Probabilistic Multileave for Online Retrieval Evaluation

Anne Schuth[1] Robert-Jan Bruintjes[2] Fritjof Büttner[2] Joost van Doorn[2]
Carla Groenland[2] Harrie Oosterhuis[2] Cong-Nguyen Tran[2] Bas Veeling[2]
Jos van der Velde[2] Roger Wechsler[2] David Woudenberg[2] Maarten de Rijke[3]

[1]anne.schuth@uva.nl [2]firstname.lastname@student.uva.nl [3]derijke@uva.nl

University of Amsterdam, Amsterdam, The Netherlands

ABSTRACT

Online evaluation methods for information retrieval use implicit signals such as clicks from users to infer preferences between rankers. A highly sensitive way of inferring these preferences is through interleaved comparisons. Recently, interleaved comparisons methods that allow for simultaneous evaluation of more than two rankers have been introduced. These so-called multileaving methods are even more sensitive than their interleaving counterparts. Probabilistic interleaving—whose main selling point is the potential for reuse of historical data—has no multileaving counterpart yet. We propose *probabilistic multileave* and empirically show that it is highly sensitive and unbiased. An important implication of this result is that historical interactions with multileaved comparisons can be reused, allowing for ranker comparisons that need much less user interaction data. Furthermore, we show that our method, as opposed to earlier sensitive multileaving methods, scales well when the number of rankers increases.

Categories and Subject Descriptors

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

Keywords

Information retrieval; Evaluation; Interleaved comparisons; Multileaved comparisons

1. INTRODUCTION AND RELATED WORK

Search engines are constantly being improved upon. Engineers may propose changes to a ranker and such changes need be evaluated. Evaluation should not only compare alternative rankers against the current best ranker, but also against each other as these comparisons are important for guiding ranker development efforts. Pairwise preferences between rankers can be obtained in several ways. Traditional offline metrics based on relevance assessments can be computed [2] in a TREC-style evaluation setting [20].

Besides being expensive and time consuming, assessors' judgments do not necessarily agree with actual users' preferences [16]. In contrast, online metrics infer preferences directly from user interactions [12]. Interleaved comparison methods do so while being one to two orders of magnitude more sensitive than A/B testing [1, 9, 10, 15]. An interleaving method presents a user with a ranked result list that contains documents from two rankers and estimates user preferences by interpreting interactions with the interleaved list. Many variations of interleaving methods exist. *Balanced interleave* (BI) [11] randomly selects a ranker to start with and then, alternatingly, documents that have not been picked are picked from the two input rankings. BI can result in a preference for a ranker irrespective of where a user clicks. This bias is corrected in *team draft interleave* (TDI) [15]. More recent methods include *document constraints* (DC) [4] and *optimized interleave* (OI) [14]. Recently, interleaving outcomes were made to agree with A/B tests [19].

Interleaving methods have an important disadvantage: they can only evaluate two rankers at a time. When the number of rankers that have to be compared increases, this quickly becomes infeasible. Schuth et al. [18] have proposed multileaved comparison methods as a way of directly inferring preferences among a set of rankers. Instead of requiring rankers to be compared pairwise, their method allows for a single comparison of multiple rankers at once. Multileaved comparison methods produce a list that contains results from multiple rankers and use clicks on this list to infer a ranking over rankers. Schuth et al. [18] propose extensions of TDI and OI to *team draft multileave* (TDM) and *optimized multileave* (OM), respectively. Both learn preferences much faster than interleaving methods. Each has its own merit: TDM learns slightly slower but results in more accurate preferences while OM initially learns faster and scales better when the number of compared rankers goes up.

The interleaving and multileaving methods listed so far can only infer preferences among rankers based on interaction data that is generated using those very same rankers, thus preventing generalizations to new and unseen rankers. *Probabilistic interleave* (PI), however, is a recent interleaving method that can reuse *historical interaction data* collected using other rankers than the ranker being evaluated to infer preferences between rankers [6, 7]. This allows for comparisons that are more efficient in terms of how many user impressions are required for reliable comparisons.

We propose an extension of PI to *probabilistic multileave* (PM): a multileaving method that is able to reuse historical interaction data. An evaluation method is *sensitive* if it quickly detects differences in the quality of rankings. That is, if rankers are of different quality, the evaluation method should detect those differences with the least number of comparisons. An evaluation method is *unbiased* if all the rankers are equal in expectation when clicks are random. That is, the method should evaluate rankers fairly and only on the basis of their actual performance. We show experimentally: (1) that PM is at least as sensitive as TDM (which in turn is more sensitive than OM); (2) that PM is unbiased; and (3) that PM scales well when the number rankers that are compared increases. An important implication of our results is that historical interactions with multileaved comparisons

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.

ACM 978-1-4503-3621-5/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2766462.2767838.

Algorithm 1 Creating a multileaved list in PM.

1: **Input**: rankers $\mathcal{R}$, size k.
2: $L \leftarrow []$ *// initialize interleaved list*
3: **while** True **do**
4: $\quad \mathcal{R}' \leftarrow \mathcal{R}$
5: $\quad$ **while** $|\mathcal{R}'| > 0$ **do**
6: $\quad\quad \mathcal{R}_j \leftarrow$ draw from uniform distribution over $\mathcal{R}'$
7: $\quad\quad \mathcal{R}' \leftarrow \mathcal{R}' \setminus \{\mathcal{R}_j\}$
8: $\quad\quad d \leftarrow$ draw from $\mathcal{R}_j$ with $P(d|\mathcal{R}_j)$ *// see (1)*
9: $\quad\quad L.append(d)$
10: $\quad\quad$ **if** $|L| = k$ **then**
11: $\quad\quad\quad$ **return** L
12: $\quad\quad$ **for** $\mathcal{R}_j \in \mathcal{R}$ **do**
13: $\quad\quad\quad \mathcal{R}_j.remove(d)$ *// renormalize $P(d|\mathcal{R}_j)$*

can be reused, allowing for ranker comparisons that need much less user interaction data. This, in turn, means that users are exposed less often to inferior rankers and that more rankers can be compared with the same number of user interactions.

2. METHOD

In this section we derive *probabilistic multileave* (PM), a multileaving version of *probabilistic interleaving* (PI). Extending PI to multiple rankers is non-trivial due to three challenges. First, the interleaving method, which combines multiple rankings to one list, must be extended to accommodate more than two rankers in a probabilistic way; see Section 2.1. Secondly, we need to represent the outcomes of a multileaved comparison such that it allows for more than two rankers. Lastly, we need to extend the marginalization over all assignments[1] to the multileave setting without increasing the number of possible assignments beyond what we can reasonably handle.

2.1 Constructing multileaved lists

Probabilistic interleaving [6] constructs interleaved lists in a way similar to TDI. For each rank in the interleaved list a coin is flipped to decide which ranker assigns the next document. Instead of picking the top document from that ranking, like in TDI, the document is drawn from a softmax distribution. After a document has been chosen it is removed from all rankings it occurs in and all softmax distributions are renormalized. An extension of this approach to the setting with r rankers can be found in Algorithm 1. As in TDM, and unlike PI, we choose to introduce the notion of rounds on line 5: in each round *all* rankers are chosen at random to contribute a document. This way we increase the change of the multileaved list to include documents from the top of each ranker that is to be compared while ensuring non-zero probability for all possible multileavings.

Documents d are drawn from ranker $\mathcal{R}_j$ with probability

$$P(d|\mathcal{R}_j) = \frac{\frac{1}{r_j(d)^\tau}}{\sum_{d' \in D} \frac{1}{r_j(d')^\tau}}, \tag{1}$$

where $r_j(d)$ denotes the rank of the document in ranker $\mathcal{R}_j$. When documents are removed from $\mathcal{R}_j$, as is done on line 13, this changes the distribution. The softmax decay is controlled by τ which we set to 3, following PI [6].

2.2 Inferring preferences

Once the multileaved list has been created, shown to a user, and once the user has clicked on zero or more documents, these clicks can be interpreted as preferences for rankers.

[1]Following TDM and PI, documents in a multileaved list are assigned to a rankers' teams in order to distribute credit from clicks.

Algorithm 2 Inferring preferences in PM.

1: **Input**: multileaved list L, rankers $\mathcal{R}$, clicks C.
2: $A \leftarrow \{\}$ *// assignment tree keeps track of outcome and probabilities*
3: $A' \leftarrow \{\langle o \leftarrow [0, ..., 0], p \leftarrow 0\rangle\}$ *// init assignment tree, $|o| = |\mathcal{R}|$*
4: **for** $d \in L$ **do**
5: $\quad A \leftarrow A', A' \leftarrow \{\}$ *// next layer in assignment tree*
6: $\quad$ **for** $\mathcal{R}_j \in \mathcal{R}$ **do**
7: $\quad\quad p_j \leftarrow P(d|\mathcal{R}_j)$ *// see (1)*
8: $\quad\quad \mathcal{R}_j.remove(d)$ *// renormalize $P(d|\mathcal{R}_j)$*
9: $\quad$ **for** $\langle o, p\rangle \in A$ **do**
10: $\quad\quad$ **for** $\mathcal{R}_j \in \mathcal{R}$ **do**
11: $\quad\quad\quad$ **if** $random() > \frac{1}{|\mathcal{R}|} \cdot n^{\frac{1}{|L|}}$ **then**
12: $\quad\quad\quad\quad$ **continue** *// sample, skip branch*
13: $\quad\quad\quad p' \leftarrow p + \log(\frac{p_j}{2})$ *// log probability of assigning d to $\mathcal{R}_j$*
14: $\quad\quad\quad o' \leftarrow o$ *// copy outcome vector from parent*
15: $\quad\quad\quad$ **if** $d \in C$ **then**
16: $\quad\quad\quad\quad o'_j \leftarrow o'_j + 1$ *// click on d, increment outcome for $\mathcal{R}_j$*
17: $\quad\quad\quad A' \leftarrow A' \cup \{\langle o', p'\rangle\}$ *// append to next layer*
18: $o \leftarrow [0, ..., 0]$ *// init outcome, $|o| = |\mathcal{R}|$*
19: **for** $\langle o', p'\rangle \in A'$ **do**
20: $\quad o \leftarrow o + o' \cdot e^{p'}$ *// aggregate outcome vector o weighted with $e^{p'}$*
21: **return** o

Hofmann et al. [6] proposes an unbiased estimator of the expected outcome of a comparison outcome over many such query impressions as

$$E[o(C, A)] \approx \frac{1}{|Q|} \sum_{q \in Q} o(c_q, a_q). \tag{2}$$

Here, Q denotes a given set of queries, with clicks c_q and assignments a_q. In the PI implementation, $o(c, a) \in \{-1, 0, 1\}$ is used to denote the outcome of a comparison, which we change to $o_j(c, a) \in \mathbb{N}$ to denote the outcome (credit) for a single ranker $\mathcal{R}_j$. This outcome simply counts how many clicked documents were assigned to this ranker, as in TDI or TDM. Changing the outcome notation leads us to also compute the expected outcome per ranker $\mathcal{R}_j$ as

$$E[o_j(C, A)] \approx \frac{1}{|Q|} \sum_{q \in Q} o_j(c_q, a_q). \tag{3}$$

The PI algorithm then proposes a marginalization over all possible assignments. The intuition behind why this is a good idea for both PI and PM is that through a single interaction with users, inferences can be performed as if the $\mathcal{R}$ rankers were multileaved many more times. This allows for highly sensitive comparisons.

We follow the marginalization of PI closely, and marginalize over all possible assignments as

$$E[o_j(C, A)] \approx \frac{1}{|Q|} \sum_{q \in Q} \sum_{a_q \in \tilde{A}_q} o_j(c_q, a_q) \cdot P(a_q|L_q, q). \tag{4}$$

In this equation, $P(a_q|l_q, q)$ denotes the probability of an assignment a_q occurring given the interleaved list L_q for query q. We borrow this probability directly from the PI implementation.

A major difference with PI, however, is that we can no longer consider all possible assignments A as there are generally too many of them, namely $|\mathcal{R}|^{|L|}$, which, even with a small amount of rankers, is prohibitively large. Instead, we limit the number of assignments by taking an, in expectation, random and uniform sample from them. We denote the sample as $\tilde{A}$, and control the size $n \approx |\tilde{A}|$ by, deciding with probability $\frac{1}{|\mathcal{R}|} \cdot n^{\frac{1}{|L|}}$ whether we consider a *branch* of assignments in a *tree* of assignments not. This sampling happens

on line 11 in Algorithm 2 which lists our method for inferring preferences in PM. The algorithm builds a tree of assignments by considering assigning each document $d \in L$ (line 4) to all possible rankers $\mathcal{R}_j \in \mathcal{R}$ (line 10) with the probability mentioned before (line 11). The algorithm keeps track of the outcome for each ranker for each assignment branch (line 16). It also tracks the probability of each such assignment (line 13). Once the outcomes $o_j(c_q, a_q)$ have been computed for all assignments $a_q \in A'$, we aggregate them weighted with their assigned probabilities on line 20 into a single outcome. From this outcome we construct the preference matrix $\hat{P}_{ij}$ just as is done in TDM.

Note that in our implementation, just as in PI, we use log probabilities and sums to avoid buffer underflow. Also following PI, as an optmization, we only actually compute assignments and probabilities up to the lowest clicks.

3. EXPERIMENTS

Our implementation of probabilistic multileaving, PM, is evaluated experimentally for sensitivity, bias, and scalability using the experimental setup detailed below.

The methodology used to evaluate the different interleaving and multileaving methods is borrowed from [18]. Several features (i.e., BM25, LMIR.JM, Sitemap, PageRank, HITS, and TF.IDF) from the NP2003 and NP2004 dataset [13] are used as different rankers. These rankers can then be evaluated against each other using the aforementioned interleaving and multileaving methods. Evaluation is done by simulating users interacting with the interleaving and multileaving methods. In this setup, described by Hofmann [5], clicks are produced based on a *cascade click model* (CCM) [3] with four instantiations.

	$P(click = 1\|R)$		$P(stop = 1\|R)$	
Relevance R	$R = 0$	$R = 1$	$R = 0$	$R = 1$
perfect	0.0	1.0	0.0	0.0
navigational	0.05	0.95	0.2	0.9
informational	0.4	0.9	0.1	0.5
random	0.5	0.5	0.0	0.0

Queries are sampled from training data, clicks are generated based on relevance labels for each query. We use 1000 queries per run. Results are averaged over 25 repetitions over 5 folds for 2 datasets. We sample $|\mathcal{R}| = 5$ features as rankers for each run.

Ground truth NDCG [8] is computed for each ranker on held out test data. Then, just as in [18], a matrix P is defined using the difference in expected NDCG between the separate rankers:

$$P_{ij} = 0.5(NDCG(R_i) - NDCG(R_j)) + 0.5. \quad (5)$$

Any interleaving or multileaving method tries to estimate this matrix P. Interleaving methods do this by iterating over all pairs of rankers $i, j \in \mathcal{R}$, while multileaving methods compare all rankers $\mathcal{R}$ at once. To measure performance, we follow [18] and use a *binary error metric* to evaluate the correctness of this estimated matrix $\hat{P}$:

$$E_{bin} = \frac{\sum_{i,j \in \mathcal{R} \wedge i \neq j} sgn(\hat{P}_{ij} - 0.5) \neq sgn(P_{ij} - 0.5)}{|\mathcal{R}| \cdot (|\mathcal{R}| - 1)}.$$

After every simulated query this error can be computed for each interleaving and multileaving method.[2] We test significance with a two tailed student-t test.

To evaluate the *sensitivity* of PM, it is compared to TDM and PI. A multileaving method should be more sensitive than an interleaving method as it can compare more than two rankers at the same time. For this evaluation, we use the *perfect*, *navigational*, and *informational* instantiations. It is hypothesized that multileaving will need fewer impressions to converge than interleaving.

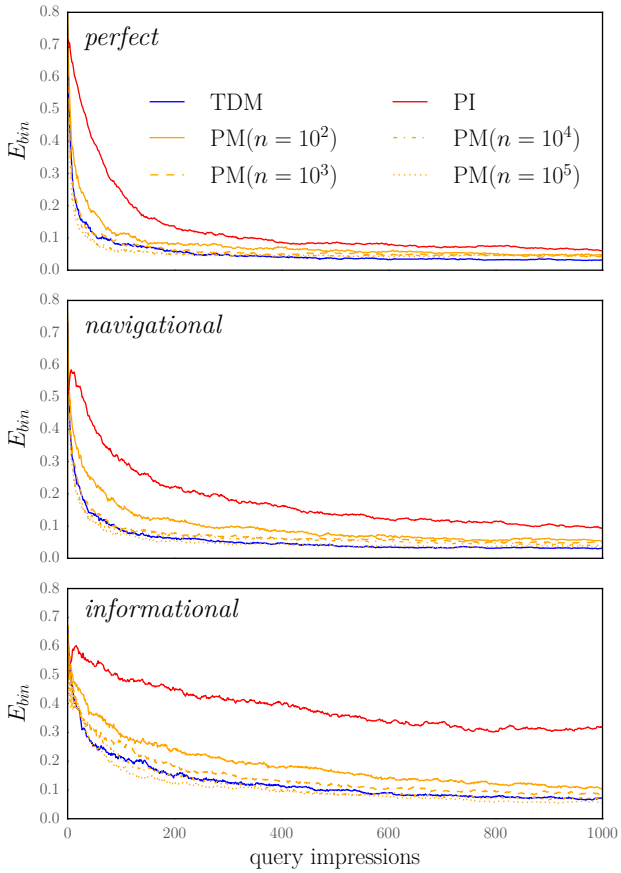

Figure 1: The binary error E_{bin} is plotted against the number of queries on which the error was evaluated for interleaving and multileaving methods. Clicks are generated by a *perfect*, *navigational*, and *informational* instantiations of CCM [3].

To assess the *bias* of all of the interleaving and multileaving methods mentioned above, we observe the error under a *random* clicking user. The error, in this case, is not based on ground truth NDCG matrix P_{ij} as in (5) but instead on a ground truth $P_{ij} = 0.5$ for all pairs of rankers i, j.

Lastly, we look into *scaling up* the number of rankers. The experiments listed above are all run using $|\mathcal{R}| = 5$ rankers; we also investigate what happens when $|\mathcal{R}| = 20$ for the navigational click model and with the sample size of PM fixed to $n = 10^4$.

4. RESULTS AND ANALYSIS

In Figure 1 we see that for impressions from all three click models, TDM performs similar to PM with a large number of samples n, although TDM performs slightly (but not significantly) better in the long run for the (unrealistic) *perfect* click model. The multileaving methods both perform much better than PI, as expected since the latter can only compare two rankers at a time.[3]

The binary error for both probabilistic multileaving quickly goes down to almost zero. Performance after 500 queries for the same experiments is found in Table 1 where we also perform statistical significance testing. In the table we see that PM always has a

[2] All code is open source and available at `https://bitbucket.org/ilps/lerot` [17].

[3] The bump for PI in the first few query impressions, that is mostly visible under *informational* feedback, is due to the fact that it takes a while for PI to have preferences for all pairs of rankers.

Table 1: Mean E_{bin} scores after 500 impressions for PM and SPM compared to baselines PI (first symbol) and TDM (second symbol). The symbol $^{\blacktriangle}$ means statistically better with $p < 0.01$ and $^{\triangle}$ for $p < 0.05$, whereas $^{\blacktriangledown}$ and $^{\triangledown}$ are their inverses.

	perfect	*navigational*	*informational*
PI	0.085 (0.08)	0.137 (0.11)	0.363 (0.15)
TDM	0.037 (0.06)	0.038 (0.05)	0.099 (0.09)
PM($n = 10^2$)	0.062 (0.07) $^{\blacktriangle\blacktriangledown}$	0.073 (0.07) $^{\blacktriangle\blacktriangledown}$	0.162 (0.10) $^{\blacktriangle\blacktriangledown}$
PM($n = 10^3$)	0.054 (0.05) $^{\blacktriangle\blacktriangledown}$	0.060 (0.06) $^{\blacktriangle\blacktriangledown}$	0.117 (0.09) $^{\blacktriangle\triangledown}$
PM($n = 10^4$)	0.046 (0.05) $^{\blacktriangle}$-	0.054 (0.05) $^{\blacktriangle\blacktriangledown}$	0.090 (0.08) $^{\blacktriangle}$-
PM($n = 10^5$)	0.046 (0.05) $^{\blacktriangle}$-	0.039 (0.05) $^{\blacktriangle}$-	0.087 (0.08) $^{\blacktriangle}$-

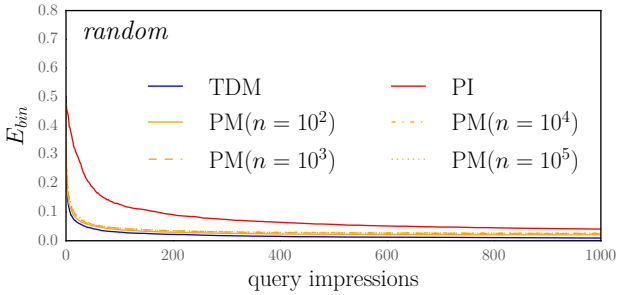

Figure 2: The error is plotted against the number of queries. The error was evaluated by comparing to a ground truth of *no preferences* (i.e., $P_{ij} = 0.5$ for all i, j). Clicks are generated by a *random* instantiation of CCM [3].

lower error than PI and when there are enough samples ($n \geq 100$) statistically significantly so. However, when the number of samples is too small, PM is outperformed significantly by TDM. When the number of samples increases sufficiently, PM is on par with TDM in terms of sensitivity. Interestingly, when noise increases, performance of PM decreases less compared to TDM.

In terms of bias, we see in Figure 2 that PM is on par with TDM. Both methods only need about 100 impressions from a random user to conclude that no preferences between rankers can be inferred. Again naturally, PI needs much more query impressions to draw the same conclusion because it needs to compare all pairs of rankers. PM is as unbiased as TDM, irrespective of the number of samples.

Lastly, we investigate what happens when the number of rankers $|\mathcal{R}|$ that are being compared increases from the five rankers used until now. We test this with $|\mathcal{R}| = 20$ and find that after 500 navigational query impressions for PI, $E_{bin} = 0.56$, for TDM $E_{bin} = 0.15$, and for PM($n = 10^4$) we find $E_{bin} = 0.13$. The advantage of multileaving over interleaving is clearly shown by these numbers. But moreover, PM clearly outperforms TDM when the number of rankers increases. We confirm a finding from Schuth et al. [18] who showed this to be an inherent disadvantage of TDM as it needs to represent al rankers with teams in the multileaving. PM, because it marginalizes over all possible team assignments, does not have this drawback and still performs well when the number of rankers goes up.

5. CONCLUSION

We have introduced a new method for online ranker evaluation called *probabilistic multileave* (PM). PM extends *probabilistic interleave* (PI) such that it can compare more than two rankers at once, while keeping PI's characteristic of being able to reuse historical interaction data. We empirically compared PM to PI as well as an earlier multileaving method called *team draft multileave* (TDM). The new method infers preferences between rankers by marginalizing over a *sample* of possible team assignments. We use a sample of controlled size to keep the computation tractable and show experimentally that given a large enough sample, our method is both as sensitive and as unbiased as TDM and more so than PI. That is, PM is capable of quickly finding meaningful differences between rankers and it does not infer preferences where it should not.

An important implication of this work is that historical interactions with multileaved comparisons can be reused, allowing for ranker comparisons that need much less user interaction data. Furthermore, we show that our method, as opposed to earlier sensitive multileaving methods, scales well when the number of rankers increases.

Acknowledgements. This research was partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), 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, Amsterdam Data Science, the Dutch national program COMMIT, the ESF Research Network Program ELIAS, the Elite Network Shifts project funded by the Royal Dutch Academy of Sciences (KNAW), the Netherlands eScience Center under project number 027.012.105, the Yahoo! Faculty Research and Engagement Program, the Microsoft Research PhD program, and the HPC Fund.

REFERENCES

[1] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue. Large-scale validation and analysis of interleaved search evaluation. *ACM Trans. Inf. Syst.*, 30(1), 2012.
[2] C. W. Cleverdon, J. Mills, and M. Keen. Factors determining the performance of indexing systems. Aslib cranfield project, Cranfield: College of Aeronautics, 1966.
[3] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM '09*. ACM, 2009.
[4] J. He, C. Zhai, and X. Li. Evaluation of methods for relative comparison of retrieval systems based on clickthroughs. In *CIKM '09*. ACM, 2009.
[5] K. Hofmann. *Fast and Reliably Online Learning to Rank for Information Retrieval*. PhD thesis, University of Amsterdam, 2013.
[6] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM '11*. ACM, 2011.
[7] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing historical interaction data for faster online learning to rank for IR. In *WSDM '13*. ACM, 2013.
[8] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
[9] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02*. ACM, 2002.
[10] T. Joachims. Evaluating retrieval performance using clickthrough data. In *Text Mining*. Physica/Springer, 2003.
[11] T. Joachims, L. A. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in Web search. *ACM Trans. Inf. Syst.*, 25(2), 2007.
[12] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 18(1):140–181, 2009.
[13] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR '07*, 2007.
[14] F. Radlinski and N. Craswell. Optimized interleaving for online retrieval evaluation. In *WSDM '13*. ACM, 2013.
[15] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM '08*. ACM, 2008.
[16] M. Sanderson. Test collection based evaluation of information retrieval systems. *Found. & Tr. Inform. Retr.*, 4(4):247–375, 2010.
[17] A. Schuth, K. Hofmann, S. Whiteson, and M. de Rijke. Lerot: An online learning to rank framework. In *LivingLab '13*. ACM, 2013.
[18] A. Schuth, F. Sietsma, S. Whiteson, D. Lefortier, and M. de Rijke. Multileaved comparisons for fast online evaluation. In *CIKM '14*, pages 71–80. ACM, 2014.
[19] A. Schuth, K. Hofmann, and F. Radlinski. Predicting search satisfaction metrics with interleaved comparisons. In *SIGIR'15*. ACM, 2015.
[20] E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005.