Click-based Hot Fixes for Underperforming Torso Queries

Masrour Zoghi*
University of Amsterdam
Amsterdam, Netherlands
m.zoghi@uva.nl

Tomáš Tunys
Czech Technical University
Prague, Czech Republic
tunystom@fel.cvut.cz

Lihong Li
Microsoft Research
Redmond, WA
lihongli@microsoft.com

Damien Jose, Junyan Chen, Chun Ming Chin
Microsoft Bing
Bellevue, WA
{dajose, junyanch, chuchin}@microsoft.com

Maarten de Rijke
University of Amsterdam
Amsterdam, Netherlands
derijke@uva.nl

ABSTRACT

Ranking documents using their historical click-through rate (CTR) can improve relevance for frequently occurring queries, i.e., so-called head queries. It is difficult to use such click signals on non-head queries as they receive fewer clicks. In this paper, we address the challenge of dealing with *torso* queries on which the production ranker is performing poorly. Torso queries are queries that occur frequently enough so that they are not considered as tail queries and yet not frequently enough to be head queries either. They comprise a large portion of most commercial search engines' traffic, so the presence of a large number of underperforming torso queries can harm the overall performance significantly. We propose a practical method for dealing with such cases, drawing inspiration from the literature on learning to rank (LTR). Our method requires relatively few clicks from users to derive a strong re-ranking signal by comparing document relevance between pairs of documents instead of using absolute numbers of clicks per document. By infusing a modest amount of exploration into the ranked lists produced by a production ranker and extracting preferences between documents, we obtain substantial improvements over the production ranker in terms of page-level online metrics. We use an exploration dataset consisting of real user clicks from a large-scale commercial search engine to demonstrate the effectiveness of the method. We conduct further experimentation on public benchmark data using simulated clicks to gain insight into the inner workings of the proposed method. Our results indicate a need for LTR methods that make more explicit use of the query and other contextual information.

Keywords

Learning to rank; Underperforming queries

1. INTRODUCTION

An important problem in Information Retrieval (IR) is to rank the documents retrieved in response to a query such that the information need of the user is satisfied as quickly as possible. Originally, this problem was addressed by ranking documents based on their textual similarity to the query, using a feature ranker such as BM25. With the proliferation of such features, arose the problem of combining the recommendations put forth by the multitude of features. Machine learning algorithms rose to prominence as the preferred solution, and this gave rise to the field of learning to rank [32, LTR].

Despite the success of LTR, invariably the learned model performs suboptimally on certain queries, either due to the inadequacy of the features or because the query does not fit the profile of the "generic" query and since LTR methods are statistical by their very nature, they gladly sacrifice a minority of "misbehaving" queries to boost the overall performance across the full population. This problem might disappear with the introduction of new features or LTR algorithms. However, given the delay involved in the development and deployment process, it is desirable to devise a more automated and agile method for remedying underperformance. We propose a method for achieving this aim using implicit feedback from users.

We are particularly interested in improving the performance of LTR algorithms on a specific class of queries: *underperforming torso queries*. Torso queries are queries that occur less frequently than head queries, but are not so rare to be considered tail queries [33]. When sorted by frequency and split into three quantiles, queries in the second quantile are called torso queries [23]. We use a pragmatic definition that is given in absolute terms: queries that receive between 10 and 200 impressions during our two-week experimental period. Of course, the frequency of a torso query is highly dependent on the amount of traffic a search engine encounters; often, it is high enough to render our ideas practical.

A torso query is *underperforming* if the document ranking produced in response to it by the production ranker is of lower quality than the best ranking according to a page-level metric of interest [16, 27]. How can we use click feedback to improve the performance of an offline trained ranker on torso queries?

The idea of using click feedback to improve search results predates many LTR algorithms in use today. For instance, Joachims [24] proposed extracting pairwise preferences between documents for a given query, by searching through click logs produced by an already deployed ranker for skipped-clicked pairs: pairs of documents that appeared in one impression and the document that was placed in the lower position was clicked, but not the one in the higher position. Because the lower document was clicked and so inspected, assuming that the user scanned the list in descending order, we can deduce that the higher document must have also been inspected, but not clicked, so it is likely to be less relevant.

Despite the astuteness of the observation in the last paragraph,

*Part of this work was carried out when the first author was visiting Microsoft Research Redmond.

SIGIR '16, July 17 - 21, 2016, Pisa, Italy

ISBN 978-1-4503-4069-4/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2911451.2911500

without some level of exploration in the ranker that was used to collect the click logs, the estimated preferences are biased and so in practice, in the absence of intervention or prior assumptions [2], this method may not be as effective as one would hope. We test the hypothesis that, with adequate exploration, the preferences expressed by users, through their clicks, can lead to significant gains in performance. We adapt an idea that has proven effective in supervised LTR to our setting: in [5], *lambda-gradients* were proposed as a method for reordering the documents used in training according to their relative relevance, where the relevances were obtained from manually-labeled assessments. Applying this idea to the click-based preferences obtained through the procedure described in the last paragraph, we arrive at our method, which we call *click-based lambdas* and describe more explicitly in §2.2.

We test the effectiveness of click-based lambdas using real users interacting with a commercial search engine (cf. §3). A key finding in §3.5 is that, given proper exploration, click-based lambdas can lead to significant improvements over existing LTR methods. The most immediate application of the findings in this paper is to remedy queries on which a production ranker is performing sub-optimally. Given a set of queries that have been identified to be underperforming using one of several methods developed in recent years [16, 27, 29], they could be singled out for exploration and re-ranking without affecting the remainder of the traffic.

In addition to experiments with real users, we conduct more controlled experiments using simulated user clicks obtained from a click model in order to help us deepen our understanding of the proposed method, particularly in relation to LTR methods; see §4.1. In particular, our results show that click-based lambdas are most effective in the case of underperforming queries; see §4.4.

Importantly, our click-based lambdas method does not generalize across query-document pairs. The intended practical use case are queries on which the production ranker performs sub-optimally and for which this performance gap fails to be remedied by making modifications to the model. In such a situation, click-based lambdas can be used temporarily, as an *unsupervised*[1] *hot fix*.

We address the following research questions: (**RQ1**) Can click signals together with exploration be used to improve online performance of a commercial search engine on torso queries that receive much fewer click signals than head queries? (**RQ2**) To what extent does LambdaMART (the state-of-the-art LTR method) obtain the best ranking for each query? (**RQ3**) Can a combination of click feedback obtained from exploration and scores produced by LambdaMART lead to improvements over both methods? (**RQ4**) Is there a relationship between the effect of click-based lambdas on a query and the performance of the original LambdaMART model on it?

Our results are useful for IR researchers because they demonstrate a gap that exists between what the state-of-the-art LTR algorithms achieve and what could be achieved if one were to utilize user feedback; this presents a challenge to the researcher to invent novel methods for combining offline LTR algorithms with interaction data. Another challenge is to devise improved methods for detecting underperforming queries. Our results are useful for IR practitioners because they provide a simple yet effective solution for obtaining meaningful gains on torso queries, which comprise a large portion of the traffic, particularly in the case of underperforming queries. Given the simplicity of the proposed method, we believe it will be broadly useful in practice.

In summary, we provide an unbiased comparison of a click-based ranker against both a highly optimized, commercial search engine and the state-of-the-art LTR algorithm, LambdaMART, and demonstrate that using a small dose of exploration we can obtain improvements over both. The required exploration is small enough that the proposed method could be useful for underperforming torso queries, affecting a large segment of the traffic.

[1]By "unsupervised" we mean without the use of manually curated labels.

2. METHOD

In this section, we describe our method, click-based-lambdas, as well as the necessary background.

2.1 Background

We begin by introducing some notions and notations used in the rest of the paper. We are concerned with improving the performance of a production ranker on a fixed segment of the queries. For the sake of simplicity of the exposition, we will consider a single query in this segment, denoted q, with documents $d_1, \ldots, d_L$. The goal, then, is to order these documents so as to optimize some metric and in this paper we will assume that the metric of interest is a *page-level online metric*, which is evaluated based on the way the users react when presented with the given ranking of $d_1, \ldots, d_L$. An instance of such a metric is *page click-through rate* above a certain threshold K, i.e. the probability of receiving a click on any of the top K ranked results: we denote this by PCTR@K.

Let us point out that when considering PCTR, we may choose to limit ourselves to *satisfied* clicks if need be, where a click is deemed satisfied if the user spent 30 seconds or more reading the clicked document or if it was the last click in the session [2]. Indeed, inspired by previous work [11], this is a restriction that we impose. Other popular online metrics include time to success and session success rate. However, we use PCTR to evaluate the different methods under consideration here, since it is an online click-based metric that we could safely use in the case of both the real user data and the click model experiments.

Note that if the goal was to optimize an offline metric, such as Discounted Cumulative Gain (DCG), which is calculated based on annotated relevance judgments, then we could directly apply the many LTR algorithms in the literature. However, without assuming a particular user click model that could be used to estimate online metrics, LTR methods cannot be used to directly address the problem of optimizing an online metric; the only option is to optimize for an offline metric, such as DCG, which correlates well with the online metric of interest.

Next, we briefly review the main ideas behind the state-of-the-art LTR algorithm [7], called LambdaMART [6], which is the main source of inspiration for our proposed method. The idea is to use a succession of regression trees, each estimating the so-called *lambdas*, which are numbers that are assigned to each query-document pair (q, d_i) and indicate how much the score assigned to d_i by the previous trees needs to be modified to improve the offline metric of interest. More precisely, consider a query q and two documents d_i and d_j such that d_i has a higher relevance score for q than d_j, and suppose that the sum of the scores assigned by the regression trees grown so far to each document d_k is s_k; moreover, assume that we are interested in an offline metric M (e.g., DCG), and let ΔM_{ij} denote the difference in terms of M between the ranked list of documents sorted according to the scores s_k and the same list with the documents d_i and d_j swapped. Note that we are assuming that this quantity can be calculated, which is the case when dealing with offline metrics that can be calculated based on relevance judgments, but not so for online metrics, which is what we are concerned with in this paper. Then we define

$$\lambda_{ij} = \frac{\Delta M_{ij}}{1 + e^{\sigma(s_i - s_j)}}, \tag{1}$$

Algorithm 1 Click-based lambdas

Input: Query q, documents $d_1, \ldots, d_L$ and T impressions where q was issued to the search engine, a ranking of the documents d_i was displayed to the user and the resulting clicks were recorded

1: $\lambda_i \leftarrow 0$ for each $i = 1, \ldots, L$
2: **for** $t \in \{1, \ldots, T\}$ **do**
3: **for** (i, j) such that d_i was clicked in impression t and d_j was not clicked, but was ranked higher than the last clicked document **do**
4: $\lambda_i \leftarrow \lambda_i + 1$
5: $\lambda_j \leftarrow \lambda_j - 1$
6: **end for**
7: **end for**

Return: The documents d_i sorted in decreasing order of λ_i

where σ is a positive number that is tuned for performance in practice. Then, for each document d_i, λ_i is obtained by summing λ_{ij} over j such that document d_j is less relevant than d_i and subtracting λ_{ji} for every document d_j that is more relevant than d_i.

The general idea is that for every document that is ranked above d_i and should be ranked below it, d_i gets an "up-vote" and for every document that is ranked below d_i but should be above it, d_i gets a "down-vote" and the sum of these votes tells us whether d_i should be moved up or down in the list and how far. Then, LambdaMART fits a new regression tree to the λ_i and adds the tree to the ensemble. Although LambdaMART is known to be locally optimal for optimizing offline metrics [10], the same does not necessarily hold for online metrics that we focus on in this work.

2.2 Click-based lambdas

We propose using an online, click-based analogue of the lambdas used in LambdaMART to decide which documents should be pushed to the top. Unlike LambdaMART, our method, *click-based lambdas*, is unsupervised and only uses interaction data.

Algorithm 1 provides the pseudo-code for the method: given a query q with documents $d_1, \ldots, d_L$, a number of impressions recording the order in which these documents were presented to various users and the resulting clicks, we start out by setting $\lambda_i = 0$ for each document d_i (Line 1), and updating them as described in the following. Given an impression, suppose that the documents were presented in the order

$$\begin{array}{ll} \mathbf{d_{i_1}} & \\ \mathbf{d_{i_2}} & \checkmark \\ \mathbf{d_{i_3}} & \\ \mathbf{d_{i_4}} & \checkmark \\ \vdots & \end{array}$$

with the check mark ✓ denoting the positions where clicks were observed. Then, given a document d_i that was clicked and a document d_j that lies above the last clicked document and was not clicked (e.g., $(i, j) = (i_2, i_3)$ or (i_4, i_1)), we set $\lambda_{ij} = 1$ and $\lambda_{ji} = -1$, which is equivalent to setting σ to 0 and ΔM_{ij} to 1 in (1). Given this, we update the λ_i as follows (Lines 4 and 5):

$$\lambda_i = \lambda_i + \lambda_{ij}$$
$$\lambda_j = \lambda_j + \lambda_{ji}.$$

As the discussion of LambdaMART in §2.1 indicates, it is possible to modify the above procedure by adopting a more complex definition for λ_{ij}, but in this paper we are more interested in the question of whether or not click signals can be used to improve online performance, rather than the most optimal method for doing so.

A few comments are in order: first of all, note that there are no requirements on the ranking $d_{i_1}, d_{i_2}, \ldots$, so it could very well be provided by a deterministic ranker. However, the method benefits strongly from exploration; this is due to the following observation: consider a pair of documents d_i and d_j that are retrieved in response to a query q and consider the sum of the λ_{ij} across all impressions involving q; if these impressions presented random rankings, then this sum would be an estimate of the preference that the users have for d_i over d_j. Now, suppose that the ranker is deterministic and d_i always appears above d_j; then, the sum of the λ_{ij} could never be positive, even if d_i is strongly preferred to d_j: it can at best be equal to 0. Our experience with click-based lambdas confirms the intuition that without exploration they can lead to degradation; we do not present these negative results due to space constraints.

Given the above observations, our experiments employ two types of exploration. In the experiments with real user clicks discussed in §3, we use uniformly random shuffling of the top 5 documents ranked by the production system, while for the more flexible click model-based experiments in §4, we use a less disruptive form of exploration obtained through interleaving the existing ranker with a random one, which results in roughly half of the documents being chosen randomly. The former exploration scheme was imposed upon us due to the evaluation methodology, as discussed in §3.3.

A final comment is that our click-based lambdas method does not generalize across query-document pairs and so it is not to be thought of as a replacement for LTR methods, but as a remedy for their shortcomings, and hence should be run alongside LTR methods or as a post-processing step to such methods. See §5.

3. THE PRACTICAL IMPACT OF CLICK-BASED LAMBDAS

In this section, we address our first research question and present our experiments on real users to test the extent to which the idea of click-based lambdas can lead to improvements in practice over a production ranker in a large-scale commercial search engine. We describe our experimental setup, the data, the evaluation method and the results.

3.1 Experimental design

In order to answer **RQ1**, we use click logs collected by a popular search engine, where the results produced by the production ranker were randomized. The purpose of the randomization is twofold. First, we would like to test the hypothesis that infusing exploration into the ranked lists and using the resulting feedback can lead to improved rerankings. Secondly, and more importantly, such randomized click logs allow for a counterfactual comparison of the production ranker against the outcome of our click-based lambdas as well as a number of other baselines discussed below without the need to run numerous A/B tests on real users. In particular, we compare these methods based on their performance according to PCTR@3, as described in §2.1. Below, we detail the specifics of our experiments and evaluation methodology.

3.2 Exploration click logs

In this section, we describe the click logs that were used in the experiments aimed at answering **RQ1**. The top 5 results produced by a commercial search engine were shuffled uniformly at random before being presented to the users, and the clicks on the reshuffled ranked list were recorded. This randomization was applied to a fraction of the non-head queries that were issued to the search engine over a period of two weeks. The data was split into two parts, one for training the rankers being compared and the other for evaluation. As pointed out above, the shuffling is necessary for us to be able to conduct unbiased evaluation of our various modifications to the production ranker, without having to run expensive A/B tests.

Algorithm 2 Get Importance Probabilities

Input: Exploration log and evaluation cutoff K
1: **for** each ranking $P \in \mathcal{C}_K^L$ **do**
2: $C(P) = \#\{I_j \mid I_j^{1,\ldots,K} = P\}$
3: **end for**
4: $IP(P) = \frac{C(P)}{\sum_Q C(Q)}$ for each $P \in \mathcal{C}_K^L$
Return: IP

The choice of 5 as the number of documents to be shuffled was motivated by the practical issue of the amount of exploration data that would be needed for us to be able to carry out our experiments. The difficulty with importance sampling, the technical core of the unbiased evaluation approach as described below, in the context of ranking is that the number of possible rankings (a.k.a. actions) grows rapidly with the length of the list. More specifically, given ranked lists of length L, in order to be able to evaluate the top L results of any ranker, we would like to have all $L!$ different impressions in our randomized logs. Since 5! is equal to 120, this means that less than 1% of the impressions in the exploration log can be used for evaluation. Now, given that the randomization of the results was only applied to a small fraction of the non-head traffic, shuffling even the top 6 documents would have required the randomization of a far greater portion of the traffic than was allowed by our exploration budget. It should be noted that such considerations apply to the empirical evaluation protocol we adopt here, not to the click-based lambdas approach proposed in this work.

To carry out the needed comparisons, the dataset is randomly split into two folds of equal size: one fold is reserved for the unbiased IPS evaluation (see below) and *part of* the other fold is used for training. More specifically, for a given training size, n_{imps}, we randomly select n_{imps} impressions per query in the training fold and train click-based lambdas and each of the baseline algorithms discussed above on these n_{imps} impressions and evaluate the resulting rankers using the test fold. Naturally, in the case of the production and random rankers, we skip the training part of the above procedure and only evaluate it on the test fold, since the production ranker is pre-determined by the commercial search engine and there is nothing to learn for the random ranker.

In order to avoid issues with bias, we restrict ourselves to queries such that the top 3 documents in their impressions in the test fold contain all 60 ($= 5 \times 4 \times 3$) possible ordered lists of 3 documents chosen out of the top 5 documents, which were shuffled in our dataset. Imposing this restriction leaves us with 57 queries and roughly 58,000 impressions. This restriction is put in place for the sake of our evaluation methodology: our method click-based lambdas imposes no such requirement.

The above process is repeated 1000 times for training sizes n_{imps} going between 2 and 20 impressions per query in increments of 2 and between 20 and 200 impressions in increments of 20.

3.3 Evaluation method

We set as our main metric of interest *satisfied page click-through rate on the top* 3 *positions*, which we denote by PCTR@3. This is the probability that we observe a satisfied click on one of the top 3 documents presented by the ranker, where a click is deemed satisfied if the user spent 30 seconds or more reading the clicked document or if it was the last click in the session. One important reason for this choice is that at least on some (mobile) devices the top 3 documents are visible to the users without any extra effort on their part (i.e., without having to scroll down), and so it is desirable to improve the quality of the top 3 results as much as possible. Of course, for any K, given the appropriate exploration dataset, one can replicate our experiments to evaluate for PCTR@K.

Algorithm 3 Get Importance Numerators

Input: Exploration log of length N, evaluation cutoff K and ranker R with rankings $R_1, \ldots, R_r$
1: $IN(P) = 0$ for all $P \in \mathcal{C}_K^L$
2: **for** $i \in \{1, \ldots, r\}$ **do**
3: $IN(R_i) = \#\{I_j \mid R(I_j) = R_i\}/N$
4: **end for**
Return: IN

Algorithm 4 Evaluate $CTR@K$ of ranker

Input: Exploration log of length N, number of shuffled documents L, evaluation cutoff K and ranker R with rankings $R_1, \ldots, R_r$
1: Let $(d_1, \ldots, d_K) = I_1$
2: Let $\sigma_1, \ldots, \sigma_n \in \mathcal{C}_K^L$ be permutations on K elements such that ranking R_i orders the documents as $d_{\sigma_i(1)}, \ldots, d_{\sigma_i(K)}$
3: $IN =$ Get Importance Numerators$(\{I_j\}, K, R)$
4: $IP =$ Get Importance Probabilities$(\{I_j\}, K)$
5: $nM = 0$
6: $nCM = 0$
7: **for** $i \in \{1, \ldots, N\}$ **do**
8: $P = I_i^{1,\ldots,K}$
9: $nM = nM + IN(P)/IP(P)$
10: **if** I_i received a click on the top K documents **then**
11: $nCM = nCM + IN(P)/IP(P)$
12: **end if**
13: **end for**
14: $CTR@K = nCM/nM$
Return: $CTR@K$

Next, we explain the Inverse Propensity Scoring (IPS) evaluator [4, 30, 31, 41] used to evaluate rankers on the exploration click logs. To understand the basic idea, let us first consider the situation with a single query, i.e., we will fix a query q and L documents $d_1, \ldots, d_L$ that need to be ranked in response to the issued query q. Let us also assume that all permutations of the L documents have been presented to users and the clicks have been logged. Then, given any ranker and any page-level metric, we can extract all impressions in our exploration log that order the L documents in the same way as our ranker and evaluate the metric on those impressions only. Such a procedure is statistically equivalent to an actual A/B test, so will give us an unbiased estimate of the metric on the given ranker; consequently, this estimate converges to the true value of the metric given enough exploration data.

This basic idea needs to be modified to address both the existence of multiple queries and the case that the ranker under consideration does not produce the same ranking of the documents $d_1, \ldots, d_L$ in every impression (e.g., due to personalization). To deal with this, we make use of importance sampling to evaluate the metric by treating the impressions in the randomized log as samples from a "source" distribution and the rankings produced by the ranker we would like to evaluate as samples from a "target" distribution. Then, we weight the contribution of each matching impression by the ratio of the probability of observing the given impression under the target and the source distributions: these probabilities are calculated from the data using Algorithms 3 and 2, respectively. These ratios are then combined as in Algorithm 4 to get the desired estimate. Since we are interested in PCTR@K, we can use impressions that match only the top K results produced by our ranker. Below, we provide the details of the IPS evaluator.

DEFINITION 1. *We denote by $\mathcal{C}_K^L$ the set of K-element ordered subsets (called "rankings") of the documents $d_1, \ldots, d_L$. Note that the size of $\mathcal{C}_K^L$ is equal to $L!/(L-K)!$.*

Suppose we are given an *exploration log*, by which we mean impressions $I_1, \ldots, I_N$, each consisting of some permutation of the documents $d_1, \ldots, d_L$, such that for each ranking $P \in \mathcal{C}_K^L$, there is one impression, I_j, in the log such that the ordered subset of top K documents (denoted by $I_j^{1,\ldots,K}$) matches P. Algorithm 2 calculates the probability of seeing each ranking P in the log.

Moreover, let R be a ranker that assigns one of the rankings $R_1, \ldots, R_r \in \mathcal{C}_K^L$ to each impression. We will denote by $R(I_j)$ the ranking that R assigns to the j-th impression. Algorithm 3 calculates the probability of encountering each R_j in the log.

Algorithm 4 loops through the N impressions and whenever there is a match between $R(I_j)$ and $I_j^{1,\ldots,K}$, it increments the number of matches (i.e. nM) by the ratio between $IN(I_j^{1,\ldots,K})$ and $IP(R(I_j))$, and if additionally there was a click on the top K documents, we also increment the number of clicked matches (i.e. nCM). The algorithm returns the ratio of the final nM and nCM.

3.4 Baselines

We compare click-based lambdas to the following baselines, which are applied to the same n_{imps} impressions used to calculate our click-based lambdas:

CTR ranker: Given a query, for each of the top 5 ranked documents, we record its click-through rate (CTR), which is equal to the number of times the document received a click divided by the number of times it was displayed (i.e., the number of times the query was issued). The CTR ranker then sorts the documents in decreasing order of their CTRs.

CTR1 ranker: Same as above with CTR replaced by CTR1, i.e., the number of times the document was clicked while at the top of the list divided by the number of times it appeared there.

Position bias corrected CTR ranker: This is a modification of the CTR ranker, where the clicks are counted as follows: given a query q appearing in impressions $I_1, \ldots, I_N$, we first count for each j the number, n_j, of impressions with clicks at the j^{th} position; now, given an impression with a click on document d that appears at the j^{th} position, instead of incrementing the numerator of the CTR for document d by 1 as in the CTR ranker, we add $1/n_j$. The idea is that if you are half as likely to get a click on the second position than on the first position in a collection of randomly shuffled result pages, then receiving a click on the second position is worth two clicks on the first position.

In addition to the above, we also compare against the **production** and **random** rankers, where the random ranker shuffles the top 5 results produced by the production ranker.

3.5 Results

In this section, we provide results to answer **RQ1**. We make use of the experimental setup outlined in §3.1. The average performances (determined over the 1,000 repetitions described at the end of §3.2) are depicted in Fig. 1, with 95% confidence bands drawn around each mean curve. Moreover, the results are normalized by the performance of the production ranker.

As these results illustrate, even with as few as 20 shuffled impressions per query, all four click-based methods improve significantly upon the production ranker, with three of them giving improvements even with as few as 10 impressions (click-based lambdas, CTR ranker, Bias corrected CTR ranker). Moreover, click-based lambdas give us the largest boost as the number of impressions grows.

Furthermore, inspecting the plot more closely, one can make the following observations:

- The performances of both the click-based lambdas method and the CTR1 ranker improve as the training size increases, whereas that does not seem to be the case with the CTR ranker and its position bias corrected variant.
- For training sizes that are equal to 80 impressions per query or larger, click-based lambdas significantly outperform all other methods.
- Despite the asymptotic flaw of the two CTR rankers, they appear to be very sample efficient, as even 6 randomized impressions per query suffice for them to improve upon the production ranker.
- The position bias correction employed here does not remedy the shortcomings of the CTR ranker, since the results of the two rankers (the CTR ranker and its position bias corrected version) are intertwined with each other.

The most perplexing of these observations are those involving the CTR ranker and its bias corrected variant, which seem to suggest that CTR can result in non-trivial improvements quickly, but then it hits a ceiling. We are not aware of any user models that can explain this, so we leave it as future work to come up with an explanation.

Given these results, we can answer **RQ1** in the affirmative, with the addition that relative comparisons between documents can lead to the largest improvements, as demonstrated by the click-based lambdas method. Importantly, this can be done efficiently enough to be useful for torso queries (with between 10 and 200 impressions in our exploration log), hence our click-based lambdas method has the potential to positively affect a large portion of the traffic.

4. ANALYZING THE IMPACT OF CLICK-BASED LAMBDAS

In this section, we take a closer look at the click-based lambdas method in an attempt to gain a deeper understanding of its inner workings. To this end, we conduct experiments using simulated clicks generated by a click model. This allows us to acquire a more detailed understanding of the effect of click-based lambdas and to investigate less severe forms of exploration. We opted for a more artificial experimental setup because of both the extra flexibility and the added control over the production ranker that it affords us. As we will see, in this setup we can dissect the results using quantities that cannot even be calculated in the case of real click logs.

4.1 Experimental design

The click model [8] we use for this set of experiments is a probabilistic version of the cascade click model [9, 15, 18], which inspects the documents in the result list from top to bottom and clicks on each document with a probability that depends on its relevance and abandons the search with another probability, which also depends on the current document under examination: note that we allow for more than one click on a result page. This click model is applied to the MSLR Web-30K [34] and Yahoo! Learning to Rank Challenge [7] datasets; see Table 1 for the specifics of these datasets. In particular, the documents in these datasets are assigned one of five relevance labels and we use the parameters specified by the navigational click model described in previous work [20] to set the click and abandonment probabilities, which depend on the relevance of the document being examined by the simulated user.

One important benefit of this setup is that it allows us to compute the *normalized* page click-through rate, denoted nPCTR, of a page,

Table 1: The specifics of the datasets used.

Datasets	Queries	URLs	Features
MSLR-WEB30K [34]	31,531	3,771,125	136
YLR Set 1 [7]	29,921	709,877	519
YLR Set 2 [7]	6,330	172,870	596

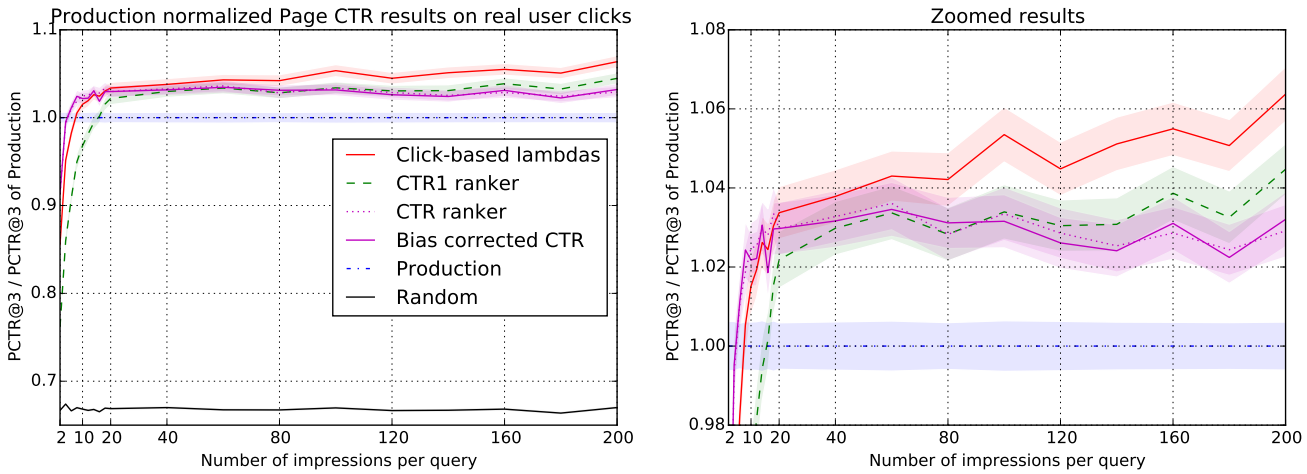

Figure 1: Page CTR@3 results for increasing numbers of impressions per query. The right plot is a zoomed version of the left one.

which we define as follows:

DEFINITION 2. *Given a click model, a query q and ranked documents $d_1, \ldots, d_L$, we define* nPCTR@K *of the ranked list to be equal to PCTR@K of the ranked list divided by the PCTR@K of the perfect ranking, i.e., the one obtained by sorting the documents in descending order of relevance.*

Note that PCTR is related to nPCTR in the same way that Discounted Cumulative Gain (DCG) is related to Normalized DCG (NDCG). Even though our ultimate objective is to optimize for PCTR, the utility of nPCTR is in that it gives us an indication of how suboptimal our ranked list is. Note that PCTR might be small because either there are no relevant documents to present or the ranked list is of low quality: nPCTR resolves this uncertainty by only measuring the quality of the ranker, rather than that of the documents being ranked. We will make extensive use of this metric in our click model based experiments in order to get a better understanding of the aggregate results, as explained below.

Using this setup, we carry out three sets of experiments:

1. To address **RQ2**, we consider a per query decomposition of the performance of an *offline-trained* LambdaMART model according to the nPCTR performance of each query: by an offline-trained model, we mean one that has been trained on the manually produced relevance judgments provided with our LTR datasets. This gives us a more detailed understanding of how suboptimal the performance of LambdaMART is across queries. The results are presented in §4.2.
2. To address **RQ3**, we interleave the above LambdaMART model with a random ranker for a fixed number of impressions for each query and combine the resulting click-based lambdas with the scores produced by LambdaMART in various ways and compare the performance of the resulting ranker against that of the off-line model. The results are presented in §4.3.
3. To address **RQ4** and better understand the performance changes observed in the last set of experiments, we consider a decomposition of the results into query segments according to the performance of the LambdaMART ranker on the query, where performance is measured according to nPCTR. The results are provided in §4.4.

The rationale behind the type of exploration experiment carried out in the two sets of experiments outlined above is the following: given a set of queries on which our production ranker is performing substantially worse than the ideal ranker, it might be easier to motivate an exploration scheme where a certain fraction of the top results are chosen at random, rather than a complete shuffling of the results as in the case of the exploration click log used in §3. Note that in the case of the latter, complete shuffling was necessary to facilitate offline evaluation of the re-ranking schemes investigated in this paper, but it is not necessary for the methods themselves.

4.2 Detailed analysis of the performance of LambdaMART

Given the results in §3, a natural question is how close the results produced by LTR models are to being optimal across queries. This question is interesting because it offers us a deeper understanding of whether or not there is a need for paying closer attention to individual queries. This is the purpose of **RQ2**. In order to address it, we take a closer look at the quality of the rankings produced by a parameter-tuned[2] LambdaMART model trained on two large datasets and inspect the nPCTR of the queries in the dataset, as measured using the click-model experimental setup described in §4.1. The motivation for using nPCTR, as opposed to PCTR, is that the former is a more direct indicator of the quality of the ranker, regardless of the relevance of the documents in the dataset.

More precisely, using a large parameter sweep, we train LambdaMART models using the training folds of the MSLR and Yahoo! Set 1[3] datasets; we also use the validation fold to regularize the number of trees grown by the model through early stopping of the tree growing process if no improvement on the validation set was observed after the addition of 50 consecutive trees. Given all models trained in this manner, the model with the highest nPCTR@3 performance on the validation fold is evaluated on the test fold, keeping track of the performance on individual queries.

Before proceeding to the results, let us address the issue of the target metric used by LambdaMART: we use PCTR, nPCTR and NDCG with cut-offs of 3, 5 and 10 in the training process. It turns out that the model with the highest nPCTR@3 performance uses NDCG as the training metric. This is probably because NDCG is an easier objective function to optimize than PCTR and nPCTR perhaps because of the discounting used in NDCG, which might make it a more slowly varying metric as a function of the ranking.

[2]Meaning that the parameters used by LambdaMART, e.g., max tree depth, have all been tuned. In machine learning this is sometimes called "hyperparameter-tuned."

[3]For the results in this section and the ones that follow, we leave out Yahoo! Set 2 because the results of the two subsets of the Yahoo! were qualitatively similar, so we chose to leave those plots out of this presentation, in order to leave room for a more detailed discussion of the results.

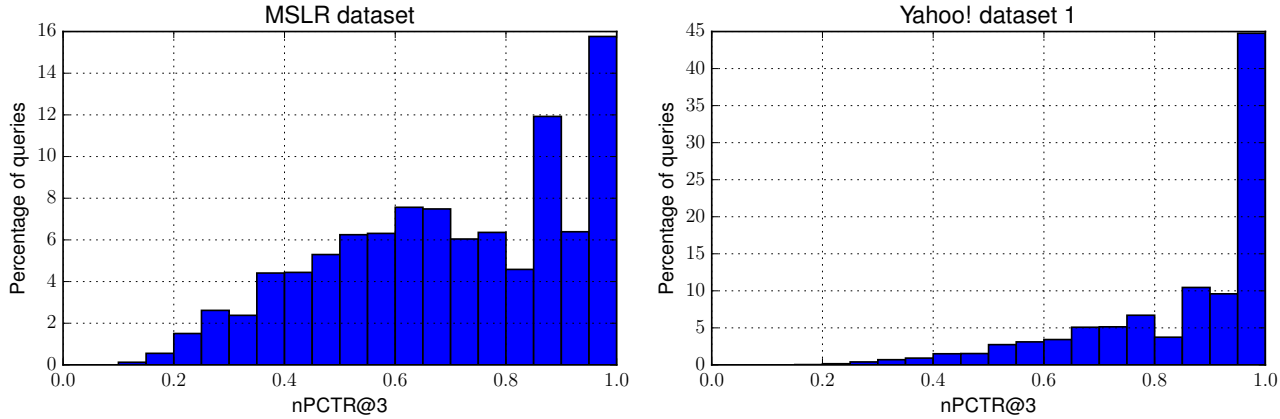

Figure 2: Histogram of the nPCTR@3 performance of LambdaMART on individual queries in the MSLR and Yahoo! datasets.

The plots in Fig. 2 show histograms of the nPCTR@3 performance of the parameter-tuned LambdaMART model selected as above on the individual queries in the test fold of the MSLR and Yahoo! Set 1 datasets. The most clear distinction between the two sets of results is that the ranking results produced by the model that was trained on the larger of the two datasets (i.e. MSLR) tend to be of much lower quality: in the case of the Yahoo! dataset, for roughly 55% of the queries, the PCTR@3 of our LambdaMART model is 90% of PCTR@3 of the ideal ranking (i.e. their nPCTR@3 is at least 0.9), whereas in the case of the MSLR dataset this is true of less than 23% of the queries.

We hypothesize that the above discrepancy is due to the following: even though the specific features that accompany the Yahoo! dataset are not specified, according to the overview provided by the challenge organizers [7], the feature set contains many click features, which could potentially aid in training a model that performs better on online metrics. In contrast, there are only three features among the 136 included with the MSLR dataset that pertain to clicks (cf. the feature list in [34]). Also, another potential explanation could be that the MSLR dataset has many more documents per query than the Yahoo! one, so this might hinder the ability of LambdaMART to learn the ideal ranking for each query.

Given the above observations, we see that the answer to **RQ2** depends to a large extent on the quality of the features used to train the model, but that a large fraction of the queries tend to underperform: for instance, our results for the Yahoo! dataset show that, in the case of more than 30% of the queries, the LambdaMART model attains nPCTR@3 of 0.8 or lower; in other words, there is room for a 20% improvement in the PCTR@3 of almost a third of the queries. In the case of the MSLR dataset, the same holds for more than 60% of the queries.

4.3 Click-based lambdas vs. LambdaMART

In this section, we continue using our click-model based setup to examine the effect that click-based lambdas have on page-level performance of our LambdaMART model. In particular, we interleave the offline-trained ranker against a random ranker, using the Team Draft interleaving procedure [39], and calculate click-based lambdas using the procedure described in §2.2. Please note that interleaving is used as a mechanism for introducing randomness into the ranked lists for the purpose of exploration, rather than the usual goal of comparing two rankers. This is performed for various numbers of interleaved impressions per query (denoted n_{imps}). We then consider convex combinations of the resulting click-based lambdas and the scores produced by our LambdaMART model and evaluate the resulting rankers. The reason for considering convex combinations of the LambdaMART scores and the click-based lambdas is to make use of the information provided by both methods.

The plots in Fig. 3 illustrate the results of these experiments on the MSLR and the Yahoo! datasets for 5 different values of n_{imps}. As with the results in the previous section, there are qualitative differences between the results for the two datasets:

- After 20 interleaved impressions, click-based lambdas already outperform LambdaMART in the case of the MSLR dataset: to see this, note that in the bottom curve in the left plot in Fig. 3, the right end (i.e., where $\alpha = 1$), which corresponds to click-based lambdas, is higher than the left end (i.e., $\alpha = 0$), which corresponds to LambdaMART. On the other hand, in the case of the Yahoo! dataset, we need more impressions for click-based lambdas to outperform LambdaMART on their own.
- In the case of the MSLR dataset, given 100 interleaved impressions, there is very little that is gained from combining the scores of LambdaMART with click-based lambdas over how the latter was performing, which is in contrast to the Yahoo! dataset, on which even 100 interleavings against the random ranker are not enough to render the offline model unnecessary.

We attribute these differences to the worse performance of the LambdaMART model trained on the MSLR dataset; see §4.2.

Irrespective of the difference between the results for the two datasets, what can be readily observed in both plots is that even after a small number of impressions for which the ranked results produced by LambdaMART are interleaved with a random ranker, the scores produced by LambdaMART can be modified using click-based lambdas to produce significantly improved ranked lists. Thus, the answer to **RQ3** is affirmative: using a modest amount of exploration, involving randomly selecting roughly half of the documents in 20 impressions, we can combine the resulting click-based lambdas with the LambdaMART scores to obtain substantial improvements over the LambdaMART model as well as click-based lambdas themselves.

4.4 Decomposition of the effect of click-based lambdas

Here, we inspect more closely the effect that combining click-based lambdas with LambdaMART scores has on individual queries. This is done by considering one particular value of α (as in §4.3) and investigating the change in performance, compared to the original LambdaMART model, on a per query basis. We then group the queries according to their performance under LambdaMART.

The two plots in Fig. 4 depict this decomposition for the MSLR and the Yahoo! datasets when α and n_{imps} are set to be 0.1 and 100, respectively: the value of α is set to 0.1 because in all of the

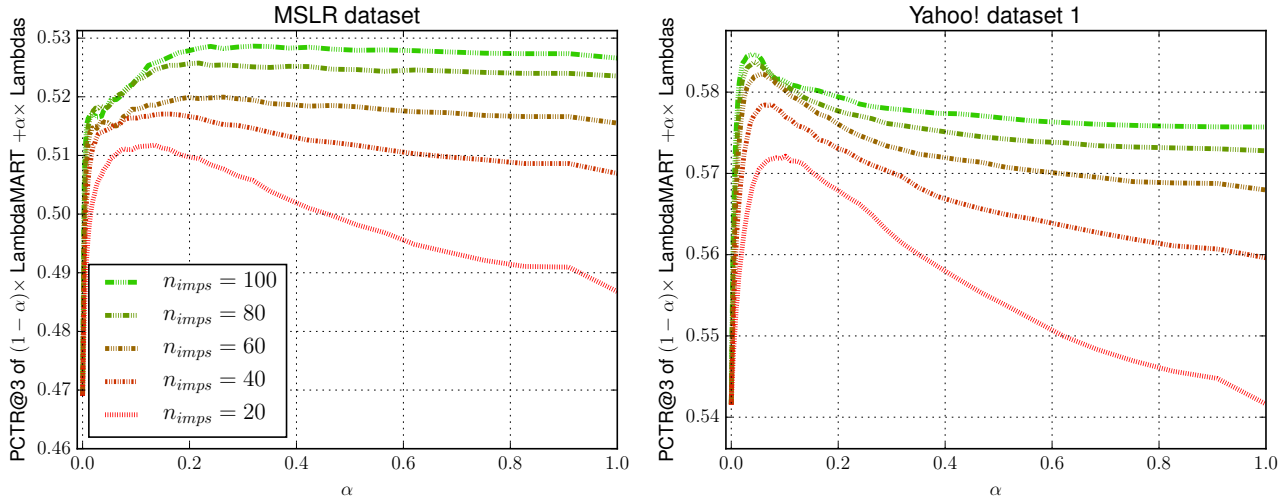

Figure 3: PCTR@3 of convex combinations of LambdaMART with click-based lambdas.

curves in the two plots in Fig. 3 the value at $\alpha = 0.1$ is roughly close to the maximum, so this seems like a good practical guess for α in the absence of more detailed knowledge of the performances; moreover, we chose the number of interleaved impressions to be large to get a better understanding of the changes in the "limit" as the amount of exploration becomes large.

In both plots in Fig. 4 the horizontal axis groups queries according to nPCTR@3 of the LambdaMART model, while the vertical axis measures the change in PCTR@3 as click-based lambdas are added to the scores produced by LambdaMART. Perhaps the most prominent phenomenon in these plots is that most of the degradation caused by click-based lambdas occurs in the case of queries on which the performance of the original LambdaMART model is rather close to being optimal. This is rather obvious in the case of the Yahoo! dataset, where the majority of the degradations are caused by queries on which the nPCTR@3 of the LambdaMART model is higher than 0.975, i.e., those represented by the right-most red bar in the bottom plot in Fig. 4. However, this is also the case in the case of the MSLR dataset, although perhaps in a slightly less extreme form. We consider this finding to be a positive one, since it implies that if one can determine queries on which the production ranker's performance is close to being optimal, then we can avoid applying the ideas presented in this paper to those queries and avoid the pitfall of ruining the ranked lists produced by LambdaMART where they are performing well, leading to more effective use of the signals harvested from the exploration data.

In other words, **RQ4** can be answered in the affirmative: click-based lambdas tend to help most in the case of queries that are underperforming under the offline-trained LambdaMART model, so given an identification of such queries, it is advisable to apply the method proposed in this paper to those queries only.

4.5 Summary

The issue of underperforming queries is one that LambdaMART grapples with in most cases. Click-based lambdas used in conjunction with a small amount of exploration can lead to significant improvements, in particular when combined with the scores produced by the original LambdaMART model. This suggests that a practical solution can be devised to remedy the underperforming queries affecting our ranker's performance, which we discuss next.

5. PRACTICAL CONSIDERATIONS

In this section, we say a few words about one possible use-case for the click-based lambdas method. This is in particular relevant

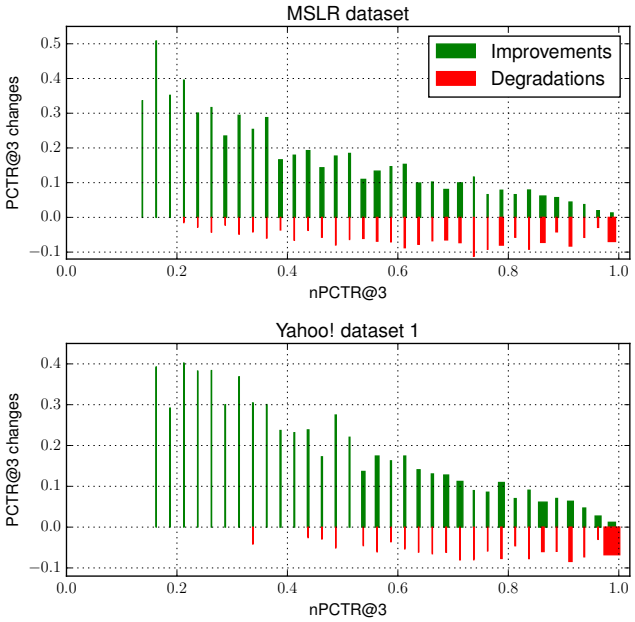

Figure 4: PCTR@3 improvements and degradations of queries if LambdaMART were to be replaced by LambdaMART + $0.1 \times$ Lambdas: the queries are grouped by their nPCTR@3 under LambdaMART. The green boxes take account of queries whose PCTR@3 improves by the transition from LambdaMART to LambdaMART + $0.1 \times$ Lambdas, while the red boxes correspond to queries that would suffer a loss. The height of each box shows the average change in the PCTR@3 of the queries in that bin and the width measures the number of queries in the bin: so, the area of the box is an indication of the effect that the changes in the performances of the queries in the given box have on the overall performance.

given the fact that all of the methods discussed in this paper are query-document specific, in the sense that the outcome is a re-ranking of a specific set of documents for each query, with no generalization across query-document pairs. This of course does not conform to the regular mold of LTR methods. However, as we describe below, our method can be used in conjunction with them. The idea is that even though eventually better features and LTR algorithms might be devised that could address the shortcom-

ings of the existing ranker, we would like to get better results right away, rather than waiting for the invention and deployment of an improved system. We see this as a more automated alternative to rule-based patches that try to deal with specific types of queries.

Let us now address the issue of implementation in practice: given a set of underperforming queries and their corresponding ranked lists of documents provided by click-based lambdas, we can simply use the existing features to fit a (potentially large) regression tree to the lambdas. Note that the goal here is not generalizability, so the regression tree is simply a compression mechanism. Moreover, given a binary feature that singles out our underperforming queries, we can add a node to the top of the above tree to make sure it is only applied to the relevant query-document pairs. This modified tree can then be added to our existing ranker, either as a post processing step or as an additional tree in our existing ensemble of trees: in the latter case, we can modify the existing trees by adding the same underperforming query identification to the root of each tree to make sure the existing ranker is not applied to the queries for which we have click-based lambdas. In this way, we incorporate the re-rankings learned by our method into the existing ranker without the need to alter the existing infrastructure dramatically.

A natural question at this point is whether or not it is possible to use click-based lambdas to modify the LTR model globally, rather than restricting the modification to the underperforming queries. Preliminary results involving numerous attempts at augmenting the LambdaMART model resulted in no improvements in the overall performance of the ranker beyond that of the original LambdaMART model that was trained on the relevance judgments.

As the experimental results in §3.5 demonstrate, a few dozen randomized impressions is all that is required to obtain non-trivial improvements. So, given, e.g., a query that is issued a few hundred times a week, using our click-based lambdas method, obtaining improvements over the course of a month is easily within reach, even assuming that no more than 20% of the impressions can be tampered with for the purpose of exploration. In the case of many popular retrieval systems, be it web search engines or recommender systems, the rough requirements outlined above apply to torso queries.[4]

6. RELATED WORK

In addition to the work on evaluation (based, in particular, on inverse propensity scoring and click models) discussed in §3.3 and §4.1, we discuss related work on implicit feedback, exploration and online learning to rank, and ranker specialization.

6.1 Implicit feedback

Learning from user feedback has a rather long history in IR [1, 11, 22, 24, 36], much of it predating many of the major breakthroughs in LTR. As a result of that, many of the experimental results in the literature on this topic need to be revisited, given that the baselines used for comparison are substantially weaker than the LTR models [7] and features [42] in use today. Extensive user studies indicate that user clicks can be of great use in inferring the relative relevance of documents to a given query [25]. Given this, a natural question is whether implicit feedback can still be useful despite a decade of advances in LTR. Our results show that the answer is *yes*.

An important exception to the above criticism regarding the weakness of the baselines is [35], which is the work most closely related to ours. Like us, the authors use a commercial search engine as their main baseline. Their method corresponds to our CTR1 baseline, which our results in §3.5 show to be less efficient than click-based lambdas in learning a good ranking. Also, CTR1 only takes into account documents that appear in the first position, so it is important that each document appears in the top position relatively often. Another important difference between this work and [35] is the latter's focus on *recency ranking*, which is solely concerned with queries with shifting intent; we make no such assumption.

[4]This argument also applies to head queries, but the sample efficiency demonstrated here is more likely to be of greater significance in the case of the less frequent queries.

6.2 Exploration and online LTR

Another body of work related to ours is that of online LTR, where the goal is to balance exploration and exploitation [17, 26] while improving the ranker incrementally [14, 19, 21, 28, 37, 38, 40, 43]. This is in contrast to the "explore first" strategy investigated in this paper, where the exploration is carried out entirely before we set out to extract a ranker from the data. The advantage of the adaptive exploration scheme is in its potential ability to avoid unnecessary exploration, however such online schemes are generally very difficult to utilize in industrial applications, both for technical and business reasons. On the other hand, the exploration schemes discussed here are not as sophisticated, but they tend to be much easier use in practice. In particular, in the case of underperforming queries, where there is compelling evidence that the existing ranker is performing poorly, it is much easier to make the case for exploration.

6.3 Underperforming queries

One group of queries that have generated special attention and inspired work on ranker specialization are so-called underperforming queries [16, 27]. As discussed in §4.4, click-based lambdas are most effective in the case of underperforming queries and most harmful in the case of queries on which the production ranker is performing well, so the method would benefit greatly from prior knowledge of the quality of the existing ranker on individual queries. We have not made use of such underperforming query detection techniques for the experiments in §3 and the results presented in §3.5 are not restricted to underperforming queries. Such a restriction would of course inflate the improvements in the performance of click-based lambdas compared to the production ranker.

One solution for dealing with underperforming queries is ranker specialization [3, 12, 13, 16], where one explicitly takes into account the fact that queries vary significantly in terms of ranking and may need different treatments. The general idea is to cluster "similar" queries into different groups, based on different notions of similarity, and train a *specialized* ranker for each cluster. The main issue with this body of work is the weakness of the baselines used for comparison because the models used for the specialized rankers are linear and so are those of the baselines. It remains an interesting open question to apply the same idea with more complex models such as ensembles of trees.

7. CONCLUSION

We have investigated the use of click feedback together with exploration to improve upon learning to rank (LTR) methods. We have carried out comparisons against both a commercial search engine, using real users, and the state of the art LTR algorithm, LambdaMART, using a well-established simulated user model. Furthermore, we experimented with two different exploration schemes: random shuffling of top L documents produced by the production ranker in use, for some fixed number L, and interleaving the results of the ranker against a random ranker.

The results of these experiments demonstrate that significant improvements can be obtained in terms of page-level online metrics,

in particular when employing relative comparisons between documents as with our click-based lambdas method. We provide a simple yet efficient remedy to close the gap between the online performance of LTR methods and that of click-based methods in the case of underperforming torso queries.

We have focused on exploration methods that are simple and easily implementable. A more sophisticated exploration scheme that actively adapts to user feedback (e.g., [37]) is likely to reduce the amount of exploration needed to obtain meaningful gains, but the added complexity makes it more challenging to use. The question of what exploration scheme would lead to the largest gains while minimizing the adverse effect on user experience remains an interesting open problem. Another natural question arising out of this work is that of improved methods for detecting underperforming queries. One approach could be to use guided exploration in the presented results to detect changes in the performance of the ranker. We conjecture that the gap between the performance of the LTR methods and that of the click-based methods explored here is due to the context-insensitivity of the current state of the art, tree-based LTR algorithms, which tend not to make extensive use of query features, as well as other contextual features. If true, then a natural direction for further research would be to devise LTR algorithms that make better use of contextual information.

Acknowledgments. This research was supported by Ahold, Amsterdam Data Science, the Bloomberg Research Grant program, the Dutch national program COMMIT, Elsevier, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the ESF Research Network Program ELIAS, the Royal Dutch Academy of Sciences (KNAW) under the Elite Network Shifts project, the Microsoft Research Ph.D. program, the Netherlands eScience Center under project number 027.012.105, the Netherlands Institute for Sound and Vision, the Netherlands Organisation for Scientific Research (NWO) under project nrs 727.011.005, 612.001.116, HOR-11-10, 640.006.013, 612.066.930, CI-14-25, SH-322-15, 652.002.001, 612.001.551, the Yahoo Faculty Research and Engagement Program, and Yandex. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, 2006.

[2] P. Bennett, M. Shokouhi, and R. Caruana. Implicit preference labels for learning highly selective personalized rankers. In *ICTIR*, 2015.

[3] J. Bian, X. Li, F. Li, Z. Zheng, and H. Zha. Ranking specialization for web search: A divide-and-conquer approach by using topical RankSVM. In *WWW*, 2010.

[4] L. Bottou et al. Counterfactual reasoning and learning systems: The example of computational advertising. *JMLR*, 14:3207–3260, 2013.

[5] C. Burges, R. Ragno, and Q. Le. Learning to rank with non-smooth cost functions. In *NIPS*, 2007.

[6] C. J. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Techn. Report MSR-TR-2010-82, MSR, June 2010.

[7] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *JMLR-Proceedings Track*, 14:1–24, 2011.

[8] A. Chuklin, I. Markov, and M. de Rijke. *Click Models for Web Search*. Morgan & Claypool, 2015.

[9] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM*, 2008.

[10] P. Donmez, K. M. Svore, and C. J. C. Burges. On the local optimality of LambdaRank. In *SIGIR*, 2009.

[11] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *TOIS*, 23(2): 147–168, 2005.

[12] X. Geng, T.-Y. Liu, T. Qin, A. Arnold, H. Li, and H.-Y. Shum. Query dependent ranking using k-nearest neighbor. In *SIGIR*, 2008.

[13] G. Giannopoulos, U. Brefeld, T. Dalamagas, and T. Sellis. Learning to rank user intent. In *CIKM*, 2011.

[14] A. Grotov, S. Whiteson, and M. de Rijke. Bayesian ranker comparison based on historical user interactions. In *SIGIR*, 2015.

[15] F. Guo, C. Liu, and Y.-M. Wang. Efficient multiple-click models in web search. In *WSDM*, 2009.

[16] A. Hassan, R. W. White, and Y.-M. Wang. Toward self-correcting search engines: Using underperforming queries to improve search. In *SIGIR*, 2013.

[17] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in learning to rank online. In *ECIR*, 2011.

[18] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM*, 2011.

[19] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing historical interaction data for faster online learning to rank for IR. In *WSDM*, 2013.

[20] K. Hofmann, S. Whiteson, and M. de Rijke. Fidelity, soundness, and efficiency of interleaved comparison methods. *TOIS*, 31(4), 2013.

[21] K. Hofmann, S. Whiteson, and M. de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval*, 16(1):63–90, 2013.

[22] K. Hofmann, L. Li, and F. Radlinski. Online evaluation for information retrieval. *Foundations and Trends in Information Retrieval*, 2016. To appear.

[23] A. Jain and G. Mishne. Organizing query completions for web search. In *CIKM*, 2010.

[24] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

[25] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in Web search. *TOIS*, 25(2), 2007.

[26] M. Karimzadehgan and C. Zhai. A learning approach to optimizing exploration-exploitation tradeoff in relevance feedback. *Information Retrieval*, 16:307–330, 2013.

[27] Y. Kim, A. Hassan, R. W. White, and Y.-M. Wang. Playing by the rules: Mining query associations to predict search performance. In *WSDM*, 2013.

[28] B. Kveton, C. Szepesvari, Z. Wen, and A. Ashkan. Cascading bandits: Learning to rank in the cascade model. In *ICML*, 2015.

[29] D. Lefortier, P. Serdyukov, and M. de Rijke. Online exploration for detecting shifts in fresh intent. In *CIKM*, 2014.

[30] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *WSDM*, 2011.

[31] L. Li, S. Chen, A. Gupta, and J. Kleban. Counterfactual estimation and optimization of click metrics in search engines: A case study. In *WWW (Companion)*, 2015.

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

[33] Z. Liu, G. Singh, N. Parikh, and N. Sundaresan. A large scale query logs analysis for assessing personalization opportunities in e-commerce sites. In *WSCD*, 2014.

[34] Microsoft Learning to Rank Datasets, 2012. http://research.microsoft.com/en-us/projects/mslr/default.aspx.

[35] T. Moon, W. Chu, L. Li, Z. Zheng, and Y. Chang. An online learning framework for refining recency search results with user click feedback. *TOIS*, 30(4):20:1–20:28, 2012.

[36] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *KDD*, 2005.

[37] F. Radlinski and T. Joachims. Active exploration for learning rankings from clickthrough data. In *KDD*, 2007.

[38] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *ICML*, 2008.

[39] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM*, 2008.

[40] A. Slivkins, F. Radlinski, and S. Gollapudi. Ranked bandits in metric spaces: learning diverse rankings over large document collections. *JMLR*, 14(1):399–436, 2013.

[41] A. L. Strehl, J. Langford, L. Li, and S. M. Kakade. Learning from logged implicit exploration data. In *NIPS*, 2011.

[42] J. Teevan, D. J. Liebling, and G. Ravichandran Geetha. Understanding and predicting personal navigation. In *WSDM*, 2011.

[43] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML*, 2009.