experiments on unbiased LambdaRank. One is to compare the effectiveness of different bias estimation methods. The other is to compare different debiasing methods on LamdaRank. What's more, we verify the difference of NDCG between regular data and randomized data and confirm our analysis on the consistency between NDCG on randomized data and click-through rate online.

The experiment of different debiasing methods is summarized in Table 3 and we use the segmented statistic bias in Table 2 as our bias estimation method. The result shows that item-level debiasing with t_i^+ and t_j^- has the best performance on randomized data.

Thus we conduct the experiment of different bias estimation methods in Table 2 on item-level debiasing with t_i^+ and t_j^-, which means we need to estimate t_j^- in this experiment. What's more, Table 1 shows the features we used in bias estimation model.

Table 2: List of Bias Estimation Methods

Estimation Method	Description
Global Statistic	The bias is estimated for each position globally
Segmented Statistic	The bias is estimated for each position per segment with different view-depth and queryIntent
Query-wise Normalization of Segmented Statistic	The bias is estimated for per segment and normalized in each request
Query-wise Bias Estimation Model	The positive bias and negative bias are estimated by two NN models with pairwise logistic loss and normalized in each request.

Table 3: List of debiasing methods.

Debiasing Method	Description
No Weight	No bias correction on LambdaRank with regular data. This servers as our baseline.
Query-level Debiasing	We use the normalized bias weight $w_Q = \frac{t_i^+}{\max_{x_i \in Q}(t_i^+)}$ of the first clicked item x_i in a request Q. And we do the inverse propensity weighting on lambda gradient as $\hat{\lambda_i} = \frac{\lambda_i}{w_Q}$
Item-level Debiasing with only t_i^+	We use the normalized bias weight $w_i^+ = \frac{t_i^+}{\max_{x_i \in Q}(t_i^+)}$ of the each clicked item x_i in a request Q. And we do the inverse propensity weighting of pairwise lambda gradient as $\hat{\lambda_{ij}} = \frac{\lambda_{ij}}{w_i^+}$
Item-level Debiasing with t_i^+ and t_j^-	We use the normalized bias weight $w_i^+ = \frac{t_i^+}{\max_{x_i \in Q}(t_i^+)}$ and $w_j^- = \frac{t_j^-}{\max_{x_j \in Q}(t_j^-)}$ of the each clicked item x_i and unclicked item x_j in a request Q. And we do the inverse propensity weighting of pairwise lambda gradient as $\hat{\lambda_{ij}} = \frac{\lambda_{ij}}{w_i^+ w_j^-}$

5.2 Experimental Results

In this section, we present the results on two sets of experiments. The ranking metrics of NDCG are calculated on regular data D_e and randomized data $\mathfrak{R}$ for each experiment.

Table 4: Comparison of different bias estimation methods

Estimation Method	NDCG on $\mathfrak{R}$	NDCG on D_e	CTR online
Base with noDebiasing	0.77472	0.81023	- BASE -
Global Statistic	0.77457	0.80891	-2.0‰
Segmented Statistic	0.77479	0.80922	+0.5‰
Query-wise Normalization of Segmented Statistic	0.77487	0.80958	+1.0‰
Query-Position-Dependent Bias Estimation Model	0.77514	0.80932	+2.3‰

Table 5: Comparison of different debiasing methods

Debiasing Method	NDCG on $\mathfrak{R}$	NDCG on D_e	CTR online
Base with noDebiasing	0.77472	0.81023	- BASE -
Query-level Debiasing	0.77476	0.80933	+0.2‰
Item-level Debiasing with t_i^+	0.77497	0.80951	+1.5‰
Item-level Debiasing with t_i^+ and t_j^-	0.77514	0.80932	+2.3‰

In the experiment of bias estimation methods, we estimate both $P(t_i^+)$ and negative bias t_j^- for item-level debiasing. Table 4 summarized the comparison of different bias estimation methods. One observation is that the global bias statistic method cannot beat the base because bias distribution is quite different in each request of complex search scenario in Meituan App and the inaccuracy of bias estimation could mislead the unbiased training. The result shows that the query-position-dependent bias estimation model has the best performance on randomized data $\mathfrak{R}$ and achieves a 2.3‰improvement on CTR online.

In the experiments of debiasing methods, we use a same set of t_i^+ and t_j^- estimated by the query-position-dependent bias estimation model for variable control. Table 5 summarized the comparison of different debiasing methods. Results verifies that our bias estimation model works well in each debiasing methods. The result of item-level debiasing with only t_i^+ has a 1.5‰improvement on CTR. Further more, the item-level debiasing with t_i^+ and t_j^- has the best performance on randomized data $\mathfrak{R}$ and achieves a 2.3‰improvement on CTR online.

Another observation of our experiments is that every debiasing method achieves lower NDCG on regular data D_e but higher NDCG on randomized data $\mathfrak{R}$ than base. And two sets of experiments both verifies the consistency between NDCG on randomized data and CTR online, which is correspond with our analysis of inconsistency problem.

We conclude our findings of the experiments as follows:

- The item-level debiasing method with t_i^+ and t_j^- achieves a better performance rather than other debiasing methods.
- The traditional bias estimation methods do not work well in complex search scenario.
- The query-position-dependent bias estimation model with context-aware achieves the best performance of all bias estimation methods on randomized data $\mathfrak{R}$ and a 2.3‰improvement on CTR online.
- Debiasing methods usually achieve a better performance on randomized data $\mathfrak{R}$ but a worse performance on regular data D_e. And the performance on randomized data $\mathfrak{R}$ is consistent with CTR online while the the performance on regular data D is not.

6 CONCLUSION

In this paper, we target at alleviating position bias problem in learning-to-rank for complex search scenarios. Position bias affects the training process of all kinds of learning-to-rank models, leading to the evaluation inconsistency problem and sub-optimal ranking models. To alleviate the inconsistency problem, we introduce the weighted NDCG method and ranking list randomization method. And we use the randomized data as our validation data set in this paper. For unbiased training, existing debiasing techniques do not take contextual information into consideration, thus can not capture context-aware biases in complex search scenarios. To alleviate such problems, we propose a context-aware position debiasing method, which is a query-position-dependent bias estimation model that could capture the patterns of context(e.g. display templates, target search domains, users' preference). Furthermore, we verified the effectiveness of pairwise debiasing method proposed in Unbiased LambdaMart [9] and fix the negative bias by training a negative bias estimation model on a randomized data set. Finally, experimental results demonstrate that evaluation on randomized data is more consistent with the online performance than using biased validation data. Both offline and online experiments indicate that the CAPD method out-performs existing debiasing methods under a complex search setting. An absolute improvement of 2.3‰on online click-through rate is achieved in Meituan search scenario.

REFERENCES

[1] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.
[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*. 89–96.
[3] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.
[4] Christopher J Burges, Robert Ragno, and Quoc V Le. 2007. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*.

193–200.

[5] Olivier Chapelle and Ya Zhang. 2009. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*. 1–10.

[6] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An experimental comparison of click position-bias models. In *Proceedings of the 2008 international conference on web search and data mining*. 87–94.

[7] Lesley H Curtis, Bradley G Hammill, Eric L Eisenstein, Judith M Kramer, and Kevin J Anstrom. 2007. Using inverse probability-weighted estimators in comparative effectiveness analyses with observational databases. *Medical care* (2007), S103–S107.

[8] Georges E Dupret and Benjamin Piwowarski. 2008. A user browsing model to predict search engine click data from past observations.. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. 331–338.

[9] Ziniu Hu, Yang Wang, Qu Peng, and Hang Li. 2019. Unbiased LambdaMART: An unbiased pairwise learning-to-rank algorithm. In *The World Wide Web Conference*. 2830–2836.

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

[11] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. 2017. Accurately interpreting clickthrough data as implicit feedback. In *ACM SIGIR Forum*, Vol. 51. Acm New York, NY, USA, 4–11.

[12] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. 781–789.

[13] Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and trends in information retrieval* 3, 3 (2009), 225–331.

[14] Ramesh Nallapati. 2004. Discriminative models for information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. 64–71.

[15] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*. 521–530.

[16] Ellen M Voorhees et al. 1999. The TREC-8 question answering track report. In *Trec*, Vol. 99. 77–82.

[17] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 115–124.

[18] Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. The lambdaloss framework for ranking metric optimization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1313–1322.

[19] Yisong Yue, Rajan Patel, and Hein Roehrig. 2010. Beyond position bias: Examining result attractiveness as a source of presentation bias in clickthrough data. In *Proceedings of the 19th international conference on World wide web*. 1011–1018.