

Optimizing Base Rankers Using Clicks

A Case Study using BM25

Anne Schuth, Floor Sietsma, Shimon Whiteson, and Maarten de Rijke

ISLA, University of Amsterdam, The Netherlands
{anne.schuth, f.sietsma, s.a.whiteson, derijke}@uva.nl

Abstract. We study the problem of optimizing an individual base ranker using clicks. Surprisingly, while there has been considerable attention for using clicks to optimize linear combinations of base rankers, the problem of optimizing an individual base ranker using clicks has been ignored. The problem is different from the problem of optimizing linear combinations of base rankers as the scoring function of a base ranker may be highly non-linear. For the sake of concreteness, we focus on the optimization of a specific base ranker, viz. BM25. We start by showing that significant improvements in performance can be obtained when optimizing the parameters of BM25 for individual datasets. We also show that it is possible to optimize these parameters from clicks, i.e., without the use of manually annotated data, reaching or even beating manually tuned parameters.

1 Introduction

Traditional approaches to evaluating or optimizing rankers are based on editorial data, i.e., manually created explicit judgments. Recent years have witnessed a range of alternative approaches for the purpose of evaluating or optimizing rankers, which reduce or even avoid the use of explicit manual judgments. One type of approach is based on pseudo test collections, where judgments about query-document pairs are automatically generated by repurposing naturally occurring labels such as hashtags or anchor texts [1–3].

Another type of approach is based on the use of implicit signals. The use of implicit signals such as click data to evaluate or optimize retrieval systems has long been a promising alternative or complement to explicit judgments [4, 12–14, 18]. Evaluation methods that interpret clicks as absolute relevance judgments have often been found unreliable [18]. In some applications, e.g., for optimizing the click-through rate in ad placement and web search, it is possible to learn effectively from click data, using various learning to rank methods, often based on bandit algorithms. Click models can effectively leverage click data to allow more accurate evaluations with relatively little editorial data. Moreover, interleaved comparison methods have been developed that use clicks not to infer absolute judgments but to compare rankers by observing clicks on interleaved result lists [7].

The vast majority of work on click-based evaluation or optimization has focused on optimizing a linear combination of base rankers, thereby treating those rankers as black boxes [9, 10, 26]. In this paper, we try to break open those black boxes and examine whether online learning to rank can be leveraged to optimize the base rankers themselves. Surprisingly, even though a lot of work has been done on improving the weights of base rankers in a combined learner, there is no previous work on online learning of the parameters of base rankers and there is a lot of potential gain from this new form of

optimization. We investigate whether individual base rankers can be optimized using clicks. This question has two key dimensions. First, we aim to use clicks, an implicit signal, instead of explicit judgments. The topic of optimizing individual base rankers such as TF.IDF, BM25 or DFR has received considerable attention over the years but that work has almost exclusively used explicit judgments. Second, we work in an online setting while previous work on optimizing base rankers has almost exclusively focused on a more or less traditional, TREC-style, offline setting.

Importantly, the problem of optimizing base rankers is not the limiting case of the problem of optimizing a linear combination of base rankers where one has just one base ranker. Unlike the scoring function that represents a typical online learning to rank solution, the scoring function for a single base ranker is not necessarily linear. A clear example is provided by the well-known BM25 ranker [19], which has three parameters that are related in a non-linear manner: k_1 , k_3 and b .

In this paper, we pursue the problem of optimizing a base ranker using clicks by focusing on BM25. Currently, it is common practice to choose the parameters of BM25 according to manually tuned values reported in the literature, or to manually tune them for a specific setting based on domain knowledge or a sweep over a number of possible combinations using guidance from an annotated data set [5, 25]. We propose an alternative by learning the parameters from click data. Our goal is not necessarily to improve performance over manually tuned parameter settings, but rather to obviate the need for manual tuning.

Specifically, the research questions we aim to answer are as follows.

RQ1 How good are the manually tuned parameter values of BM25 that are currently used? Are they optimal for all data sets on average? Are they optimal for individual data sets?

RQ2 Is it possible to learn good values of the BM25 parameters from clicks? Can we approximate or even improve the performance of BM25 achieved with manually tuned parameters?

Our contributions are (1) the insight that we can potentially achieve significant improvements of state-of-the-art learning to rank approaches by learning the parameters of base rankers, as opposed to treating them as black boxes which is currently the common practice; (2) a demonstration of how parameters of an individual base ranker such as BM25 can be learned from clicks using the dueling bandit gradient descent approach; and, furthermore, (3) insight into the parameter space of a base ranker such as BM25.

2 Related work

Related work comes in two main flavors: (1) work on ranker evaluation or optimization that does not use traditional manually created judgments, and (2) specific work on optimizing BM25.

Several attempts have been made to either simulate human queries or generate relevance judgments without the need of human assessors for a range of tasks. One recurring idea is that of pseudo test collections, which consist of automatically generated sets of queries and for every query an automatically generated set of relevant documents (given some document collection). The issue of creating and using pseudo test collections goes back at least to [24]. Azzopardi et al. [2] simulate queries for known-item search

and investigate term-weighting methods for query generation. Asadi et al. [1] describe a method for generating pseudo test collections for training learning to rank methods for web retrieval; they use anchor text in web documents as a source for sampling queries, and the documents that these anchors link to are regarded as relevant documents for the anchor text (query). Berendsen et al. [3] use a similar methodology for optimizing microblog rankers and build on the idea that tweets with a hashtag are relevant to a topic covered by the hashtag and hence to a suitable query derived from the hashtag.

While these methods use automatically generated labels instead of human annotations, the setting is still an offline setting; learning takes place after collecting a batch of pseudo relevant documents for a set of queries. Clicks have been used in both offline and online settings for evaluation and optimization purposes, with uses ranging from pseudo-relevance feedback [14] to learning to rank or re-rank [12, 13]. Radlinski et al. [18] found that evaluation methods that interpret clicks as absolute relevance judgments are unreliable. Using bandit algorithms, where rankers are the *arms* that can be *pulled* to observe a click as feedback, it is possible to learn effectively from click data for optimizing the click-through rate in ad placement and web search [17]. Carterette and Jones [4] find that click models can effectively leverage click data to allow more accurate evaluations with relatively little editorial data. In this paper, we use probabilistic interleave [7], an interleaved comparison method that uses clicks not to infer absolute judgments but to compare base rankers by observing clicks on interleaved result lists; we use this relative feedback not only to optimize a linear combination of base rankers, as has been done before, but also to optimize an individual ranker. Our optimization method uses this relative feedback in a dueling bandit algorithm, where *pairs of rankers* are the arms that can be pulled to observe a click as relative feedback [9, 10, 26].

Our case study into optimizing an individual base ranker using clicks focuses on BM25; a parameterized (with parameters k_1 , k_3 and b) combination of term frequency (TF), inverse document frequency (IDF) and query term frequency (cf. Section 3.1). A good general introduction to this ranker can be found in [20], while detailed coverage of early experiments aimed at understanding the model’s parameters can be found in [22]. Improvements to standard BM25 have previously been investigated by Svore and Burges [23], who apply BM25 to different document fields. Then a machine learning approach is used to combine the results on these different fields. However, there the parameters of BM25 are still set at a fixed value. Most similar to the work presented here is [5, 25]. There, however, the parameters of BM25 are optimized based on relevance labels, not clicks, in an offline learning, so that the parameters learned cannot be adapted while search takes place. Interestingly, over the years, different values of the key parameters in BM25 are used as manually tuned “default” values; e.g., Qin et al. [16] use $k_1 = 2.5$, $k_3 = 0$, $b = 0.8$ for the .gov collection. They use $k_1 = 1.2$, $k_3 = 7$, $b = 0.75$ for the OHSUMED collection, while Robertson and Walker [19] use $k_1 = 2.0$, $b = 0.75$.

3 Method

Today’s state-of-the-art ranking models combine the scores produced by many base rankers and compute a combination of them to arrive at a high-quality ranking. In the

simplest form, this combination can be a weighted sum:

$$s(q, d) = w_1 \cdot s_1(q, d) + \dots + w_n \cdot s_n(q, d), \quad (1)$$

where w_i is the weight of each base ranker $s_i(q, d)$ that operates on the query q and document d . The base rankers may have internal parameters that influence their performance. We focus on one particular base ranker, BM25, which has three parameters that determine the weight applied to term frequency, inverse document frequency and other query or document properties in the BM25 scoring function.

Below, we first recall BM25 in full detail and then describe how we use clicks to optimize BM25’s parameters.

3.1 Implementation of BM25 Several variants of BM25 are used in the literature. We use the variant that is used to compute the BM25 feature in the LETOR data set [16]. Given a query q and document d , the BM25 score is computed as a sum of scores for every term q_i in the query that occurs at least once in d :

$$BM25(q, d) = \sum_{q_i: tf(q_i, d) > 0} \frac{idf(q_i) \cdot tf(q_i, d) \cdot (k_1 + 1)}{tf(q_i, d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avgdl})} \cdot \frac{(k_3 + 1) \cdot qtf(q_i, q)}{k_3 + qtf(q_i, q)} \quad (2)$$

The terms used in this formula are:

- $idf(q_i)$ (*inverse document frequency*): computed as $idf(q_i) := \log\left(\frac{N - df(q_i) + 0.5}{df(q_i) + 0.5}\right)$ where N is the total number of documents in the collection and $df(q_i)$ is the number of documents in which the term q_i occurs at least once;
- $tf(q_i, d)$ (*term frequency*): the number of times term q_i occurs in document d ;
- $qtf(q_i, q)$ (*query term frequency*): the number of times term q_i occurs in query q ;
- $\frac{|d|}{avgdl}$: the length of document d , normalized by the average length of documents in the collection;
- k_1 , b and k_3 : the parameters of BM25 that we want to optimize. Usually, k_1 is set to a value between 1 and 3, b is set somewhere around 0.8 and k_3 is set to 0. Note that when k_3 is set to 0 the entire right part of the product in Eq. 2 cancels out to 1.

3.2 Learning from clicks Most learning to rank approaches learn from explicit, manually produced relevance assessments [15]. These assessments are expensive to obtain and usually produced in an artificial setting. More importantly, it is not always feasible to obtain the assessments needed. For instance, if we want to adapt a ranker towards a specific user or a group of users, we cannot ask explicit feedback from these users as it would put an undesirable burden upon these users.

Instead, we optimize rankers using clicks. It has been shown by Radlinski et al. [18] that interpreting clicks as absolute relevance judgments is unreliable. Therefore, we use a dueling bandit approach: the candidate preselection (CPS) method. This method was shown to be state-of-the-art by Hofmann et al. [9]. It is an extension of the dueling bandit gradient descent (DBGD) method, proposed in [26]. Briefly, DBGD works as follows. The parameters that are being optimized are initialized. When a query is presented to the learning system, two rankings are generated: one with the parameters set at the current best values, another with a perturbation of these parameters. These two rankings are interleaved using probabilistic interleave [7, 8], which allows for the

reuse of historical interactions. The interleaved list is presented to the user and we observe the clicks that the user produces, which are then used to determine which of the two generated rankings was best. If the ranking produced with the perturbed set of parameters wins the interleaved comparison, then the current best parameters are adapted in the direction of the perturbation. CPS is a variant of DBGD that produces several candidate perturbations and compares these on historical click data to decide on the most promising candidate. Only the ranking produced with the most promising perturbation is then actually interleaved with the ranking generated with the current best parameters and exposed to the user.

The difference between the current best ranker and the perturbed ranker is controlled by the parameter δ . The amount of adaptation of the current best ranker in case the perturbed ranker wins is controlled by a second parameter, α . Together, these parameters balance the speed and the precision with which the algorithm learns. If they are too big, the learning algorithm may oscillate, skip over optimal values and never converge to the optimum. If they are too small, the learning algorithm will not find the optimum in a reasonable amount of time.

We aim to learn the BM25 parameters k_1 , b and k_3 from clicks, using the learning method described above. Because the parameters are of very different orders of magnitude, with b typically ranging between 0.45 and 0.9 and k_1 typically ranging between 2 and 25, we chose to use a separate δ and α for each parameter. This is necessary because what may be a reasonable step size for k_1 will be far too large for b . Therefore we have, for example, a separate δ_{k_1} and δ_b . This allows us to govern the size of exploration and updates in each direction.

4 Experimental setup

We investigate whether we can optimize the parameters of a base ranker, BM25, from clicks produced by users interacting with a search engine. Below, we first describe the data we use to address this question. Then we describe how our click-streams are generated, and our evaluation setup.¹

4.1 Data For all our experiments we use features extracted from the .gov collection that is also included in the LETOR data set [16]. The .gov collection consists of a crawl of the .gov domain with about 1M documents. The six sets of queries and relevance assessments we use are based on TREC Web track tasks from 2003 and 2004. The data sets 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 contain between 50 and 150 queries and approximately 1,000 judged documents per query.

We index the original .gov collection to extract low-level features such as term frequency and inverse document frequency that are needed for BM25. While indexing, we do not perform any pre-processing (e.g., no stemming, no stop word removal). We only extract features for the documents in the LETOR data set [16]. All the data sets we use are split by query for 5-fold cross validation.

¹ All our code is open source, available at <https://bitbucket.org/ilps/lerot> [21].

Table 1. Instantiations of the DCM click model used in our experiments, following [9].

	$P(\text{click} = 1 R)$		$P(\text{stop} = 1 R)$	
	$r = 0$	$r = 1$	$r = 0$	$r = 1$
<i>perfect</i>	0.0	1.0	0.0	0.0
<i>navigational</i>	0.05	0.95	0.2	0.9
<i>informational</i>	0.4	0.9	0.1	0.5
<i>almost random</i>	0.4	0.6	0.5	0.5

4.2 Clicks We employ a click simulation framework analogous to that of [9]. We do so because we do not have access to a live search engine or a suitable click log. Note that, even if a click log was available, it would not be adequate since the learning algorithm is likely to produce result lists that never appear in the log.

In our click simulation framework, we randomly sample with replacement a query from the queries in the data set. The learning system is then responsible for producing a ranking of documents for this query. This ranking is then presented to a simulated user, which produces clicks on documents that can be used by the learning system to improve the ranker. In our experiments, we use the Dependent Click Model (DCM) by Guo et al. [6] to produce clicks. This click model assumes that users scan a result list from top to bottom. For each examined document, users are assumed to determine whether it potentially satisfies their information need enough to click the document. This decision is based on relevance labels manually given to each document-query combination. We model this as $P(C|R)$: the probability of clicking a document given its relevance label. After clicking, the user might continue scanning down the document list or stop, which is modeled as $P(S|R)$: the probability of stopping after a document given its relevance label. Again following [9], we instantiate $P(C|R)$ and $P(S|R)$ as in Table 1. We use four instantiations of the click model: the *perfect* click model, in which exactly every relevant document is clicked; the *navigational* click model, in which users almost only click relevant documents and usually stop when they have found a relevant document; the *informational* click model, in which non-relevant documents are also clicked quite often and users stop after finding a relevant document only half of the time; and the *almost random* click model in which there is only a very small difference in user behavior for relevant and non-relevant documents.

4.3 Learning setup We employ the learning approach described in Section 3.2. For CPS we use the parameters suggested by [9]: we use $\eta = 6$ candidate perturbations and we use the $\lambda = 10$ most recent queries. We initialize the weights of the BM25 model randomly. For the learning of BM25 parameters, we set $\alpha_b = 0.05$ and $\delta_b = 0.5$. We computed the average ratio between k_1 and b across the parameter values that were optimal for the different data sets, and set α_{k_1} and δ_{k_1} accordingly. This ratio was 1 to 13.3, so we set $\alpha_{k_1} = 0.665$ and $\delta_{k_1} = 6.65$. These learning parameters have been tuned on a held out development set.

4.4 Evaluation and significance testing As evaluation metric, we use nDCG [11] on the top 10 results, measured on the test sets, following, for instance, [23, 25]. For each learning experiment, for each data set, we run the experiment for 2,000 interactions with the click model. We repeat each experiment 25 times and average results over the 5

Table 2. NDCG scores for various values of BM25 parameters k_1 and b , optimized for different data sets. The first and last row give the scores with parameters that have been manually tuned for the .gov and OHSUMED collections, respectively [16]. Other parameter values are chosen to produce maximal scores, printed in boldface, for the different data sets listed in the first column. For all results, $k_3 = 0$. Statistically significant improvements (losses) over the values manually tuned for .gov are indicated by Δ ($p < 0.05$) and \blacktriangle ($p < 0.01$) (∇ and \blacktriangledown).

	k_1	b	HP2003	HP2004	NP2003	NP2004	TD2003	TD2004	Overall
.gov	2.50	0.80	0.674	0.629	0.693	0.599	0.404	0.469	0.613
HP2003	7.40	0.80	0.692	0.650	0.661 \blacktriangledown	0.591	0.423 \blacktriangle	0.477	0.614
HP2004	7.30	0.85	0.688	0.672 Δ	0.657 \blacktriangledown	0.575	0.423 \blacktriangle	0.482 Δ	0.613
	2.50	0.85	0.671	0.613	0.682	0.579 ∇	0.404	0.473	0.605 ∇
	7.30	0.80	0.690	0.647	0.661 \blacktriangledown	0.592	0.423 \blacktriangle	0.477	0.613
NP2003	2.60	0.45	0.661	0.572 ∇	0.719	0.635	0.374 \blacktriangledown	0.441 \blacktriangledown	0.607
	2.50	0.45	0.660	0.572 ∇	0.718	0.635	0.374 \blacktriangledown	0.441 \blacktriangledown	0.607
	2.60	0.80	0.675	0.629	0.692	0.601	0.403	0.470	0.613
NP2004	4.00	0.50	0.663	0.584	0.705	0.647 Δ	0.386 ∇	0.446 \blacktriangledown	0.609
	2.50	0.50	0.663	0.573 ∇	0.713	0.635	0.381 \blacktriangledown	0.444 \blacktriangledown	0.607
	4.00	0.80	0.680	0.645	0.683	0.605	0.414 Δ	0.474	0.616
TD2003	25.90	0.90	0.660	0.597	0.515 \blacktriangledown	0.478 \blacktriangledown	0.456 Δ	0.489 Δ	0.550 \blacktriangledown
	2.50	0.90	0.676	0.607	0.672	0.560 \blacktriangledown	0.405	0.471	0.600 \blacktriangledown
	25.90	0.80	0.645	0.576	0.535 \blacktriangledown	0.493 \blacktriangledown	0.445	0.482	0.549 \blacktriangledown
TD2004	24.00	0.90	0.664	0.604	0.520 \blacktriangledown	0.481 \blacktriangledown	0.449 Δ	0.491 Δ	0.553 \blacktriangledown
	2.50	0.90	0.676	0.607	0.672	0.560 \blacktriangledown	0.405	0.471	0.600 \blacktriangledown
	24.00	0.80	0.645	0.578	0.538 \blacktriangledown	0.496 \blacktriangledown	0.446	0.482	0.550 \blacktriangledown
OHSUMED	1.20	0.75	0.662 ∇	0.589 \blacktriangledown	0.703	0.591	0.398	0.461 \blacktriangledown	0.605 ∇

folds and these repetitions. We test for significant differences using the paired t-test in answering RQ1 and the independent measures t-test for RQ2.

5 Results and analysis

We address our research questions in the following two subsections.

5.1 Measuring the performance of BM25 with manually tuned parameters In order to answer RQ1, we compute the performance of BM25 with the parameters used in the LETOR data set [16]. The parameter values used there differ between the two document collections in the data set. The values that were chosen for the .gov collection were $k_1 = 2.5$, $b = 0.8$ and $k_3 = 0$. The values that were chosen for the OHSUMED collection were $k_1 = 1.2$, $b = 0.75$ and $k_3 = 7$. We refer to these values as the manually tuned .gov or OHSUMED parameter values, respectively. Note that the manually tuned .gov parameter values were tuned to perform well on average, over all data sets.

The results of running BM25 with the .gov manual parameters (as described in Section 4) are in the first row of Table 2. We also experiment with different values of k_1 , b and k_3 . We first tried a range of values for k_1 and b . For k_1 the range is from -1 to 30

with steps of 0.1 and for b the range is for -0.5 to 1 with steps of 0.05 . The results are in Table 2. For each of the data sets, we include the parameter values that gave maximal nDCG scores (in bold face). For each value of k_1 and b , we show the performance on each data set and the average performance over all data sets.

The results show that when we average over all data sets, no significant improvements to the manually tuned .gov parameter values can be found. This is to be expected, and merely shows that the manual tuning was done well. However, for four out of six data sets, a significant improvement can be achieved by deviating from the manually tuned .gov parameter values *for that particular dataset*. Furthermore, taking the average optimal nDCG, weighted with the number of queries in each data set, yields an overall performance of 0.644 nDCG. Thus, it pays to optimize parameters for specific data sets.

In cases where both k_1 and b were different from the manually tuned .gov values, we also consider the results of combining k_1 with the manually tuned .gov value for b and vice-versa. E.g., when $k_1 = 4.0$ and $b = 0.5$, for the NP2004 data set the value of b has a bigger impact than the value of k_1 : changing k_1 back to the manually tuned value causes a decrease of nDCG of 0.012 points, while changing b to the manually tuned value gives a decrease of 0.042 points. However, in other cases the value of k_1 seems to be more important. E.g., for the TD2003 data set we can achieve an improvement of 0.041 points by changing k_1 to 25.9 , while keeping b at the manually tuned 0.8 .

The bottom row in Table 2 shows the results of a BM25 ranker with the manually tuned OHSUMED parameter values. This ranker performs worse than the manually tuned .gov values averaged over all data sets, which, again, shows that it makes sense to tune these parameters, rather than just taking a standard value from the literature.

For the third parameter k_3 , we performed similar experiments, varying the parameter value from 0 to $1,000$. There were no significant differences in the resulting nDCG scores. The small differences that did occur favored the manually tuned value 0 . The fact that k_3 hardly has any influence on the performance is to be expected, since k_3 weights the query term frequency (cf. Equation 2), the number of times a word appears in the query. For most query terms, the query term frequency is 1 . Since the weight of this feature does not greatly affect the result, we omit k_3 from the rest of our analysis.

Using these results, we are ready to answer our first research question, RQ1. The manually tuned .gov values for k_1 and b are quite good when we look at a combination of all data sets. When looking at different data sets separately, significant improvements can be reached by deviating from the values that were manually tuned for the entire collection. This shows that tuning of the parameters to a specific setting is a promising idea. Considering the last parameter k_3 , the standard value was optimal for the data sets we investigated.

5.2 Learning parameters of BM25 using clicks In this section we answer RQ2: can we learn the parameters of BM25 from clicks? We aim to learn the parameters per data set from clicks. Our primary goal is not to beat the performance of the manually tuned .gov parameters. Should optimizing a base ranker such as BM25 prove successful, i.e., reach or even beat manually tuned values, the advantage is rather that optimizing the parameters no longer requires human annotations. Furthermore, learning the parameters eliminates the need for domain-specific knowledge, which is not always available, or sweeps over possible parameter values, which cost time and cannot be done online.

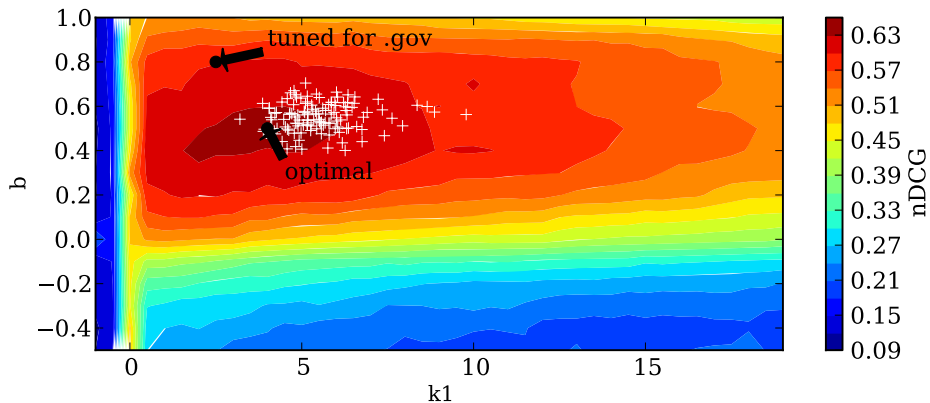


Fig. 1. Optimization landscape for two parameters of BM25, k_1 and b , for the NP2004 data set measured with nDCG. White crosses indicate where individual runs of the learning algorithm plateaued when learning from clicks produced with the *perfect* click model. For the other five data sets we experimented with we obtained a similar landscape with the peak on a different location.

To begin, we visualize the optimization landscape for the two BM25 parameters that matter: k_1 and b . We use the data obtained from the parameter sweep described in Section 5.1. The optimization landscape is unimodal and generally smooth when averaged over many queries, as illustrated by Fig. 1 for the NP2004 data set. We find similar landscapes with peaks at different locations (listed in Table 2) for other data sets. This observation suggests that a gradient descent approach such as DBGD is a suitable learning algorithm. Note, however, that the online learning algorithm will never actually observe this landscape: it can only observe *relative* feedback from the interleaved comparisons and moreover, this feedback is observed on a *per query* basis.

Next, we optimize k_1 and b for each individual data set using the four instantiations of our click model: *perfect*, *navigational*, *informational*, and *almost random*. The learning curves are depicted in Fig. 2. Irrespective of the noise in the feedback, and of the (random) starting point, the learning method is able to dramatically improve the performance of BM25. For the *perfect* click model the final performance after 2000 queries is either on a par with the manually tuned values used by Qin et al. [16] or above. We can, however, not always recover the gain we observe in the parameter sweep in Table 2 completely when learning from clicks.

For the NP2004 data set we have plotted the final parameter values that have been learned using the *perfect* click model in Fig. 1. The final parameter values are clustered near the optimal value, indicating that the learning method is indeed capable of finding the peak of the landscape. Final parameters for individual runs using each data set are depicted in Fig. 3. For each data set, the parameters converge to a different region. We also see that the manually tuned parameters are not included in any of these regions.

Performance generally degrades when clicks become less reliable. However, the performance of the navigational click model is not much lower than the performance of the perfect click model. This is a promising result, since feedback from actual users will be noisy and our learning method should be able to deal with that.

The above experiments are all initialized with random starting parameters. In cases where a good starting point is known, learning can be sped up. For example, we also

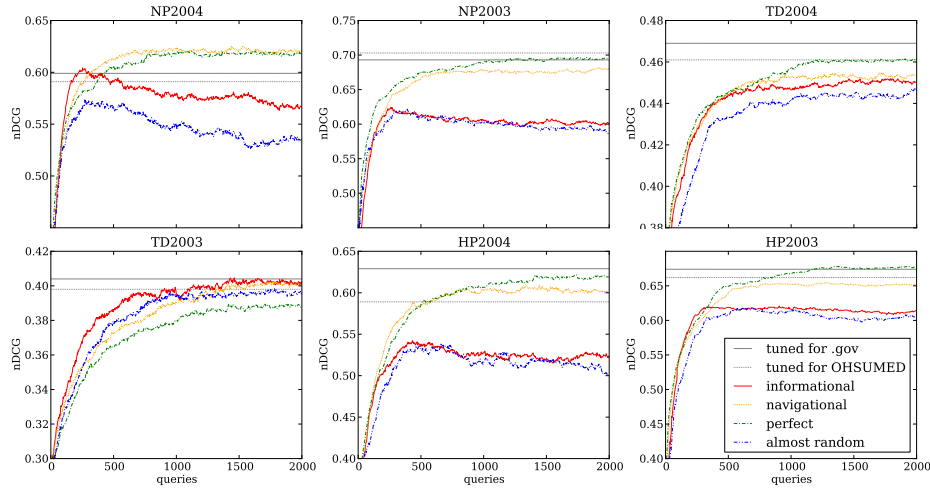


Fig. 2. Learning curves when learning the parameters of BM25 using DBGD from clicks. Measured in nDCG on a holdout data set averaged over 5-fold cross validation and 25 repetitions. The clicks used for learning are produced by the *perfect*, *navigational*, *informational* and *almost random* click model. The horizontal gray lines indicate the performance for the manually tuned *.gov* (solid) and OHSUMED (dotted) parameters.

initialized learning with the manually tuned *.gov* parameters ($k_1 = 2.5$ and $b = 0.8$) and found a plateau that was not different from the one found with random initialization. It was, however, found in fewer than 200 queries, depending on the data set.

In conclusion, we can give a positive answer to the first part of [RQ2](#). Learning good values for the BM25 parameters from user clicks is possible. As to the second part of [RQ2](#), the optimized parameters learned from clicks lead to performance of BM25 that approaches, equals or even surpasses the performance achieved using manually tuned parameters for all datasets.

6 Conclusion

In this paper we investigated the effectiveness of using clicks to optimize base rankers in an online learning to rank setting. State-of-the-art learning to rank approaches use a linear combination of several base rankers to compute an optimal ranking. Rather than learning the optimal weights for this combination, we optimize the internal parameters of these base rankers. We focussed on the base ranker BM25 and aimed at learning these parameters of BM25 in an online setting using clicks.

Our results show that learning good parameters of BM25 from clicks is indeed possible. As a consequence, it is not necessary to hand tune these parameters or use human assessors to obtain labeled data. Learning with a dueling bandit gradient descent approach converges to near-optimal parameters after training with relative click feedback on about 1,000 queries. Furthermore, the performance of BM25 with these learned parameters approaches, equals or even surpasses the performance achieved using manually tuned parameters for all datasets. The advantage of our approach lies in the fact that the parameters can be learned automatically from implicit feedback, and that the parameters can be tuned specifically for different settings. The parameters learned

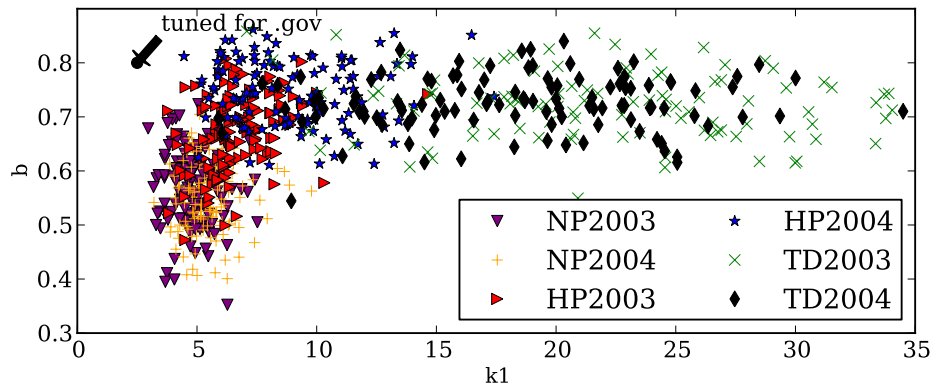


Fig. 3. Parameter settings where individual runs of the learning algorithm plateaued when learning with the *perfect* click model for 2,000 queries.

from the data in the different data sets differ greatly from each other. More importantly, the performance gains we observed by tuning the parameters for a specific data set were significant for many of the data sets considered. We could, however, not always recover this gain completely when learning from clicks.

In state-of-the-art learning to rank algorithms applied to, e.g., the LETOR data set, the BM25 score is generally used in a weighted combination with many other base rankers. For future work, it is interesting to see how the click-optimized versions of BM25 (or other base rankers, for that matter) can improve the performance of a state-of-the-art learning to rank algorithm when BM25 is used as one query-document feature among many. There are several ways in which the process of optimizing a base ranker can be integrated with state-of-the-art online learning to rank: (1) first learn the optimal parameters of the base rankers in isolation, then use the optimized base rankers and learn the optimal weights of a linear combination of these base rankers; (2) first learn the optimal weights of a linear combination of base rankers using either optimized or default parameter values for the individual base rankers and then learn the optimal parameters of the individual base rankers based on user clicks in reply to the outcome of the ensemble learning-to-rank algorithm; (3) learn the parameters of the base rankers and the weights of the ensemble together based on user clicks in reply to the outcome of the ensemble.

Acknowledgments. This research was supported by the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 288024 (LiMoSINe project), the Netherlands Organisation for Scientific Research (NWO) under project nrs 640.004.802, 727.011.-005, 612.001.116, HOR-11-10, 612.066.930, the Center for Creation, Content and Technology, the QuaMerdes project funded by the CLARIN-nl program, the TROVe project funded by the CLARIAH program, the Dutch national program COMMIT, the ESF Research Network Program ELIAS, the Elite Network Shifts project funded by the Royal Dutch Academy of Sciences, the Netherlands eScience Center under project nr 027.012.105 and the Yahoo! Faculty Research and Engagement Program.

References

- [1] N. Asadi, D. Metzler, T. Elsayed, and J. Lin. Pseudo test collections for learning web search ranking functions. In *SIGIR ’11*, pages 1073–1082. ACM, 2011.

- [2] L. Azzopardi, M. de Rijke, and K. Balog. Building simulated queries for known-item topics: An analysis using six European languages. In *SIGIR '07*, pages 455–462. ACM, 2007.
- [3] R. Berendsen, M. Tsagkias, W. Weerkamp, and M. de Rijke. Pseudo test collections for training and tuning microblog rankers. In *SIGIR '13*. ACM, 2013.
- [4] B. Carterette and R. Jones. Evaluating search engines by modeling the relationship between relevance and clicks. In *NIPS '07*, pages 217–224. MIT Press, 2008.
- [5] N. Gao, Z.-H. Deng, H. Yu, and J.-J. Jiang. Listopt: Learning to optimize for XML ranking. In *Advances in Knowledge Discovery and Data Mining*, pages 482–492. Springer, 2011.
- [6] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *WSDM '09*. ACM, 2009.
- [7] K. Hofmann, S. Whiteson, and M. de Rijke. A probabilistic method for inferring preferences from clicks. In *CIKM '11*. ACM, 2011.
- [8] K. Hofmann, S. Whiteson, and M. de Rijke. Estimating interleaved comparison outcomes from historical click data. In *CIKM '12*. ACM, 2012.
- [9] 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.
- [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*, 16(1):63–90, 2013.
- [11] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4), 2002.
- [12] S. Ji, K. Zhou, C. Liao, Z. Zheng, G.-R. Xue, O. Chapelle, G. Sun, and H. Zha. Global ranking by exploiting user clicks. In *SIGIR '09*, pages 35–42. ACM, 2009.
- [13] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02*, 2002.
- [14] S. Jung, J. L. Herlocker, and J. Webster. Click data as implicit relevance feedback in web search. *Information Processing & Management*, 43(3):791 – 807, 2007.
- [15] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.
- [16] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.
- [17] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *ICML '08*. ACM, 2008.
- [18] F. Radlinski, M. Kurup, and T. Joachims. How does clickthrough data reflect retrieval quality? In *CIKM '08*, pages 43–52, New York, NY, USA, 2008. ACM.
- [19] S. Robertson and S. Walker. Okapi at TREC-3. In *TREC-3*. NIST, 1995.
- [20] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.
- [21] A. Schuth, K. Hofmann, S. Whiteson, and M. de Rijke. Lerot: An online learning to rank framework. In *LivingLab '13*, pages 23–26. ACM, 2013.
- [22] K. Sparck Jones, S. Walker, and S. Robertson. A probabilistic model of information retrieval: Development and comparative experiments. *Information Processing and Management*, 36: 779–808, 809–840, 2000.
- [23] K. M. Svore and C. J. Burges. A machine learning approach for improved BM25 retrieval. In *CIKM '09*. ACM, 2009.
- [24] J. Tague, M. Nelson, and H. Wu. Problems in the simulation of bibliographic retrieval systems. In *SIGIR '80*, pages 236–255, 1980.
- [25] M. Taylor, H. Zaragoza, N. Craswell, S. Robertson, and C. Burges. Optimisation methods for ranking functions with multiple parameters. In *CIKM '06*. ACM, 2006.
- [26] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML '09*, 2009.