Reusing Historical Interaction Data for Faster Online Learning to Rank for IR

Katja Hofmann k.hofmann@uva.nl
Anne Schuth a.g.schuth@uva.nl
Shimon Whiteson s.a.whiteson@uva.nl
Maarten de Rijke derijke@uva.nl

ISLA, University of Amsterdam

ABSTRACT

Online learning to rank for information retrieval (IR) holds promise for allowing the development of "self-learning" search engines that can automatically adjust to their users. With the large amount of e.g., click data that can be collected in web search settings, such techniques could enable highly scalable ranking optimization. However, feedback obtained from user interactions is noisy, and developing approaches that can learn from this feedback quickly and reliably is a major challenge.

In this paper we investigate whether and how previously collected (historical) interaction data can be used to speed up learning in online learning to rank for IR. We devise the first two methods that can utilize historical data (1) to make feedback available during learning more reliable and (2) to preselect candidate ranking functions to be evaluated in interactions with users of the retrieval system. We evaluate both approaches on 9 learning to rank data sets and find that historical data can speed up learning, leading to substantially and significantly higher online performance. In particular, our preselection method proves highly effective at compensating for noise in user feedback. Our results show that historical data can be used to make online learning to rank for IR much more effective than previously possible, especially when feedback is noisy.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: H.3.3 Information Search and Retrieval

Keywords

Information retrieval, Interleaved comparisons, Learning to Rank

1. INTRODUCTION

In recent years, learning to rank methods have become popular in information retrieval (IR) as a means of tuning retrieval systems to the requirements of specific search environments, groups of users, or individual users [18]. However, most current approaches work *offline*, meaning that manually annotated data needs to be collected beforehand, and that, once deployed, the system cannot continue to adjust to user needs, unless it is retrained with additional data.

WSDM'13, February 4–8, 2013, Rome, Italy.

An alternative setting is *online* learning to rank, where the system learns directly from interactions with its users [10]. These approaches are typically based on reinforcement learning (RL) techniques [25], meaning that the system tries out new ranking functions (also called rankers), and learns from feedback inferred from users' interactions with the presented rankings. In contrast to offline learning to rank approaches, online approaches do not require any initial training material, but rather automatically improve rankers while they are being used.

A main challenge that online learning to rank for IR approaches have to address is to learn as quickly as possible from the limited quality and quantity of feedback that can be inferred from user interactions. Learning speed is particularly important in terms of the number of user interactions. The better the system's performance is after a smaller number of interactions, the more likely users are to be satisfied with the system. Also, the more effective an online learning to rank algorithm is, the more feasible it is to adapt to smaller groups of users, or even individual users.[1] Furthermore, user feedback is limited because the learning algorithm should be invisible to system users, i.e., feedback is inferred from natural (noisy) user interactions.

In this paper, we address the following question: *Can previously observed (historical) interaction data be reused to speed up online learning to rank?* Current online learning to rank approaches for IR utilize each observed data sample (consisting of a query, the displayed results, and observed user clicks on the result list) only once. This was necessary because, until recently, it was not clear how feedback from previous user interactions (that were collected with different rankers) could be reused. However, a recently developed probabilistic method for inferring relative feedback [9] has been shown to allow data re-use [11] for ranker evaluation. It was found to be effective for making ranker comparisons more reliable, especially when large amounts of historical data were available. Here, we investigate whether and how this evaluation method can be integrated with online learning to rank approaches, and whether and in what way these additional (historical, and possibly noisier or biased) evaluations can lead to faster learning.

Specifically, we make the following contributions.

- We propose the first two approaches for reusing historical data in online learning to rank for IR: *reliable historical comparisons* (RHC), which uses historical data directly to make feedback more reliable, and *candidate preselection* (CPS), which uses historical data to preselect candidate rankers.
- In extensive experiments using 9 learning to rank data sets, we

[1] In this paper we focus on the effectiveness of the learning algorithm, and assume a system is used by a group of users with similar search behavior and preferences. The question of how to form user groups to which to adapt is orthogonal to this work.

investigate whether and how historical data reuse can speed up online learning to rank and lead to higher online performance. In addition to investigating our proposed methods, we analyze the effect of noise in user feedback, and the effect of bias and variance in ranker comparisons based on historical data.

We find that historical data can be effectively reused to speed up online learning to rank for IR. Particularly effective is the CPS approach, which reuses historical data to preselect candidate rankers, and can thereby compensate for noise in user feedback. Our results directly impact the effectiveness of online learning to rank approaches, especially in settings where feedback may be noisy.

We discuss related work in §2, and detail our approaches in §3. Our experimental setup and results are described in §4 and §5 respectively. We analyze our results in §6 and conclude in §7.

2. RELATED WORK

In this section we first give a brief overview of learning to rank approaches for IR, with a focus on approaches for learning in an online setting. While the reuse of previously collected data has not been investigated in this setting, the area of off-policy evaluation in RL is related, and is briefly reviewed as well.

Learning to rank for information retrieval is an active research area, and many approaches have been proposed and refined in recent years [18]. The vast majority of these learn in a supervised manner, from manually annotated data. A number of semi-supervised approaches have been proposed more recently, which can, in addition to expensive labeled data, take into account unlabeled sample data, for example as a means of regularization [26].

Both supervised and semi-supervised approaches typically work offline, meaning that labeled training data need to be collected before training (and typically, the data are assumed to be sampled i.i.d. from some underlying distribution). Collecting training data is often expensive and, because annotators may interpret queries differently from actual users, it may not accurately capture users' preferences [22]. In contrast, online methods hold the promise of allowing learning while interacting with users of the retrieval system [10, 28]. In this setting, no training data is required before deploying the system (but any existing data could be used for bootstrapping the system), and the system is expected to transparently adapt to its users' true preferences.

Online learning to rank for IR has been modeled as a contextual bandit problem (also known as *bandits with side information* or *associative RL* [15, 24]), a type of IR problem where states (i.e., queries) are independent of each other [2, 3]. A difference between typical contextual bandit formulations and online learning to rank for IR is that in IR reward cannot be observed directly. Instead, feedback for learning can be inferred from observed user interactions with the result list, e.g., clicks can be used as noisy relative preference indications, allowing probabilistic comparisons between documents [13], or between lists of documents [21].

Several methods for online learning to rank for IR have been proposed. Most of these perform listwise learning, meaning that they learn from probabilistic comparisons between pairs of candidate rankers using [28, 29]. A first such method, Dueling Bandit Gradient Descent (DBGD) was proposed in [28]. This method implements stochastic gradient descent over a large or infinite space of ranking solutions. Alternatively, algorithms based on multi-armed bandit formulations have been developed to efficiently find the best ranking solutions of a given set [29]. Besides listwise online learning to rank approaches for IR, a pairwise approach was investigated in [10]. Our work is based on the listwise DBGD algorithm (cf., Section 3).

Central to the performance of DBGD is the choice of a reliable feedback mechanism. The algorithm learns using relative feedback, typically implemented in the form of an *interleaved comparison method* (i.e., a method for inferring relative comparisons between rankers). Previously, DBGD was evaluated with a method called Balanced Interleave (BI) [10, 14, 21]. An alternative is the Team Draft (TD) method, which addresses limitations of BI and proved reliable in large-scale IR evaluation [5, 21].

Until recently, it was not clear how interleaved comparison methods could reuse historical data. However, the recently developed Probabilistic Interleave (PI) method bridges this gap [9, 11]. PI is based on a probabilistic interpretation of interleaved comparisons, which allows it to infer comparison outcomes using data from arbitrary result lists, even if they were obtained in comparisons of rankers different from the current target rankers. Our approaches for learning with historical data reuse are enabled by this evaluation approach. To the best of our knowledge they are the first approaches that integrate comparisons based on historical data with an online learning to rank for IR approach.

From an RL perspective, our work is related to *off-policy learning* [19, 25]. Off-policy learning means that an algorithm takes actions according to one policy but uses the resulting feedback to learn about another policy. The PI method that forms the basis of our approaches can be seen as a form of off-policy learning.

Work on RL approaches for information retrieval and related areas is ongoing. In particular, solutions have been proposed in the areas of news recommendation [1, 16] and ad placement [15, 24]. Other approaches for off-policy evaluation have been developed for applications related to IR, such as news recommendation. An off-policy evaluation method for news recommendation [17] uses rejection sampling to compensate for differences in how frequently specific news articles were shown to users by the behavioral policy. However, this approach cannot be easily applied to the online learning to rank for IR setting, because here the number of actions (result lists) is very large so that a similar approach would require prohibitively large amounts of data. Our methods are the first to reuse historical data for online learning to rank in a setting where feedback is available through interleaved comparison.

3. METHOD

In this section we detail our method for online learning to rank for IR with re-use of historical data. First, we describe our problem formulation (cf., §3.1), and methods for online learning (§3.2) and for inferring feedback (§3.3) on which our work is based. Finally, we detail our two methods for reusing historical data in online learning to rank for IR (§3.4 and §3.5).

3.1 Problem Formulation

Our model of online learning to rank for IR follows [10]. In it, a retrieval system repeatedly observes queries q, submitted by users of the system. The system generates a result list $\mathbf{r}$ and presents it to the user. Users interact with $\mathbf{r}$, and any clicks $\mathbf{c}$ on result documents are again observed by the system. The system can use the observed q and $\mathbf{c}$ to update its ranking function, and is ready to observe the next query. We assume that queries are independent of each other, rendering the problem a contextual bandit problem (cf., §2) [2, 3].

The objective of our learning system is to optimize online performance, formulated as discounted cumulative reward.[2] Again following [10] we adopt the following formulation:

$$C = \sum_{t=0}^{\infty} \gamma^{t-1} reward_t(\mathbf{r}_t). \quad (1)$$

[2]Note that reward is not directly observed by the system, but is only used for external system evaluation.

Here, reward of the result list $\mathbf{r}_t$ is accumulated over an infinite number of timesteps t, and weighted by a *discount factor* $\gamma \in [0, 1)$. This means that immediate rewards are weighted higher than future rewards. The discount factor can be interpreted such that there is a $1 - \gamma$ probability that interactions will end after each timestep, e.g., if users abandon the search engine. Thus, cumulative reward weights the reward at each timestep by its probability of actually occurring and online performance is highest for systems that present the best rankings after the fewest user interactions.

3.2 Dueling Bandit Gradient Descent

Our baseline learning algorithm uses stochastic gradient descent to learn a weight vector $\mathbf{w}$ for a linear combination of ranking features. Given a query q and a document collection D, the relations between the query and each document are expressed as vectors of ranking features $\mathbf{X}$. Given a weight vector $\mathbf{w}$, document scores are obtained by $\mathbf{s} = \mathbf{wX}$. A result list for the given weight vector can then be generated by simply sorting the documents by score.

Algorithm 1 Baseline algorithm, based on [28].

1: **Input**: $f(\mathbf{l}_1, \mathbf{l}_2)$, $g(\delta, \mathbf{w})$, α, δ, $\mathbf{w}_0$, λ (default: 0)
2: $\mathbf{h} \leftarrow []$
3: **for** query q_t ($t \leftarrow 1..\infty$) **do**
4: $(\mathbf{w}'_t, \mathbf{u}_t) \leftarrow g(\delta, \mathbf{w}_t)$ *// generate candidate ranker*
5: $\mathbf{l}_1 = generate\_list(\mathbf{w_t})$; $\mathbf{l}_2 = generate\_list(\mathbf{w'_t})$
6: $(o_L, \mathbf{r}, \mathbf{c}) \leftarrow f(\mathbf{l}_1, \mathbf{l}_2)$
7: **if** $o_L > 0$ **then**
8: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \mathbf{u}_t$ *// update current best ranker*
9: **else**
10: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$
// maintain historical data if needed
11: **if** $\lambda > 0$ **then**
12: **if** $len(\mathbf{h}) = \lambda$ **then**
13: $remove(\mathbf{h}, \mathbf{h}[0])$
14: $append(\mathbf{h}, (\mathbf{r}, \mathbf{l}_1, \mathbf{l}_2, \mathbf{c}))$

Algorithm 2 $generate\_candidate(\cdot)$ (baseline method for generating candidate rankers, to be used as $g(\delta, \mathbf{w})$ in Algorithm 1).

1: **Input**: δ, $\mathbf{w}$
2: Sample unit vector $\mathbf{u}$ uniformly.
3: $\mathbf{w}' \leftarrow \mathbf{w} + \delta \mathbf{u}$
4: **return** $(\mathbf{w}', \mathbf{u})$

To learn weight vectors for ranking, we use DBGD [28], shown in Algorithm 1. Its first input is a comparison function $f(\mathbf{l}_1, \mathbf{l}_2)$, which compares two result lists $\mathbf{l}_1$ and $\mathbf{l}_2$ using user clicks $\mathbf{c}$ (the return value $o_L \in R$ indicates whether the quality of the two lists was inferred to be equal ($o_L = 0$), or whether the first ($o_L < 0$) or second ($o_L > 0$) list was inferred to be better; cf., 3.3). A second function, $g(\delta, \mathbf{w})$ is provided to generate candidate rankers. The remaining inputs are the step sizes α and δ, and an initial weight vector $\mathbf{w}_0$. An optional parameter λ indicates the maximum amount of most recent historic interaction data that the algorithm should keep in memory for possible reuse. This parameter is set to 0 in the baseline version.

The algorithm learns while interacting with search engine users as follows. At all times, the hypothesized best solution up to this point is maintained as $\mathbf{w}_t$. When a query q_t is observed, a new candidate weight vector $\mathbf{w}'_t$ is generated using $g(\cdot)$ (line 4). Then, the result lists generated for q_t using $\mathbf{w}_t$ and $\mathbf{w}'_t$ are compared using $f(\mathbf{l}_1, \mathbf{l}_2)$ (lines 5-6). If $\mathbf{w}'_t$ wins the comparison, $\mathbf{w}_t$ is updated using the update rule $\mathbf{w}_t \leftarrow \mathbf{w}_t + \alpha \mathbf{u}_t$ (line 8). Otherwise, $\mathbf{w}_t$ is not changed.

In the baseline version of this algorithm, $generate\_candidate(\cdot)$ is used to generate candidate weight vectors as follows (Algorithm 2). First, a vector $\mathbf{u}$ is generated by randomly sampling a unit vector. Then, $\mathbf{w}'$ is obtained by moving $\mathbf{w}$ by a step of size δ in the direction $\mathbf{u}$. An alternative method of candidate selection using historical data is presented in Section 3.5.

3.3 Probabilistic Interleave

Interleaved comparison methods are based on the observation that user clicks are context-dependent, i.e., interpreting them as relative feedback (in relation to surrounding documents) results in more reliable feedback than absolute interpretations [21]. Interleaved comparison methods aggregate clicks to obtain relative comparisons between result rankings, such as those generated by different rankers. Our methods for reusing historical data are based on PI [9, 11]. This interleaved comparison method enables historical data reuse because its probabilistic approach allows ranker comparisons using arbitrary result lists (e.g., result lists previously shown to users that were generated using rankers different from the ones currently compared).

Algorithm 3 Probabilistic Interleave, following [9], to be used as $f(\mathbf{l}_1, \mathbf{l}_2)$ in Algorithm 1

1: **Input**: $o(\mathbf{r}, \mathbf{l}_1, \mathbf{l}_2, \mathbf{c})$, $\mathbf{l}_1$, $\mathbf{l}_2$, τ, κ
2: $\mathbf{r} \leftarrow []$
3: **for** $i(1, 2)$ **do**
4: $\forall d \in \mathbf{l}_i : T_i \leftarrow P_i(d) = \frac{1}{index(\mathbf{l}_i, d)^\tau} Z_{i,d}$
// $Z_{i,d}$ is a normalization factor such that $\sum_{d \in \mathbf{l}_i} P_i(d) = 1$
5: **while** $(len(\mathbf{r}) < \kappa) \wedge ((\exists i : \mathbf{l}_1[i] \notin \mathbf{r}) \vee (\exists i : \mathbf{l}_2[i] \notin \mathbf{r}))$ **do**
6: $l_s \leftarrow 1$ **if** $random\_bit()$ **else** 2
7: $l_o \leftarrow 2$ **if** $l_s = 1$ **else** 1
8: $d_{next} \leftarrow sample\_without\_replacement(T_{l_s})$
9: $remove(T_{l_o}, d_{next})$
10: $append(\mathbf{r}, d_{next})$
// present $\mathbf{r}$ to user and observe clicks $\mathbf{c}$
// then compute the comparison outcome, e.g., using Eqs. 2–5
11: $o \leftarrow o(\mathbf{r}, \mathbf{l}_1, \mathbf{l}_2, \mathbf{c})$
12: **return** $(o, \mathbf{r}, \mathbf{c})$

The baseline version of PI (without historical data) is shown in Algorithm 3. Its first input is a comparison method $o(\cdot)$, that computes a comparison outcome given an interleaved list $\mathbf{r}$, original lists $\mathbf{l}_1$ and $\mathbf{l}_2$, and observed user clicks $\mathbf{c}$ (optionally, historical data $\mathbf{h}$ can be passed in). It also accepts the two lists $\mathbf{l}_1$ and $\mathbf{l}_2$ to be compared, a decay parameter τ, and the maximum desired length of the interleaved result list κ. The first step is to transform $\mathbf{l}_1$ and $\mathbf{l}_2$ into probability distributions over documents T_i (lines 3–4). The shapes of both distributions are identical and are determined by the original result lists and τ, such that the most highly ranked documents of an original result list are the most likely to be sampled. These distributions are used to generate an interleaved result list $\mathbf{r}$, which ensures that all possible permutations of documents $d \in \mathbf{l}_1 \cup \mathbf{l}_2$ can be observed with some probability $P > 0$.

During interleaving, the distribution to be used to contribute a document at a given rank of the interleaved result list $\mathbf{r}$ is randomly selected (lines 6–7). From the selected distribution (T_{l_s}) a document is drawn without replacement (8). The document is removed from the other distribution and appended to $\mathbf{r}$ (9–10). The process is repeated until all documents have been added or $\mathbf{r}$ is filled.

The interleaved result list is then shown to the user in response to the query and user clicks $\mathbf{c}$ are observed, where each entry in $\mathbf{c}$ indicates whether the corresponding document in $\mathbf{r}$ has been clicked. The observed clicks are used by the second step, *comparison*. In this step, the comparison outcome o is computed using $o(\cdot)$. In the

baseline version of PI, we use the scoring function from [9]. This approach uses knowledge of the probabilistic interleaving process sketched above to marginalize over possible outcomes as follows. Recall that during interleaving, distributions for contributing documents were selected randomly (lines 6–7). By observing which distribution contributed which document (in an assignment vector $\mathbf{a}$), we could obtain a comparison outcome. However, we can obtain more reliable outcomes by marginalizing over all possible assignments $\mathbf{a}_i \in \mathbf{A}$ that could have led to the observed $\mathbf{r}$, and weighting them by their probability $P(\mathbf{a}|\mathbf{r}, q)$. This marginalized scoring function computes outcomes as follows:

$$o = \sum_{i=1}^{|\mathbf{A}|} (n(\mathbf{a}_i, \mathbf{c}, 1) - n(\mathbf{a}_i, \mathbf{c}, 2)) P(\mathbf{a}_i|\mathbf{r}, q). \quad (2)$$

Here, $n(\mathbf{a}, \mathbf{c}, x)$ is a function that returns the number of clicks that, according to $\mathbf{a}$ were contributed by (the distribution based on) the original list $\mathbf{l}_x$. Following [9], we compute $P(\mathbf{a}|\mathbf{r}, q)$ using:

$$P(\mathbf{a}|\mathbf{r}, q) = \frac{P(\mathbf{r}|\mathbf{a}, q)P(\mathbf{a}|q)}{P(\mathbf{r}|q)} \quad (3)$$

$$P(\mathbf{a}|q) = P(\mathbf{a}) = |\mathbf{A}|^{-1} \quad (4)$$

$$P(\mathbf{r}|\mathbf{a}, q) = \prod_{i=1}^{len(\mathbf{r})} P(\mathbf{r}[i]|\mathbf{a}[\mathbf{r}[i]], \mathbf{r}[1, i-1], q), \quad (5)$$

where $P(\mathbf{r}[i])$ denotes the probability of observing document $\mathbf{r}[i]$ at rank i of the interleaved list $\mathbf{r}$ displayed for query q, given that it was contributed by the list specified in assignment $\mathbf{a}[i]$ and documents $\mathbf{r}[1, i-1]$ were already placed on $\mathbf{r}$ (obtained from $T_{1,2}$).

3.4 Reliable Historical Comparison

Based on DBGD and the interleaved comparison method PI, we can now define our first approach for reusing historical data to speed up online learning to rank. To enable this approach, we collect historical data by setting $\lambda > 0$ in Algorithm 1. Then, we use the collected historical data to supplement the interleaved comparisons based on live data as shown in Algorithm 4. We call this approach *reliable historical comparison* (RHC).

Algorithm 4 (RHC) Probabilistic interleaved comparison with reuse of historical data, to be used as $f(\mathbf{l}_1, \mathbf{l}_2)$ in Algorithm 1.

1: **Input**: $o_L(\mathbf{r}, \mathbf{l}_1, \mathbf{l}_2, \mathbf{c})$, $o_H(\mathbf{l}_1, \mathbf{l}_2, \mathbf{r}', \mathbf{l}'_1, \mathbf{l}'_2, \mathbf{c}')$, $\mathbf{r}$, $\mathbf{l}_1$, $\mathbf{l}_2$, $\mathbf{c}$, $\mathbf{h} = n \times (\mathbf{r}'_i, \mathbf{l}'_{i_1}, \mathbf{l}'_{i_2}, \mathbf{c}'_i)$
2: $o_L \leftarrow o_L(\mathbf{r}, \mathbf{l}_1, \mathbf{l}_2, \mathbf{c})$ *// compute live outcome following Eqs. 2–5*
// compute historical outcome, biased (Eqs. 2–5) or unbiased (Eq. 6)
3: $\mathbf{o}_H \leftarrow []$
4: **for** $(\mathbf{r}'_i, \mathbf{l}'_{i_1}, \mathbf{l}'_{i_2}, \mathbf{c}'_i) \in \mathbf{h}$ **do**
5: $append(\mathbf{o}_H, o_H(\mathbf{l}_1, \mathbf{l}_2, \mathbf{r}', \mathbf{l}'_1, \mathbf{l}'_2, \mathbf{c}'))$
6: $\beta_L \leftarrow 1$
7: $\beta_H \leftarrow var(\mathbf{o}_H)$
8: $o_C \leftarrow (\beta_L * mean(\mathbf{o}_H) + \beta_H * mean(o_L))/(\beta_L + \beta_H)$
9: **return** o_C

RHC takes as input two methods for computing outcomes, o_L, which accepts data from one live observation, and o_H, which accepts as input the current target lists as well as one historical observation. Further, RHC takes as input one live observation, i.e., the interleaved list, original lists, and clicks obtained when interleaving the original lists $\mathbf{l}_1$ and $\mathbf{l}_2$ that are currently compared. In addition, it accepts n historical data points that were observed in previous comparisons of two different result lists $\mathbf{l}'_1$ and $\mathbf{l}'_2$. The algorithm first generates the live outcome as before (cf., Algorithm 3), using the live outcome method o_L (line 2). Then, additional outcome estimates are computed using the historical data and $o_H(\cdot)$.

In this paper we instantiate o_H in two ways to explore the effects of bias and variance on this approach. Previous work showed that applying PI directly to historical data results in biased estimates of comparison outcomes [11]. An alternative method, that compensates for bias using importance sampling, is unbiased, but can suffer from high variance when only little data is available. Both methods were previously applied to an evaluation task. Their effectiveness was similar for relatively small amounts of data, while for large amounts of data the unbiased method was more reliable. In contrast, in the online learning to rank task addressed here, we expect the effect of bias and variance to be relatively small, because amounts of historical data are small, and because subsequent ranker pairs are relatively more similar to those used to obtain the historical samples than in the evaluation setting addressed previously.

Our first (biased) instantiation of o_H uses the same scoring method as PI to estimate outcomes for the current target lists given historical data (Eq. 2). It uses the historical $\mathbf{r}'$ and $\mathbf{c}'$ to count clicks given assignments ($n(\mathbf{a}, \mathbf{c}', x)$), but uses distributions $T_{1,2}$ based on the current target lists $\mathbf{l}_1$ and $\mathbf{l}_2$. The resulting comparison method computes outcomes based on historical data but may be biased. Under the current target rankers, document distributions may be different from those under which the historical data was collected. This means that it is not guaranteed that each target ranker has an equal chance of contributing its highly ranked documents to the interleaved list, and to obtain clicks on these documents. As a result, the target list that is more similar to the historical lists has an advantage over the less similar one.

To address the problem of bias when historical data is used for computing interleaved comparison outcomes, we use importance sampling, as proposed in [11]. Importance sampling is a statistical method for compensating for differences between distributions, when data collected under one (original) distribution is used to infer information about a different (target) distribution. This approach is frequently used in e.g., off-policy reinforcement learning [19].

Our second instantiation of o_H (with importance sampling) is:

$$o_{IS} = \sum_{i=1}^{|\mathbf{A}|} (n(\mathbf{a}_i, \mathbf{c}', 1) - n(\mathbf{a}_i, \mathbf{c}', 2)) P(\mathbf{a}_i|\mathbf{r}', q') \frac{P(\mathbf{r}'|q')}{P'(\mathbf{r}'|q')}. \quad (6)$$

As in the biased scoring method, $n(\cdot)$ counts the clicks each target list would obtain given an assignment and the original data ($\mathbf{c}'$). The obtained click difference is weighted by the probability of each assignment. In contrast to the biased method, this outcome is then weighted by its probability of occurring under the target distribution ($P(\cdot)$) versus the original (historical) distribution ($P'(\cdot)$). Intuitively, this means that observations that are more likely under the target distribution, and less likely under the original distribution, obtain a high weight and vice versa. It has been shown that outcomes obtained using importance sampling are unbiased (i.e., they converge to the expected outcome under the target distribution in the limit) [23]. While o_{IS} is unbiased, it may suffer from high variance when the target and source distributions are very different from each other, which may lead to unreliable outcome estimates.

After computing the live and historical estimates o_L and $\mathbf{o}_H$, they are combined into a final estimate o_C using the Graybill-Deal estimator [7] (line 6–8). This combined estimator weights the two estimates by the ratio of their variances. It was shown to result in a minimal variance combined estimate when the variances of the individual estimators are known, and to have strictly lower variance than either individual estimate when their variances are estimated on samples of size $n > 10$ [7].

A limitation of combining historical and live estimates according to Algorithm 4 is that for any given comparison we only have one live data point collected under the current target rankers, so that the variance of the live outcome(s) cannot be estimated. Here, we set the weight of the live outcomes to $\beta_L = 1$.[3] Our experiments in §4 below investigate whether this approximation is sufficient for improved performance. We hypothesize that the reliability of comparisons can be improved using RHC, leading to faster learning.

3.5 Candidate Preselection

Our second approach for reusing historical data to speed up online learning to rank for IR uses historical data to improve candidate generation. Instead of randomly generating a candidate ranker to test in each comparison, it generates a pool of candidate rankers, and selects the most promising one using historical data. We hypothesize that historical data can be used to identify promising rankers, and that the increased quality of candidate rankers can speed up learning. We call this second approach *candidate preselection* (CPS).

Algorithm 5 (CPS) Generating candidate rankers with preselection, for use as $g(\delta, \mathbf{w}_t)$ in Algorithm 1.

1: **Input**: $o_H(\mathbf{l}_1, \mathbf{l}_2, \mathbf{r}', \mathbf{l}'_1, \mathbf{l}'_2, \mathbf{c}')$, $\mathbf{w}_t$, δ, ζ, η, $\mathbf{h} = n \times (\mathbf{r}'_i, \mathbf{l}'_{i_1}, \mathbf{l}'_{i_2}, \mathbf{c}'_i)$
2: $\mathbf{e} = []$
// generate candidate pool
3: **for** i in $(i = 1..\eta)$ **do**
4: $append(\mathbf{e}, generate\_candidate(\delta, \mathbf{w}_t))$
// compare and eliminate candidates using historical data
5: **while** $len(\mathbf{e}) > 1$ **do**
6: $\mathbf{p} \leftarrow sample(\mathbf{e}, 2)$
7: $o_H \leftarrow []$
8: **for** i $(i = 1..\zeta)$ **do**
9: $(\mathbf{r}'_i, \mathbf{l}'_{i_1}, \mathbf{l}'_{i_2}, \mathbf{c}'_i) \leftarrow sample(\mathbf{h}, 1)$
10: $append(o_H, o_H(l(\mathbf{p}[1].w), l(\mathbf{p}[2].w), \mathbf{r}', \mathbf{l}'_1, \mathbf{l}'_2, \mathbf{c}'))$
11: **if** $mean(o_H) < 0$ **then**
12: $remove(\mathbf{e}, \mathbf{p}[2])$
13: **else if** $mean(o_H) > 0$ **then**
14: $remove(\mathbf{e}, \mathbf{p}[1])$
15: **else**
16: $remove(\mathbf{e}, sample(\mathbf{p}, 1))$
17: **return** $\mathbf{e}[0]$ *// return remaining candidate*

Our implementation of CPS is shown in Algorithm 5, which replaces the method $generate\_candidate(\cdot)$ in our baseline method (cf., Algorithm 1). As input it takes a comparison function $o_H(\cdot)$ that estimates comparison outcomes using historical data, a current weight vector $\mathbf{w}_t$, the step size δ, arguments η and ζ that determine the size of candidate pools and the number of historical comparisons to conduct per ranker pair, and a vector of historical observations $\mathbf{h}$.

The algorithm is called when a new candidate ranker is requested. It first generates a pool of η candidate rankers by calling the original $generate\_candidate(\cdot)$ function (Algorithm 2) (lines 3–4). The most promising ranker is determined in rounds, where in each round a randomly selected pair of rankers (line 6) competes. For each pair, ζ comparisons are performed on historical data points randomly sampled from $\mathbf{h}$ with replacement (8–10). After the individual historical estimates are obtained, their mean is used to determine which ranker to eliminate from the pool. If there is a winner (i.e., $o_H \neq 0$), the losing ranker is removed. Otherwise, one of the rankers is selected to be removed at random. When only one element remains in the candidate pool, it is returned as the most promising candidate.

Our candidate selection approach ensures that a single candidate is selected after a finite $((\eta - 1) \times \zeta)$ number of comparisons. Because only the best candidate needs to be selected, the randomized approach is expected to provide a good balance of effectiveness and efficiency. In cases where the compared candidates perform equally well, only one candidate needs to be retained (chosen randomly).

Like in §3.4 above, we investigate the effect of bias and variance on this approach by implementing the comparison method o_H in two different ways, (1) using the biased (but low variance) method defined by Eqs. 2–5, and (2) using the unbiased (potentially high variance) method that uses importance sampling (Eq. 6).

We hypothesize that CPS can substantially improve the quality of candidate rankers available for online learning, leading to faster learning than using live data only. Regarding the biased and unbiased version of CPS, we expect only small performance differences, as the amount of historical data reused per live comparison is small.

4. EXPERIMENTS

Our experiments are designed to investigate whether online learning to rank for IR can be sped up by using historical data. They are based on an existing simulation framework. This framework combines fully annotated learning to rank data sets with probabilistic user models to simulate user interactions with a search engine that learns online. The use of this framework allows us to investigate characteristics of our proposed learning approaches without the potential risk of hurting a real system's performance. Below, we provide details of the evaluation framework, data sets, and user models, as well as the performance metrics and parameter settings used in our experiments.

The simulation framework works as follows. An incoming user query is modeled as random sampling from a pool of queries provided with a learning to rank data set. The system generates a result list to respond to the query using the features of candidate documents that are provided with the data set (this is similar to ranking candidate documents after an initial matching/filtering step as would be done while interacting with real users). Displaying the generated result list to the user and observing clicks is simulated by applying our user model, which generates clicks probabilistically on the basis of the annotations available with the data set. The clicks are sent back to the retrieval system, where feedback is inferred and learned from. Then, the cycle continues with the next simulated user query.

We conduct all our experiments on the 9 data sets provided as LETOR 3.0 and 4.0 (LEarning TO Rank) [18]. All data sets contain feature vectors that represent the relationships between queries and documents that are typical for various search settings. In addition, the (manually assessed) relevance level of each document-query pair is provided. Finally, all data sets are pre-split by query for 5-fold cross validation.

The following search tasks are implemented: the data set *OHSUMED* models a literature search task, based on a query log of a search engine for the MedLine abstract database. This data set contains 106 queries that implement an informational search task. The remaining 8 data sets are based on TREC Web track tasks run between 2003 and 2008. The datasets *HP2003*, *HP2004*, *NP2003*, and *NP2004* implement navigational tasks, homepage finding and named-page finding respectively. *TD2003* and *TD2004* implement an informational task: topic distillation. All six data sets are based on the .GOV document collection, a crawl of the .gov domain, and contain between 50 and 150 queries and approximately 1000 judged documents per query. A more recent document collection, .GOV2

[3] We also experimented with batches of comparisons where the same original pair was used for several subsequent comparisons. However, the performance loss due to the resulting smaller number of updates outweighed the gain due to improved variance estimates.

formed the basis of *MQ2007* and *MQ2008*. These data sets contain 1700 and 800 queries respectively, but far fewer judged documents per query. The data sets *OHSUMED*, *MQ2007* and *MQ2008* are annotated with graded relevance judgments (3 grades, from 0, not relevant, to 2, highly relevant), while the remaining data sets are labeled using binary assessments.

To generate clicks, we employ a user model based on the Dependent Click Model (DCM) [8],[4] an extension of the Cascade Model [6] that has been shown to be effective in explaining users' click behavior in web search. The model explains position bias (i.e., the observation that higher-ranked results are much more likely to be clicked than lower-ranked ones) by positing that users start examining documents at the top of a result list. For each document they examine, they determine whether the document representation (e.g., consisting of title, snippet and URL) appears promising enough to warrant a click (we model this step of deciding to click with a click probability given some relevance label $P(C|R)$). After each click, users decide whether they are satisfied with the information provided in the clicked document(s) and they want to stop examining further results (with stop probability $P(S|R)$), or if they want to continue examining results.

Table 1: Overview of the click models used.

	click probabilities			*stop probabilities*		
relevance grade	0	1	2	0	1	2
perfect	0.0	0.5	1.0	0.0	0.0	0.0
navigational	0.05	0.5	0.95	0.2	0.5	0.9
informational	0.4	0.7	0.9	0.1	0.3	0.5

The user model instantiations used in our experiments are provided in Table 1. First, the *perfect* click model provides reliable feedback, and is used to obtain an upper bound on performance. The second (*navigational*) and third (*informational*) model reflect the two types of search tasks implemented by our data sets, as well as increasing levels of noise (i.e., smaller differences in click probabilities for different relevance levels). For each instance, the table provides the click and stop probabilities given a relevance grade R. For example, under the navigational model, simulated users would be very likely to encounter a highly relevant document ($P(C|2) = 0.95$), and very likely to stop examining documents once they clicked on such a document ($P(S|2) = 0.9$). Under the informational model, users are less likely to stop, and click probabilities for the different relevance grades are much more similar, resulting in a higher level of noise. For data sets with binary relevance judgments, only the two extremes are used.

Our experiments compare and contrast three baseline runs and four experimental runs:

BI Baseline – learning with live data only, using Balanced Interleave [14, 21] for interleaved comparisons. Balanced Interleave generates interleaved result lists using a mostly deterministic process, where only the starting list is selected at random. Comparisons are made based on the number of clicks each original list would obtain above a cutoff.

TD Baseline – learning with live data only, using Team Draft [20, 21] for interleaved comparisons. Team Draft randomizes interleaving per pair of ranks, and assigns clicks to the original list that contributed a document.

PI Baseline – learning with live data only, using PI (cf., §3.3).

[4]Models that take additional information into account have been shown to more accurately reflect click behavior [27], but these make stronger assumptions rendering experiments unnecessarily complex.

RHC-B Uses historical data to infer more reliable feedback (cf., §3.4), with *biased* comparison outcome estimates (Eq. 2).

RHC-U Uses historical data to infer more reliable feedback (cf., §3.4), with *unbiased* comparison outcome estimates (Eq. 6).

CPS-B Uses historical data for candidate preselection (cf., §3.5) with *biased* comparison outcome estimates (Eq. 2).

CPS-U Uses historical data for candidate preselection (cf., §3.5) with *unbiased* comparison outcome estimates (Eq. 6).

Our main evaluation metric is online performance, measured as the cumulative discounted reward obtained over the course of T interactions between system and user (cf. Eq. 1). This means that we measure the utility of result lists presented to search engine users while learning. Note that, although we run our experiments over a limited number of interactions, discounting ensures that our measured performance is a close approximation of the online performance that could be obtained in an infinite horizon setting.

To approximate the true utility of a search engine result page to a user, we compute Normalized Discounted Cumulative Gain (NDCG) on the top $\kappa = 10$ results shown to the user, a standard evaluation metric in information retrieval [4]:[5]

$$NDCG = \sum_{i=1}^{len(\mathbf{r})} \frac{2^{rel(\mathbf{r}[i])} - 1}{\log_2(i+1)} iNDCG^{-1}. \quad (7)$$

This metric sums over the gain that is based on the relevance label ($rel(\mathbf{r}[i])$) of each document, and divides it by a discount factor (based on the log of the rank i at which the document was presented). This sum is then normalized by the ideal NDCG (iNDCG) that would be obtained on an ideal document ranking.

For each data set, we run experiments over 1000 iterations (i.e., simulated interactions), and repeat each experiment 25 times and average results over all folds and repetitions. Parameters for the DBGD learning algorithm are selected to match those found to work best in previous work ($\mathbf{w}_0$ is initialized to zero, $\alpha = 0.01$, $\delta = 1$, cf., [28]). For the remaining parameters, we report results on one setting (for CPS, $\eta = 6$, $\zeta = 10$, and $\lambda = 10$; for RHC, $\lambda = 10$). The sensitivity to specific parameter settings is analyzed in §6. We test for statistically significant differences using a two-sided student's t-test. To ensure repeatability of our experiments, we make all source code used to obtain our results available online.[6]

5. RESULTS

In this section we present the results of our experiments, described in §4, to answer our main research question: *Can historical interaction data be reused to speed online learning to rank?* In addition to our main question, we also investigate *how* historical data can best be reused (i.e., to improve the reliability of evaluations, as in our RHC approach, or to improve the quality of candidate rankers, as in CPS), and whether and how historical data reuse is affected by bias and variance in outcome estimates based on historical data.

Table 2 shows the online performance obtained on all LETOR 3.0 and 4.0 data sets, for the three click model instantiations specified in Table 1, the three baseline runs BI, TD, PI, and the four experimental runs RHC-B, RHC-U, CPS-B and CPS-U. For reasons discussed below, BI outperforms the other baseline methods. Therefore, we use BI as our baseline for significance testing.

[5]Note that our formulation differs from earlier ones, including the one provided with the LETOR toolkit, where documents at rank 2 are not discounted (cf., [12]). Here, we use the formulation from [4] so that relevance differences at the highest ranks can be detected.

[6]Available from http://ilps.science.uva.nl/resources/online-learning-framework.

		BI	TD	PI	RHC-B	RHC-U	CPS-B	CPS-U
				perfect click model				
1	*HP2003*	107.84	108.94	97.62$^{\blacktriangledown}$	95.51$^{\blacktriangledown}$	95.60$^{\blacktriangledown}$	**118.24**$^{\blacktriangle}$	116.72$^{\blacktriangle}$
2	*HP2004*	99.82	99.72	89.72$^{\blacktriangledown}$	87.49$^{\blacktriangledown}$	88.54$^{\blacktriangledown}$	**109.12**$^{\blacktriangle}$	108.40$^{\blacktriangle}$
3	*NP2003*	97.94	98.15	88.67$^{\blacktriangledown}$	87.03$^{\blacktriangledown}$	89.05$^{\blacktriangledown}$	**108.82**$^{\blacktriangle}$	108.04$^{\blacktriangle}$
4	*NP2004*	102.96	102.72	93.50$^{\blacktriangledown}$	93.12$^{\blacktriangledown}$	92.93$^{\blacktriangledown}$	113.97$^{\blacktriangle}$	**114.35**$^{\blacktriangle}$
5	*TD2003*	40.38	38.81	36.21$^{\blacktriangledown}$	34.10$^{\blacktriangledown}$	33.79$^{\blacktriangledown}$	**47.26**$^{\blacktriangle}$	47.22$^{\blacktriangle}$
6	*TD2004*	36.19	35.87	34.55$^{\blacktriangledown}$	30.82$^{\blacktriangledown}$	31.60$^{\blacktriangledown}$	43.96$^{\blacktriangle}$	**44.72**$^{\blacktriangle}$
7	*OHSUMED*	70.68	70.14	68.29$^{\blacktriangledown}$	64.23$^{\blacktriangledown}$	65.29$^{\blacktriangledown}$	**72.57**	72.36
8	*MQ2007*	59.87	60.18	58.23$^{\blacktriangledown}$	58.69$^{\triangledown}$	58.22$^{\blacktriangledown}$	**64.30**$^{\blacktriangle}$	63.60$^{\blacktriangle}$
9	*MQ2008*	79.03	77.98	75.57$^{\blacktriangledown}$	76.08$^{\blacktriangledown}$	77.09$^{\blacktriangledown}$	84.35$^{\blacktriangle}$	**84.68**$^{\blacktriangle}$
				navigational click model				
10	*HP2003*	93.82	93.52	79.15$^{\blacktriangledown}$	85.29$^{\blacktriangledown}$	84.22$^{\blacktriangledown}$	114.93$^{\blacktriangle}$	**116.24**$^{\blacktriangle}$
11	*HP2004*	83.23	82.54	69.28$^{\blacktriangledown}$	77.25$^{\blacktriangledown}$	76.66$^{\blacktriangledown}$	106.12$^{\blacktriangle}$	**107.48**$^{\blacktriangle}$
12	*NP2003*	88.00	85.07	74.77$^{\blacktriangledown}$	78.75$^{\blacktriangledown}$	80.47$^{\blacktriangledown}$	106.81$^{\blacktriangle}$	**107.58**$^{\blacktriangle}$
13	*NP2004*	90.53	88.50	79.43$^{\blacktriangledown}$	85.81$^{\blacktriangledown}$	86.84$^{\triangledown}$	**113.62**$^{\blacktriangle}$	112.39$^{\blacktriangle}$
14	*TD2003*	34.60	33.20	30.99$^{\blacktriangledown}$	30.53$^{\blacktriangledown}$	31.07$^{\blacktriangledown}$	**44.77**$^{\blacktriangle}$	44.56$^{\blacktriangle}$
15	*TD2004*	32.71	31.78	30.72$^{\blacktriangledown}$	27.60$^{\blacktriangledown}$	27.78$^{\blacktriangledown}$	**41.11**$^{\blacktriangle}$	39.85$^{\blacktriangle}$
16	*OHSUMED*	67.54	67.96	65.10$^{\blacktriangledown}$	62.60$^{\blacktriangledown}$	62.27$^{\blacktriangledown}$	68.16	**70.28**$^{\triangle}$
17	*MQ2007*	58.42	58.92	56.33$^{\blacktriangledown}$	55.71$^{\blacktriangledown}$	57.36	61.61$^{\blacktriangle}$	**61.63**$^{\blacktriangle}$
18	*MQ2008*	76.14	76.32	72.84$^{\blacktriangledown}$	74.21$^{\triangledown}$	74.66$^{\triangledown}$	81.41$^{\blacktriangle}$	**81.70**$^{\blacktriangle}$
				informational click model				
19	*HP2003*	55.63	55.65	46.87$^{\blacktriangledown}$	65.54$^{\blacktriangle}$	63.92$^{\blacktriangle}$	104.46$^{\blacktriangle}$	**107.61**$^{\blacktriangle}$
20	*HP2004*	42.99	45.02	37.52$^{\triangledown}$	56.23$^{\blacktriangle}$	55.66$^{\blacktriangle}$	83.04$^{\blacktriangle}$	**89.17**$^{\blacktriangle}$
21	*NP2003*	53.38	52.88	45.38$^{\blacktriangledown}$	65.63$^{\blacktriangle}$	64.25$^{\blacktriangle}$	103.09$^{\blacktriangle}$	**103.34**$^{\blacktriangle}$
22	*NP2004*	58.31	57.62	52.32$^{\triangledown}$	71.48$^{\blacktriangle}$	72.81$^{\blacktriangle}$	96.20$^{\blacktriangle}$	**99.88**$^{\blacktriangle}$
23	*TD2003*	22.11	21.99	21.74	25.20$^{\blacktriangle}$	26.12$^{\blacktriangle}$	38.47$^{\blacktriangle}$	**38.86**$^{\blacktriangle}$
24	*TD2004*	23.66	22.87	21.60$^{\blacktriangledown}$	22.35	22.22	29.63$^{\blacktriangle}$	**30.27**$^{\blacktriangle}$
25	*OHSUMED*	63.39	**64.91**	60.48$^{\blacktriangledown}$	58.65$^{\blacktriangledown}$	58.47$^{\blacktriangledown}$	61.05	64.29
26	*MQ2007*	55.29	54.58	53.99	54.79	54.37	56.35	**57.76**$^{\blacktriangle}$
27	*MQ2008*	73.14	71.83	70.42$^{\blacktriangledown}$	70.89$^{\blacktriangledown}$	70.42$^{\blacktriangledown}$	76.63$^{\blacktriangle}$	**76.97**$^{\blacktriangle}$

Table 2: Online performance (in terms of discounted cumulative reward) when learning with interleaved comparison methods. Best runs per row are highlighted in bold. Statistically significant improvements (losses) from the baseline method *BI* are indicated by $^{\triangle}$ ($p = 0.05$) and $^{\blacktriangle}$ ($p = 0.01$) ($^{\triangledown}$ and $^{\blacktriangledown}$).

Overall, we see that the highest online performance is achieved by our CPS method for most data sets and click models. The observed improvements over BI are statistically significant with $p < 0.01$, and substantial. For example, for the data set *HP2003*, the highest performance under perfect click feedback is 118.24 (using biased estimates of comparison outcomes), which constitutes an improvement of 8.5% over the best-performing baseline method TD (cf., row 1). The only exception is the data set *OHSUMED*, for which performance under the perfect and informational click models is statistically equivalent for BI and CPS (rows 7 and 25).

To put the obtained absolute online performance scores in perspective, recall that we measure discounted cumulative gain, i.e., high online performance is obtained when a method both learns well (i.e., it achieves high offline performance, in terms of NDCG), and it learns quickly, i.e., after a small number of interactions. In our experimental setup, a method that presented perfect result lists (with NDCG $= 1$) on all interactions could obtain an online performance of 200.0, while a method that would obtain no reward on the first 500 interactions and perfect results after would achieve an online performance of only 15.0. Scores obtained by our methods fall between these two extremes, indicating that good rankers are learned within a few hundred simulated interactions.

For the baseline methods, which learn from live data only, we find that online performance of BI and TD is very similar, while that of PI is significantly lower for most data sets and click models. To understand why, we compare the offline performance and learning speed of these methods (cf., the offline performance on data set *NP2003* in Figure 1).[7] We see that the final offline performance is very similar (differences are not statistically significant), and that they also learn equally fast. Thus, PI learns as well as BI and TD, but loses online performance due to the increase in randomization during interleaving. Compared to the evaluation setting, where PI was shown to outperform BI and TD when applied to compare rankers over large amounts of data, PI performs worse than expected [9]. PI provides fine-grained information about the magnitude of ranker difference, which leads to more accurate comparisons when outcomes are aggregated over repeated samples. However, in the online learning to rank setting with live data only, ranker comparisons are based on a single data sample, and information about the magnitude of ranker differences is lost. Developing online learning methods that exploit this additional information is a direction for future work.

Our methods that learn from historical data are enabled by PI, meaning that to improve performance over the best-performing baseline, BI, the methods need to learn substantially faster, to overcome the initial performance loss incurred by the randomization inherent to PI. For the CPS method, we see that this is the case. The method achieves much higher online performance than any of the methods

[7] For brevity we present offline performance plots for only one data set. Plots for the remaining data sets are qualitatively similar.

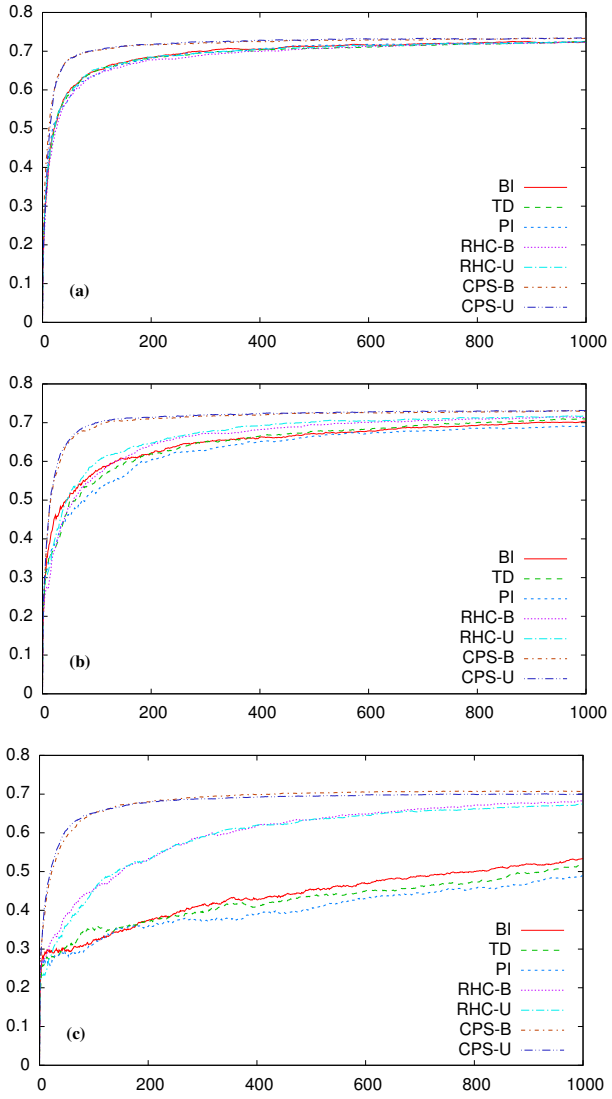

Figure 1: Offline performance (computed on held-out test queries after each learning step) on *NP2003* data set, for the *perfect*, *navigational*, and *informational* click models.

that learn with live data only. Furthermore, the performance gain for reusing historical data is particularly big when click feedback is noisy, with gains up to 107% (row 20). Looking at offline performance (Figure 1), we observe that CPS learns faster than methods that learn from live data only. In particular, the speed-up is biggest for the noisy informational click model, suggesting that the CPS method can effectively limit the effect of noise in click feedback.

Performance for our method RHC is generally lower than that obtained by CPS or the baselines that only take live data into account. However, under noisy feedback (the informational click model) online performance is significantly higher than that obtained by BI on 5 of the 9 data sets (rows 19–23). Comparing again to offline performance, we find that RHC learns as quickly as the baseline methods, but cannot make as effective use of the learned rankers, similar to PI. However, when feedback is noisy, the method does succeed in making comparisons more reliable (cf., part (c) of Figure 1). Thus, we conclude that RHC can effectively leverage historical data to make ranker comparisons more reliable, but this results in performance gains only when click feedback is indeed noisy.

The effect of removing bias is small overall. Of the differences between RHC-B and RHC-U, only that in row 26 (*MQ2007*, informational click model) is statistically significant. This small effect is expected, because for the relatively few samples that are combined in historical outcome estimates (e.g., for $\lambda = 10$), removing bias can only have a small effect on absolute outcomes. However, several patterns do emerge. For RHC, using the unbiased method (RHC-U) improves online performance for 5 of the 9 data sets, and performance decreases in 4 cases. For the navigational click model, differences are smaller, and the trend continues for the informational model. It appears that for RHC, removing bias can slightly improve learning. This can be explained by the direction of the bias. We use the λ most recent historical points. For these, the original and target distributions are expected to be most similar, so bias or variance should be smallest for these samples. Also, the original distribution will typically be biased towards the current best ranker $\mathbf{w}_t$, as it was used before. Therefore, there is a slightly increased chance of observing clicks on documents highly ranked by $\mathbf{w}_t$, giving a disadvantage to the new candidate ranker. This means that, due to this bias, a good candidate may be missed, leading to slower learning. When feedback is noisier, this effect is masked by the noise.

For CPS, the biased version of the method performs slightly better than CPS-U under perfect feedback (none of the differences are statistically significant). However, CPS-U performs better when click feedback is noisier. It appears that, when feedback is very reliable (perfect user model), the increase in noise due to removing bias may slow learning more than can be compensated for by the bias removal. However, increases in click noise can amplify the effect of bias. CPS evaluates several rankers distributed around the current best ranker, $\mathbf{w}_t$. It is itself not evaluated using historical data, but the candidate rankers that generate rankings that are more similar to the current (or previous) best ranker have an advantage. This advantage is stronger when click feedback is noisier, because highly-ranked documents that are not relevant to the query are more likely to be clicked than under more reliable feedback. This bias leads to slower learning and the observed lower online performance for CPS-B under noisy feedback. Overall, we can conclude that the unbiased methods should be preferred over biased methods, especially when click feedback is expected to be noisy.

Our main results show consistently high performance for our CPS method, which can achieve significantly and substantially higher online performance than all other methods tested. We find that reusing historical data using CPS allows faster learning than with current online learning to rank methods that take only live data into account. In the next section, we analyze our results in more detail.

6. ANALYSIS

In this section, we first compare the performance of our methods to supervised learning to rank approaches (§6.1). Then we compare our methods' sensitivity to parameter settings (§6.2 and 6.3).

6.1 Comparison with Previous Work

Most previous work on learning to rank for IR focused on supervised approaches, and measured the offline performance achieved by learners after all training data had been processed. Our approach is fundamentally different, as it learns online, from relative feedback observed on the result lists presented to users. Despite this more limited form of feedback, previous work showed that online learning to rank can achieve competitive offline performance, at least when click feedback is reliable [10].

To allow for some comparison with supervised learning to rank approaches, we show the offline performance achieved by CPS-U in terms or LETOR NDCG at different cutoffs on the perfect and informational click models in Table 3. Despite the degenerate quality of feedback available in the online setting, final performance

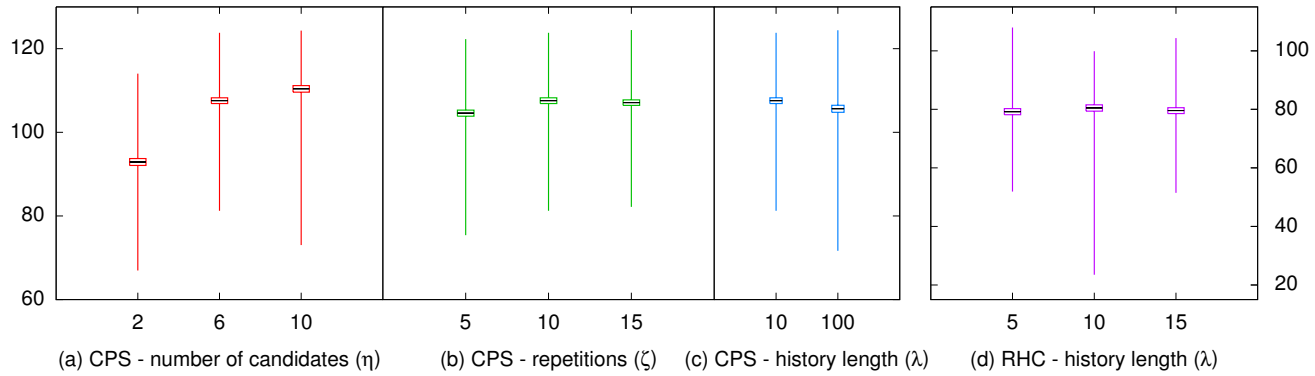

Figure 2: Online performance (min, max, and mean with standard error) after 1000 iterations for different settings of parameters (a) η, (b) ζ, and (c) λ for CPS and (d) λ for RHC on the *NP2003* data set and *navigational* click model.

after 1000 iterations is reasonable, compared to supervised learning to rank methods [18]. For example, final performance of CPS-U under perfect feedback outperforms supervised regression on four out of the seven LETOR 3.0 data sets, and it outperforms other supervised listwise approaches (RankBoost and/or FRank) on three data sets. Even though performance under the much noisier informational feedback is significantly lower, offline performance remains competitive.

	perfect			informational		
	N@1	**N@3**	**N@10**	**N@1**	**N@3**	**N@10**
HP2003	0.685	0.745	0.777	0.661	0.715	0.749
HP2004	0.533	0.662	0.719	0.533	0.662	0.719
NP2003	0.524	0.696	0.754	0.508	0.669	0.730
NP2004	0.523	0.684	0.746	0.478	0.632	0.691
TD2003	0.286	0.273	0.273	0.226	0.243	0.249
TD2004	0.307	0.279	0.250	0.188	0.200	0.188
OHSUMED	0.362	0.380	0.367	0.326	0.342	0.335
MQ2007	0.261	0.296	0.331	0.236	0.270	0.306
MQ2008	0.277	0.361	0.461	0.244	0.325	0.431

Table 3: Offline performance after 1000 iterations in terms of LETOR NDCG and cutoffs 1, 3, and 10 for CPS-U under the perfect and informational click models.

6.2 CPS - Sensitivity to Parameter Settings

Above, we reported results for only one set of parameters. Here, we investigate the sensitivity of our CPS method to changes in these parameters. CPS has three parameters: the history length λ (default: 10), the size of the candidate pool η (default: 6), and the number of historical comparisons per candidate pair ζ (default: 10). The algorithm is linear in η and ζ per live update ($O(\eta\zeta)$). An increase in λ does not significantly affect the run time of the algorithm, but only determines the number of historical samples kept in memory, from which the samples for candidate comparisons are selected.

Figure 2 (parts a–c) shows the online performance achieved by CPS-U under the navigational click model on the data set *NP2003* when varying one parameter at a time. For η, the size of the candidate pool, a smaller pool ($\eta = 2$) leads to a decrease in performance of 13.6% percent. Increasing the number of candidates to 10 increases online performance to 110.67, a much smaller change of 2.9%. This suggests that the performance reported above (§5) can be further increased by using larger candidate pools. However, returns are expected to diminish as ever more candidates are used.

For the number of repetitions (ζ), effects are much smaller. Here, increasing ζ to 15 leads to no significant improvement in mean online performance, although comparisons are slightly more reliable (shown by the higher minimum).

Increasing the history length to $\lambda = 100$ significantly decreases the performance of CPS. The reason is that the more recent historical samples used with a smaller λ are collected on ranker pairs that are more similar to the current candidate rankers. When older samples are used instead, the variance of historical outcome estimates increases (under CPS-B, bias would increase), leading to diminished performance.

Overall, we find that performance under CPS can be further improved by increasing the size of the candidate pool. For the remaining parameters, performance is relatively stable and decreases gracefully when less optimal settings are used. Finally, our analysis indicates that additional computational resources are best spent on increasing the size of the candidate pool (η). Although the linear increase in computation is expected to lead to sub-linear performance gains, developers of deployed applications are typically willing to invest in additional computing time when it translates to even small performance gains (while in a scientific setting computational resources limit what experiments are feasible to run).

6.3 RHC - Sensitivity to Parameter Settings

RHC has only one parameter, the history length λ (cf., Algorithms 1 and 4, default: 10). This parameter determines how many historic data points are kept in memory, and are used to compare the current best ranker $\mathbf{w}_t$ to the candidate ranker $\mathbf{w}'_t$. This method is linear in λ per live update.

The sensitivity of RHC-U to changes in λ is shown in Figure 2, part (d). Setting $\lambda = 5$ leads to a statistically significant decrease in online performance of this algorithm. However, this decrease constitutes a performance change of only 1.6%. Increasing the amount of history used to $\lambda = 15$ has no significant effect on online performance. We can conclude that the performance of this algorithm is relatively robust to changes in λ (thus, investing in additional resources to increase λ is not recommended).

7. CONCLUSION

In this paper we investigated whether and how historical data can be reused to speed up online learning to rank for IR. We proposed two approaches for integrating estimates based on historical data with a stochastic gradient descent algorithm for online learning to rank. Our first approach, RHC, uses historical comparison estimates to complement live comparisons and to make them more reliable. Our second approach, CPS, uses historical data for preselecting candidate rankers, thereby improving the quality of the rankers that are evaluated in live interactions with search engine users.

Our experimental evaluation of the proposed methods, based on the 9 LETOR data sets and 3 click models that allowed us to investi-

gate online performance of the methods under varying levels of click noise, yielded several insights. First, we found that the CPS method can substantially and significantly speed up online learning to rank in IR. We observed high gains in online performance over methods that use live data only for all click models. Second, performance gains of CPS were particularly high when click feedback was noisy. This result demonstrates that CPS is effective in compensating for noise in click feedback. Third, RHC was found to make ranker comparisons more reliable. However, positive effects on learning were observed only under noisy feedback and performance gains were lower than those obtained by CPS. Finally, we found that compensating for bias in click feedback had only small effects on online performance but that unbiased variants of our methods generally performed better than biased ones.

This work is the first to show that historical data can be used to significantly and substantially improve online performance in online learning to rank for IR. In addition, our approaches and experiments pointed at interesting directions for future work. While making comparisons themselves more reliable resulted in only minor speed-up in learning, major improvements were obtained when improving the quality of candidate rankers used for learning. This suggests that designing more elaborate candidate selection strategies, possibly taking properties of the solution space into account, is a promising direction for future research.

Acknowledgements

This research was partially supported by the European Union's ICT Policy Support Programme as part of the Competitiveness and Innovation Framework Programme, CIP ICT-PSP under grant agreement nr 250430, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreements nr 258191 (PROMISE Network of Excellence) and 288024 (LiMoSINe project), the Netherlands Organisation for Scientific Research (NWO) under project nrs 612.061.814, 612.061.815, 640.004.802, 727.011.-005, 612.001.116, HOR-11-10, the Center for Creation, Content and Technology (CCCT), the Hyperlocal Service Platform project funded by the Service Innovation & ICT program, the WAHSP and BILAND projects funded by the CLARIN-nl program, the Dutch national program COMMIT, by the ESF Research Network Program ELIAS, and the Elite Network Shifts project funded by the Royal Dutch Academy of Sciences.

8. REFERENCES

[1] D. Agarwal, B. Chen, P. Elango, N. Motgi, S. Park, R. Ramakrishnan, S. Roy, and J. Zachariah. Online models for content optimization. In *NIPS'08*, pages 17–24, 2008.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2): 235–256, May 2002.

[3] A. G. Barto, R. S. Sutton, and P. S. Brouwer. Associative search network: A reinforcement learning associative memory. *IEEE Trans. Syst., Man, and Cybern.*, 40:201–211, 1981.

[4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05*, pages 89–96, 2005.

[5] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue. Large-scale validation and analysis of interleaved search evaluation. *ACM Trans. Inf. Syst.*, 30(1):6:1–6:41, 2012.

[6] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM '08*, pages 87–94, 2008.

[7] F. Graybill and R. Deal. Combining unbiased estimators. *Biometrics*, 15(4):543–550, 1959.

[8] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM '09*, pages 124–131, 2009.

[9] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM '11*, pages 249–258, 2011.

[10] 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*, 2012. doi: 10.1007/s10791-012-9197-9.

[11] K. Hofmann, S. Whiteson, and M. de Rijke. Estimating interleaved comparison outcomes from historical click data. In *CIKM '12*, 2012.

[12] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4): 422–446, October 2002.

[13] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02*, pages 133–142, 2002.

[14] T. Joachims. Evaluating retrieval performance using clickthrough data. In J. Franke, G. Nakhaeizadeh, and I. Renz, editors, *Text Mining*, pages 79–96. Physica/Springer, 2003.

[15] J. Langford, A. Strehl, and J. Wortman. Exploration scavenging. In *ICML '08*, pages 528–535, 2008.

[16] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW '10*, pages 661–670, 2010.

[17] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *WSDM '11*, pages 297–306, 2011.

[18] T. Y. Liu. Learning to rank for information retrieval. *Found. and Trends in Inf. Retr.*, 3(3):225–331, 2009.

[19] D. Precup, R. Sutton, and S. Singh. Eligibility traces for off-policy policy evaluation. In *ICML'00*, pages 759–766, 2000.

[20] F. Radlinski and N. Craswell. Comparing the sensitivity of information retrieval metrics. In *SIGIR '10*, pages 667–674, 2010.

[21] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM '08*, pages 43–52, 2008.

[22] M. Sanderson. Test collection based evaluation of information retrieval systems. *Found. and Trends in Inf. Retr.*, 4(4): 247–375, 2010.

[23] D. Schuurmans. Greedy importance sampling. In *NIPS '99*, 1999.

[24] A. Strehl, C. Mesterharm, M. Littman, and H. Hirsh. Experience-efficient learning in associative bandit problems. In *ICML '06*, pages 889–896, 2006.

[25] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, USA, 1998.

[26] M. Szummer and E. Yilmaz. Semi-supervised learning to rank with preference regularization. In *CIKM '11*, pages 269–278, 2011.

[27] D. Xu, Y. Liu, M. Zhang, S. Ma, and L. Ru. Incorporating revisiting behaviors into click models. In *WSDM '12*, pages 303–312, 2012.

[28] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML'09*, pages 1201–1208, 2009.

[29] Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The k-armed dueling bandits problem. *Journal of Computer and System Sciences*, 78(5):1538 – 1556, 2012.