BBM: Bayesian Browsing Model from Petabyte-scale Data

Chao Liu
Microsoft Research
Redmond, WA 98052
chaoliu@microsoft.com

Fan Guo
Carnegie Mellon University
Pittsburgh, PA 15213
fanguo@cs.cmu.edu

Christos Faloutsos
Carnegie Mellon University
Pittsburgh, PA 15213
christos@cs.cmu.edu

ABSTRACT

Given a quarter of petabyte click log data, how can we estimate the relevance of each URL for a given query? In this paper, we propose the Bayesian Browsing Model (BBM), a new modeling technique with following advantages: (a) it does *exact* inference; (b) it is single-pass and parallelizable; (c) it is effective.

We present two sets of experiments to test model effectiveness and efficiency. On the first set of over 50 million search instances of 1.1 million distinct queries, BBM outperforms the state-of-the-art competitor by 29.2% in log-likelihood while being *57 times* faster. On the second click-log set, spanning a *quarter of petabyte* data, we showcase the scalability of BBM: we implemented it on a commercial MapReduce cluster, and it took *only 3 hours* to compute the relevance for 1.15 billion distinct query-URL pairs.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - retrieval models

General Terms

Algorithms, Experimentation, Performance

Keywords

Bayesian models, click log analysis, web search

1. INTRODUCTION

Web search has become indispensable from everyday life: questions ranging from navigating to "live.com" to how to bleach wine stains are all directed to search engines. While responding to users' information needs, search engines also log down the interaction with users, a typical form of which is what URLs are presented and which are clicked in each search result. Such click log constitutes an invaluable source of user feedbacks, which can be leveraged in many search-related applications [4], *e.g.*, query recommendation [5, 31], learning to rank [1, 18, 22, 26], and personalized search [28], just to name a few.

A central question in click log analysis is to infer user-perceived relevance for each query-URL pair. The task could be as trivial as counting the click through rate, should the search engine know the snippets of which URLs are examined by the user in addition to those clicked. As such information is not available in the log, we need to first understand how users examine search results and make decisions on clicks. Thanks to previous work in this direction (*e.g.*, [19, 20, 8]), it is well known that a number of factors must be considered in an accurate interpretation of user clicks, such as the position-bias of examination and clicks, and dependency between clicks over different documents in the same search result page. Quite a few statistical models have been recently developed to leverage user clicks for inferring user-perceived relevance, some representatives of which are the cascade model [8], Dependent Click Model [17], and User Browsing Model [11]. We will review them with details in Section 2.

In practice, it is common to have tens to hundreds of terabytes log data accumulated at the server side every day. This data stream nature of click logs imposes two computational challenges for click modeling. First, the scalability: a click model must scale comfortably to terabyte-scale or even petabyte-scale data. Even better, we would expect the modeling to be highly parallelizable for better scalability. Second, the capability to be incrementally updated, *i.e.*, incremental learning: for not losing sync with the evolving world wide web, a click model (and its relevance estimate thereof) has to be continuously updated. Considering the ever-growing search volume, we would expect a practical click model to be *single-pass*, which minimizes the storage and I/O cost for revisiting historical data.

The previous models, despite of their different user behavior assumptions, all follow the point-estimation philosophy: the estimated relevance for each query-URL pair is a single number normalized to [0,1], and a larger value indicates stronger relevance. While this single-number estimate could suffice for many usages, it nevertheless falls short of capacity for broader applications, for example, it is unspecified how to estimate the probability that URL u_i is preferred to u_j for the same query while their relevance are r_i and r_j respectively. Existing learning-to-rank algorithms derive pairwise preference relationship either from human ratings (*e.g.*, [7]) or from certain heuristics ([18, 24]). To the best of our knowledge, no principled approach has been proposed to compute a *preference probability*, which could support broader applications in both web search and online advertising.

In this paper, we propose the Bayesian Browsing Model (BBM), which builds upon the state-of-the-art User Browsing Model (UBM) and scales to petabyte-scale data with easy incremental updates. By virtue of the Bayesian approach to modeling the document relevance, the preference probability between multiple documents is well-defined and can be computed based on document relevance posteriors. Moreover, we present an exact inference algorithm for BBM that exploits the particular probabilistic dependency in the model, and reveals the relevance posterior in closed form after a single pass over the log data.

In summary, we make the following contributions in this study:

- We propose BBM, together with an exact inference algorithm that derives the relevance posterior in closed form, which facilitates single-pass, incremental computation.
- We compare BBM with a state-of-the-art model in a real world data set of over 50 million search instances for 1.1 million distinct queries. BBM is both effective (29.2% improvement in average log-likelihood) and efficient (57 times faster).
- We implement BBM on a commercial MapReduce cluster, and it takes *only 3 hours* to compute relevance for more than 1.15 billion query-URL pairs by mining a 265 TeraByte click log.

The rest of the paper is organized as follows. Section 2 introduces preliminaries and discusses previous click models. We elaborate on BBM and relevant algorithms in Section 3. Performance comparison results are reported in Section 4, and the petabyte-scale experiment is presented in Section 5. Section 6 covers related work, and the paper is concluded in Section 7.

2. PRELIMINARIES

We first introduce notations in this paper. A web user submits a *query* q to a search engine, gets back top-M (usually $M = 10$) URLs, examines them, and clicks some or none of them. This whole process is called a *search instance*, and any subsequent re-submission or reformulation is regarded as another instance. Because relevance is defined w.r.t. *query-URL pairs*, we hold q fixed in the following discussion unless noted otherwise. We will use *document* and *URL* exchangeably in this paper.

Suppose there are n search instances for a given query, denoted by I_k $(1 \leq k \leq n)$, and a total of N distinct URLs are shown in the n instances, denoted by u_j $(1 \leq j \leq N)$. For each search instance I_k, the document impression is defined by a function ϕ_k such that $\phi_k(i) = j$ if u_j is the document impressed at the ith position (or rank) in I_k. A higher rank for u_j corresponds to a smaller i.

In click models, examination and clicks are treated as probabilistic events. For a particular search instance (k is fixed), we use binary random variables E_i to denote the examination event of the document snippet at position i (documents in bottom positions may not be examined) and C_i for the click event. Therefore, $P(E_i = 1)$ is the examination probability for position i and $P(C_i = 1)$ is the corresponding click probability.

The popular *examination hypothesis* [23] can thus be summarized as follows: for $i = 1, \dots, M$,

$$P(C_i = 1|E_i = 0) = 0, \tag{1}$$
$$P(C_i = 1|E_i = 1) = r_{\phi(i)}, \tag{2}$$

where $r_{\phi(i)}$, defined formally as the *document relevance*, is the conditional probability of click after examination. It helps to disentangle clickthroughs of documents with various ranks as being caused by examination position-bias and relevance. Click models based on the examination hypothesis share this definition but differ in the derivation of $P(E_i)$.

The *cascade hypothesis* in [8] states that users always start the examination from the first document. And the examination is strictly linear to the rank: a document is examined only if all documents in higher ranks are examined.

$$P(E_1 = 1) = 1, \tag{3}$$
$$P(E_{i+1} = 1|E_i = 0) = 0. \tag{4}$$

Under the cascade hypothesis, E_{i+1} is conditionally independent of all examine/click events above i once we know E_i, but E_{i+1} may also depend on the click C_i.

The *cascade model* proposed in [8] puts together previous two hypotheses and further constrains that

$$P(E_{i+1} = 1|E_i = 1) = 1 - C_i, \tag{5}$$

which implies that a user keeps examining the next document until reaching the first click, after which the user abandons the search instance and stops the examination.

The *dependent click model* (DCM) [17] further generalizes the cascade model to instances with multiple clicks by introducing a set of query-independent global parameters as conditional probabilities of examining the next document after a click, *i.e.*, Eq. 5 is replaced by

$$P(E_{i+1} = 1|E_i = 1, C_i = 1) = \lambda_i, \tag{6}$$
$$P(E_{i+1} = 1|E_i = 1, C_i = 0) = 1, \tag{7}$$

where λ_i $(1 \leq i < M)$ are position-dependent *user behavior parameters*.

The *User Browsing Model* (UBM) [11] is also based on the examination hypothesis, but does not follow the cascade hypothesis. Instead, it assumes that the examination probability E_i is determined by the preceding click position $r_i = \arg\max_{l<i}\{C_l = 1\}$ and the distance from the current position to the preceding click $d_i = i - r_i$:

$$P(E_i = 1|C_{1:i-1}) = \beta_{r_i,d_i}. \tag{8}$$

If there is no preceding click, $r_i = 0$. We denote all possible (r, d)'s by the set

$$\mathcal{T} = \{(r, d)|0 \leq r < M, 1 \leq d \leq M - r\}. \tag{9}$$

User behavior parameters in UBM are $\{\beta_{r,d}|(r, d) \in \mathcal{T}\}$, and the size of this set is $M(M + 1)/2$, which is an order of magnitude larger than DCM. The log-likelihood of the search instance under UBM is

$$\log P(C_{1:M}) = \sum_{i=1}^{M}\Big(C_i \log r_{d_i} + C_i \log \beta_{r_i,d_i} + (1 - C_i)\log(1 - r_{d_i}\beta_{r_i,d_i})\Big). \tag{10}$$

We will compute the average log probability of click sequences over test search instances to compare model effectiveness. In our on-going work, we find that UBM is slightly

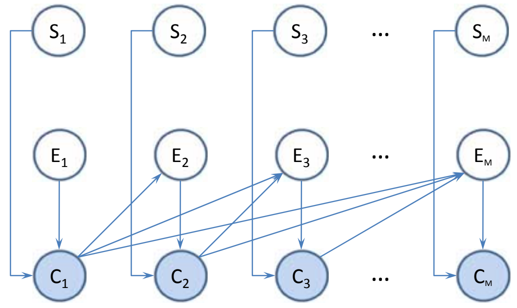


Figure 1: The graphical model representation of BBM. Observed click variables are shaded.

better than DCM by less than 5 percent, but at the expense of a much higher computation cost.

3. BAYESIAN BROWSING MODEL

UBM represents the state-of-the-art of click model design with a sophisticated user behavior model and better performance than DCM. Estimates of the document relevance r_{d_i}'s and global parameter $\beta_{r,d}$'s can be obtained by maximizing the log-likelihood over all search instances across queries. However, because of the coupling between r_{d_i}'s and $\beta_{r,d}$'s in the last term of Eq. 10, iterative coordinate-ascent algorithm has to be employed which requires multiple passes of updates for the relevance of all query-URL pairs until convergence. While still applicable, the iterative algorithm will nevertheless hamper the model scalability when the input log goes beyond terabytes. In addition, the maximum-likelihood estimation still leave the problem of computing preference probabilities for multiple documents unsolved.

We therefore propose BBM, the Bayesian Browsing Model, which builds upon UBM by modeling document relevance as hidden random variables and imposing independent priors on them. This augmentation, although seemingly trivial, achieves several theoretical and practical advantages over its non-Bayesian counterpart:

- **Scalability**: no iteration is needed anymore, and a single pass suffices for computing global parameters and inferring document relevance.
- **Algorithmic Accuracy**: *exact* posterior for document relevance can be derived in closed form, and empirical results demonstrate the model effectiveness with the proposed algorithms.
- **Enhanced Modeling Capacity**: based on posterior *distribution* rather than a single point estimate, we can derive a number of interesting results which include the preference probability (Eq. 27).

In the remainder of this section, we first go over the formal model specification in Section 3.1, then proceed to the exact inference algorithm in Section 3.2 and the maximum-likelihood parameter estimation in Section 3.3. A toy example is given in Section 3.4 to illustrate inference procedures. Finally, we discuss the parallel version of the algorithm in Section 3.5.

3.1 Model Specification

For a given query with n search instances and N distinct URLs. Let $\mathbf{R} = (R_1, R_2, \ldots R_N)$ be the corresponding relevance variables ranging in $[0,1]$. While any independent prior can be applied to $\mathbf{R}$, the *iid* non-informative uniform prior is the most straightforward choice, *i.e.*,

$$p(\mathbf{R}) = \prod_{j=1}^{N} p(R_j) = \prod_{j=1}^{N} \mathbb{I}_{[0,1]}(R_j). \tag{11}$$

We will mention the general case for other priors later, and note that when the priors over different documents are independent, so are their posteriors and in the BBM family. Therefore, BBM is consistent.

Figure 1 plots the graphical model of BBM for a particular search instance. The model consists of three layers of random variables. The top layer $\mathbf{S} = (S_1, S_2, \ldots, S_M)$ are nominal relevance variables such that $S_i = R_{\phi(i)}$. The other two layers $\mathbf{E}$ and $\mathbf{C}$ represent examination and click events respectively. The full model specification that accompanies Fig. 1 is as follows:

$$p(\mathbf{S}) = \prod_{i=1}^{M} p(S_i) = \prod_{i=1}^{M} \mathbb{I}_{[0,1]}(S_i), \tag{12}$$

$$P(E_1 = 1) = \beta_{0,1}, \tag{13}$$

$$P(C_i = 1 | E_i = 0, S_i) = 0, \tag{14}$$

$$P(C_i = 1 | E_i = 1, S_i) = S_i, \tag{15}$$

$$P(E_i = 1 | C_1, \ldots, C_{i-1}) = \beta_{r_i, d_i}, \tag{16}$$

where r_i, d_i are defined in the same way as in Eq. 8.

3.2 Document Relevance Inference

For a general Bayesian network with similar structures as BBM, computation of the posterior over $\mathbf{R}$ given the click data $\mathbf{C}^{1:n}$ is intractable. Here we provide an exact inference procedure for BBM, which not only gives the relevance posterior in closed form, but also reveals how to obtain these posteriors through a single pass of the data. This algorithm exploits the particular dependency structure of BBM, and is therefore preferred to the existing off-the-shelf approximate inference algorithms such as expectation propagation [21] and generalized belief propagation [30].

First, based on the Bayes theorem, we have

$$p(\mathbf{R}|\mathbf{C}^{1:n}) \propto p(\mathbf{R}) \prod_{k=1}^{n} P(\mathbf{C}^k|\mathbf{R}). \tag{17}$$

Our goal is to first obtain an un-normalized posterior distribution over $\mathbf{R}$, and then to normalize it later. Since the prior $p(\mathbf{R})$ is already known, what is needed is $P\left(\mathbf{C}^k \middle| \mathbf{R}\right)$, a quantity computed for each instance I_k. Again, an un-normalized version of $P\left(\mathbf{C}^k \middle| \mathbf{R}\right)$ suffices.

In order to compute $P\left(\mathbf{C}^k \middle| \mathbf{R}\right)$, we notice, from the graphical model in Fig. 1, that C_i^k is independent of S_j^k $(j < i)$ conditioned on C_j^k $(j < i)$. With this conditional independence property,

$$\begin{aligned} & P(C_i^k | C_1^k, \ldots, C_{i-1}^k, \mathbf{S}^k) \\ = \; & P(C_i^k | C_1^k, \ldots, C_{i-1}^k, S_i^k) \\ = \; & \sum_{E_i^k=0}^{1} P(C_i^k | E_i^k, S_i^k) P(E_i^k | C_1^k, \ldots, C_{i-1}^k) \\ = \; & \left(\beta_{r_i^k, d_i^k} S_i\right)^{C_i^k} \left(1 - \beta_{r_i^k, d_i^k} S_i\right)^{1 - C_i^k}. \end{aligned} \tag{18}$$

Because Eq. 18 also holds for $i = 1$ given $r_1 = 0$ and $d_1 = 1$, we can multiply the terms for the M positions together to obtain

$$\begin{aligned} P(\mathbf{C}^k|\mathbf{S}^k) &= \prod_{i=1}^{M} P(C_i^k|C_1^k, \ldots, C_{i-1}^k, \mathbf{S}^k) \\ &= \prod_{i=1}^{M} \left(\beta_{r_i^k, d_i^k} S_i^k\right)^{C_i^k} \left(1 - \beta_{r_i^k, d_i^k} S_i^k\right)^{1-C_i^k} \quad (19) \end{aligned}$$

Using $S_i^k = R_{\phi_k(i)}$, Eq. 19 translates into

$$\begin{aligned} P(\mathbf{C}^k|\mathbf{R}) &= P\big(\mathbf{C}^k|R_{\phi_k(1)}, \ldots, R_{\phi_k(M)}\big) \\ &= \prod_{i=1}^{M} \left(\beta_{r_i^k, d_i^k} R_{\phi_k(i)}\right)^{C_i^k} \left(1 - \beta_{r_i^k, d_i^k} R_{\phi_k(i)}\right)^{1-C_i^k}, \quad (20) \end{aligned}$$

where r_i^k denotes the preceding click position before position i in the instance I_k and $d_i^k = i - r_i^k$.

Plugging Eq. 20 back into Eq. 17 and using the uniform prior over $\mathbf{R}$, the un-normalized posterior over $\mathbf{R}$ is

$$\begin{aligned} p(\mathbf{R}|\mathbf{C}^{1:n}) &\propto \prod_{k=1}^{n}\prod_{i=1}^{M} \left(\beta_{r_i^k, d_i^k} R_{\phi_k(i)}\right)^{C_i^k} \left(1 - \beta_{r_i^k, d_i^k} R_{\phi_k(i)}\right)^{1-C_i^k} \\ &\propto \prod_{k=1}^{n}\prod_{i=1}^{M} R_{\phi_k(i)}{}^{C_i^k} \left(1 - \beta_{r_i^k, d_i^k} R_{\phi_k(i)}\right)^{1-C_i^k}. \quad (21) \end{aligned}$$

This posterior is a product of $2nM$ *linear* factors over $R_1, \ldots, R_N$, and for each R_j there are at most $|\mathcal{T}| + 1 = M(M+1)/2 + 1$ distinct linear factors, namely, R_j itself and $(1 - \beta_{r,d} R_j)$ for each possible $(r, d) \in \mathcal{T}$. Therefore, we can reorganize Eq. 21 as

$$\begin{aligned} p(\mathbf{R}|\mathbf{C}^{1:n}) &\propto \prod_{j=1}^{N} p\big(R_j|\mathbf{C}^{1:n}\big) \\ &\propto \prod_{j=1}^{N} R_j^{N_j} \prod_{(r,d)\in\mathcal{T}} \Big(1 - \beta_{r,d} R_j\Big)^{\widetilde{N}_{j,r,d}} \quad (22) \end{aligned}$$

where the exponents are calculated by

$$N_j = \sum_{k=1}^{n}\sum_{i=1}^{M} \mathbb{I}\Big[(C_i^k = 1) \wedge (\phi_k(i) = j)\Big] \quad (23)$$

$$\widetilde{N}_{j,r,d} = \sum_{k=1}^{n}\sum_{i=1}^{M} \mathbb{I}\Big[(C_i^k = 0) \wedge (\phi_k(i) = j) \wedge (r_i^k = r) \wedge (d_i^k = d)\Big] \quad (24)$$

We note that the posterior as written by Eq. 22 indicates that the joint posterior over $\mathbf{R}$ factorizes over R_j's, so the posterior of R_j's are independent of each other, and we can consequently compute the un-normalized posterior (and normalize it) for each R_j independently.

Specifically, by letting

$$\gamma_{\frac{r(2M-r+1)}{2}+d} = \beta_{r,d} \quad (25)$$

so that $\gamma_1 = \beta_{0,1}, \gamma_2 = \beta_{0,2}, \ldots, \gamma_M = \beta_{0,M}, \gamma_{M+1} = \beta_{1,1}, \ldots, \gamma_{M(M+1)/2} = \beta_{M-1,1}$), $\beta_{r,d}$'s are linearized, and the posterior of R_j is represented by

$$p(R_j|\mathbf{C}^{1:n}) \propto R_j^{e_0} \prod_{i=1}^{M(M+1)/2} (1 - \gamma_i R_j)^{e_i}, \quad (26)$$

Algorithm 1 : LearnBBM($\mathbf{C}$, ϕ, $\mathcal{N}$)

```
Input:  $\mathbf{C} = \{\mathbf{C}^1, \ldots, \mathbf{C}^n\}$: click data
        $\phi = \{\phi_1, \ldots, \phi_n\}$: impression data
Output: $\mathcal{N} = \{N_j, \widetilde{N}_{j,r,d}\}$ for $j = 1, \ldots, N$ and $(r, d) \in \mathcal{T}$:
        all the exponents in the posterior
01: initialize every value in $\mathcal{N}$ to 0;
02: for each search instance $k = 1, \ldots, n$
03:     initialize the preceding click position $r = 0$;
04:     for each position $i = 1, \ldots, M$
05:         set index for current document $j = \phi_k(i)$;
06:         if $C_i^k == 1$
07:             $N_j++$;
08:             update the preceding click position $r = i$;
09:         else
10:             set the distance to the preceding click $d = i - r$;
11:             $\widetilde{N}_{j,r,d}++$;
12:         end
13:     end
14: end
Complexity: Time $O(nM)$, Space $O(NM^2)$.
```

where $e_0 = N_j$ and $e_{\frac{r(2M-r+1)}{2}+d} = \widetilde{N}_{j,r,d}$. We denote the exponent vector $(e_0, e_1, \cdots, e_{M(M+1)/2})$ by $\mathbf{e}$, which fully characterizes the posterior of R_j. Normalizing Eq. 26 can be done through a univariate numerical integration over $R_j \in [0, 1]$. In practice, we find that mid-point interpolation with $B = 100$ uniformly distributed points is sufficient. Once normalized, both the density function $p(R_j)$ and the cumulative distribution function $\Phi(R_j)$ can be evaluated with the desired precision.

The numerical integration approach could be also applied to compute any posterior expectations at interest such as posterior mean and variance, which gives the single-number estimate of the relevance. In addition, by further employing the independence property between posteriors, we can calculate the preference probability between documents u_i and u_j by

$$\begin{aligned} P(u_j \succ u_i) &= P(R_j > R_i) \\ &= \int\int_{R_j > R_i} p_j(R_j) p_i(R_i) dR_j dR_i \\ &= \int_0^1 p_j(R_j)\Phi_i(R_j) dR_j, \\ &\approx \frac{1}{B}\sum_{b=1}^{B} p_j\left(\frac{b-0.5}{B}\right)\Phi_i\left(\frac{b-0.5}{B}\right). \quad (27) \end{aligned}$$

Finally, we conclude this subsection with Algorithm 1, which shows how to obtain these exponents with a single pass through the click log for a given query with n search instances. As shown by Eq. 26, the posterior of R_j is fully characterized by $M(M+1)/2 + 1$ numbers, so Algorithm 1 is essentially a counting procedure, and we will further illustrate the algorithm by an example in Section 3.4.

A more flexible choice for each univariate prior $p(R_j)$ is the beta distribution, such that the prior mean and smoothness can be set flexibly by choosing appropriate parameters in the beta family. Algorithm 1 still applies to this case, and the only amendment is that we may need to augment the exponent vector $\mathbf{e}$ by a single additional dimension to accommodate the $(1-r_j)$ term coming from the prior. Similarly, other prior distribution in the parametric polynomial form could also be imposed.

3.3 Parameter Estimation

To estimate model parameters $\beta = \{\beta_{r,d} | (r,d) \in \mathcal{T}\}$, we first compute the likelihood for each search instance I_k in the training set, assuming independent prior on document relevance. From Eq. 20, we have

$$
\begin{aligned}
P(\mathbf{C}^k) &= \int_{\mathbf{S^k}} P(\mathbf{C}^k|\mathbf{S}^k) P(\mathbf{S}^k) d\mathbf{S}^k \\
&= \prod_{i=1}^{M} \int_0^1 (\beta_{r_i^k,d_i^k} S_i^k)^{C_i^k} (1 - \beta_{r_i^k,d_i^k} S_i)^{1-C_i^k} dS_i^k \\
&= \prod_{i=1}^{M} \left(\beta_{r_i^k,d_i^k}/2\right)^{C_i^k} \left(1 - \beta_{r_i^k,d_i^k}/2\right)^{1-C_i^k} \qquad (28)
\end{aligned}
$$

Therefore the log-likelihood can be derived and rewritten as

$$
\begin{aligned}
&\ell(\beta; \mathbf{C}^{1:n}) \\
&= \sum_{k=1}^{n} \sum_{i=1}^{M} \Big(C_i^k \log(\beta_{r_i^k,d_i^k}/2) + (1 - C_i^k) \log(1 - \beta_{r_i^k,d_i^k}/2) \Big) \\
&= \sum_{(r,d)\in\mathcal{T}} \Big(N_{r,d} \log(\beta_{r,d}/2) + \widetilde{N}_{r,d} log(1 - \beta_{r,d}/2) \Big), \qquad (29)
\end{aligned}
$$

where the counts $N_{r,d}$ and $\widetilde{N}_{r,d}$ are defined in a similar fashion as the exponents in Eqs. 23 and 24:

$$
N_{r,d} = \sum_{k=1}^{n} \sum_{i=1}^{M} \mathbb{I}\Big[(C_i^k = 1) \wedge (r_i^k = r) \wedge (d_i^k = d)\Big] \qquad (30)
$$

$$
\widetilde{N}_{r,d} = \sum_{k=1}^{n} \sum_{i=1}^{M} \mathbb{I}\Big[(C_i^k = 0) \wedge (r_i^k = r) \wedge (d_i^k = d)\Big] \qquad (31)
$$

Maximizing the log-likelihood function in Eq. 29 can be again decomposed to $|\mathcal{T}|$ univariate problem, and for each (r,d), we have

$$
\frac{\partial \mathcal{L}(\beta)}{\partial \beta_{r,d}} = \frac{N_{r,d}}{\beta_{r,d}} - \frac{\widetilde{N}_{r,d}}{2 - \beta_{r,d}} \qquad (32)
$$

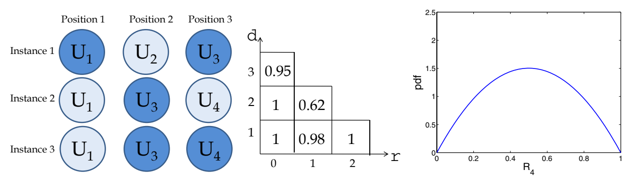
with the optimized value

$$
\widehat{\beta}_{r,d} = \min\left\{1, \frac{2N_{r,d}}{N_{r,d} + \widetilde{N}_{r,d}}\right\} \qquad (33)
$$

Note that the counts needed in the parameter estimation can be integrated in Algorithm 1 by adding "$N(j,d) ++$" and "$\widetilde{N}_{r,d} ++$" to lines 07 and 11 respectively. The additional space complexity is only $O(M^2)$ and they are shared across all queries. Parameters can be computed before evaluating the posterior, and there is no iteration between inference and estimation.

3.4 A Toy Example

Figure 2(a) depicts 3 search instances that involve 4 distinct URLs and 3 positions, so $n = 3, M = 3, N = 4$. Nodes shaded with darker color indicate clicks whereas those with lighter color are not clicked. The set of $\beta_{r,d}$ parameters are shown on the right with pre-defined values. Under this setting, the posterior for each document is specified by 7 numbers. We now take u_4 as an example, and initialize its exponent vector $\mathbf{e}_4 = (0,0,0,0,0,0,0)$ because of the usage



(a) Click Data and Parameter Values (b) Posterior for R_4

Figure 2: A Toy Example of BBM Inference

of the uniform prior. The first dimension is for the exponent N_4 and others for $\widetilde{N}_{4,r,d}$.

Since u_4 is not impressed in Instance 1, $\mathbf{e}_4$ is not updated after scanning Instance 1. Then in Instance 2, because u_4 is not clicked on position 3 ($i = 3$) with the preceding click on position 2 ($r_i = 2$), $\mathbf{e}_4$ is updated to $(0,0,0,0,0,0,1)$, which is equivalent to multiplying the factor $(1 - \beta_{2,1} r_4)$ to the existing posterior. Finally, in Instance 3, since u_4 is clicked, the first element of $\mathbf{e}_4$ is incremented to 1, and $\mathbf{e}_4 = (1,0,0,0,0,0,1)$, so given the values for parameters β, we have

$$
p(R_4|\mathbf{C}^{1:3}) \propto R_4(1 - R_4)
$$

with the normalization constant equals 1/6. The normalized posterior is plotted in Fig. 2(b).

While we use pre-defined values, the model parameters β need to be estimated across all queries in practice. Given the 3 search instances, the update formulae for the counts for parameter estimation are

$$
\begin{aligned}
N_{r,d} &= N_{r,d} + (1,2,0,0,1,1), \\
\widetilde{N}_{r,d} &= \widetilde{N}_{r,d} + (2,0,0,1,0,1).
\end{aligned}
$$

3.5 Scaling BBM with MapReduce

A straightforward way of implementing the sequential inference algorithm (Algorithm 1) on an array of machines is to distributively store and update the exponents. However, a number of practical issues remain to be solved, *e.g.*, load balancing, fault tolerance, and data distributions, *etc.*. In this subsection, we discuss how to leverage the MapReduce paradigm to scale BBM to petabyte-scale data.

MapReduce [9] is a programming model and an associated commodity computer-based infrastructure that provide automatic and reliable parallelization once a computation task is expressed as a `Map` and a `Reduce` functions. Specifically, the `Map` function reads in a logical record, and emits a set of (intermediate key, value) pairs. The MapReduce infrastructure then groups together all values with the same intermediate key and pass them to the `Reduce` function. The `Reduce` function accepts an intermediate key and a set of values for that key, and outputs the final results. In this way, a user of the MapReduce infrastructure only needs to provide the two functions, and let the infrastructure deal with practical issues in parallelization. Open-source implementations of MapReduce infrastructure are readily available, *e.g.*, the Apache Hadoop.

To compute the relevance for each query-URL pair, we record all the exponents in a $(M(M+1)/2+1)$-dimentional vector $\mathbf{e}$ in a similar format to Eq. 26. In the context of BBM, each exponent value can be uniquely identified by the tuple (q, u, val), where (q, u) forms the query-URL pair and

$val = r(2M - r - 1)/2 + d$ (from Eq. 25) is the index of the exponent. The variables r, M, d are the same as defined in Section 2. The `Map` functions reads a search instance, scans each position, and emits $((q, u), 0)$ if the position is clicked, and $((q, u), r(2M-r-1)/2+d)$ if not click. So here (q, u) serves as the intermediate key for MapReduce, and the index in the exponent is the value emitted by `Map` function. Then the `Reduce` function simply retrieves the exponents and update the corresponding exponent in the vector. Both functions are sketched in the following:

Algorithm 2 : `Map(I)` – Mapping a search instance I

```
Input:  $I$: current search instance.
        $I.qry$: returns the query,
        $I.phi[i]$: gives the URL on the $i$th position,
        $I.clk[i]$: indicates click on the $i$th position.
Output: $((q, u), val)$: intermediate (key, value) pairs
        for every position
01: $q = I.qry$; $r = 0$;
02: for each position $i = 1, \ldots, M$
03:     $u = I.phi(i)$;
04:     if $I.clk[i] == 1$
05:         $r = i$;
06:         $val = 0$;
08:     else
09:         $d = i - r$;
10:         $val = r(2M - r - 1)/2 + d$;
11:     end
12:     Emit($(q, u)$, $val$);
13: end
```

Algorithm 3 : `Reduce((q,u), valList)`

```
Input:  $(q, u)$: the intermediate key
        $valList$: a list of values associated with $(q, u)$
Output: $((q, u), \mathbf{e})$: $\mathbf{e}$ is the exponent vector for $(q, u)$
1: $\mathbf{e} = \mathbf{0}$;
2: for each $val$ in $valList$
3:     $\mathbf{e}[val]++$
4: end
5: return $((q, u), \mathbf{e})$
```

In the actual implementation on a commercial MapReduce infrastructure, we modify the `Map` function to accommodate the counts for parameter estimation, and use a more complicated `Reduce` function that also calculates the posterior mean for each query-URL pair. To get the preference probability for each (q, u_i, u_j) tuple, we could design another `Reduce` function over q, which computes the preference probabilities for all documents associated with the same query based on the exponent vectors. For incremental updates, we only need to modify line 1 of the `Reduce` function so that we keep the existing values of exponents instead of initializing them to 0. Experimental results will be covered in Section 5.

4. PERFORMANCE COMPARISON

We report on performance evaluation of BBM in this section. We compare BBM with UBM on both effectiveness (using test data log-likelihood) and efficiency (using training time). Most of the experimental settings as introduced below are consistent with our previous work [17].

Query Freq	# Queries	# Search Instances
1-9	792,232	3,773,744 (14.8%)
10-31	253,839	4,155,049 (16.3%)
32-99	75,538	3,995,928 (15.7%)
100-316	23,424	3,905,847 (15.4%)
317-999	6,662	3,526,396 (13.9%)
1000-5000	2,764	6,067,317 (23.8%)

Table 1: Summary of the Test Data Set

4.1 Experimental Setup

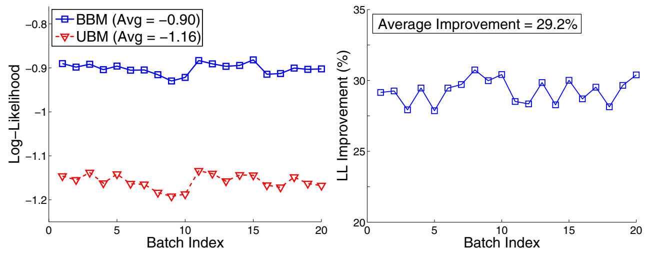
The data set was sampled from a commercial search engine over 2 months in 2008. Only search instances with at least one click are kept for evaluation, because discarded instance tend to have noisy clicks on other search elements, *e.g.*, sponsored ads and query suggestions. We kept at most 10,000 instances for each distinct query even if the query is highly frequent, because clicks for most frequent queries are very similar. For each distinct query, we evenly split its search instances for training and test based on the time stamp, and queries with at least 3 search instances in the training set are kept for subsequent evaluation. This left us 1,154,459 distinct queries and over 51 million search instances in total, which is 10 times larger than the previous experiment in [17]. A summary over the test data is provided in Table 1, where the cutoffs on query frequency are 10^1, $10^{1.5}$, 10^2, *etc.*.

To facilitate robust comparison, the data were further divided into 20 batches of approximately the same size. Each batch consists of 57,723 queries and around 2.5 million search instances on average, whose corresponding standard deviations are 232 and 0.06 million, respectively.

Both models were implemented in MATLAB 2008a on an 8-core machine with 32GB RAM running 64-bit Windows Server 2008. We fit two sets of parameters for navigational queries and informational queries according to [16]. We set the number of bins B in BBM to 100 to get adequate level of accuracy. Similarly, all parameter values for UBM ranges from 0.01 to 1.00, with 0.01 increment. For each query, we compute document relevance and position relevance based on both models. Position relevance is computed by treating each position as a pseudo-document. The position relevance can substitute the document relevance estimates for documents that appear zero or very few times in the training set (the threshold is $\lfloor 2\log_{10}(\text{Query Frequency}) \rfloor$) but do appear in the test set. This essentially smoothes the predictive model and improves the performance on the test set, and also reduces the training cost.

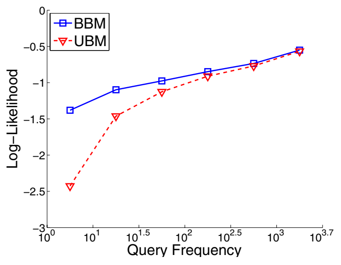
4.2 Model Comparison on Log-Likelihood

Log-likelihood (LL) is a common evaluation metric for model fitness, and it has been previously used to compare click model effectiveness [17]. For each query within every batch, two models were applied on the same training set to obtain document relevance and parameter estimates. Then we computed LL for each search instance in the test set based on Eq. 10, by replacing relevance r_{d_i} by posterior mean (for BBM) or the learned value (for UBM) as well as inserting corresponding values for β parameters. The arithmetic mean over all qualified search instances is reported as the average LL, and a larger value indicates a better model fit. If the average LL improves from ℓ_1 to ℓ_2, the improve-



(a) LL by Batch (b) LL Improvement Ratio

Figure 3: LL Comparison for Each Batch



Query Freq	Avg Ratio(%)	Stdev (%)
1-9	185	1.8
10-31	44.0	0.58
32-99	16.2	0.46
100-316	6.54	0.54
317-999	3.53	0.68
1000-5000	1.56	0.77

(a) LL by Frequency (b) LL Improvement Ratio

Figure 4: LL Comparison for Each Frequency

ment rate is $(\exp(\ell_2 - \ell_1) - 1) \times 100\%$. It measures how much more likely the click sequences in the test data are generated from the more effective click model.

Figure 3 compares the two models in log-likelihood. Specifically, Fig. 3(a) plots the average LL across 20 batches, and Fig. 3(b) illustrates the corresponding improvement rate for each batch. BBM consistently outperforms UBM, with an average improvement of 29.2%.

While Fig. 3 on average LL provides an overall picture, it is also instructive to investigate the performance for each query frequency category. Intuitively, it is easier to predict the clicks for frequent queries than for less frequent ones because of the larger training data size and the relatively more uniform click pattern associated with frequent queries. To testify this intuition and understand the relative performance of BBM and UBM, we compute the average LL for the 6 query frequency categories according to Table 1.

Figure 4(a) presents the comparison of average LL across different frequencies, whereas Figure 4(b) reports the mean and standard improvement ratio computed over 20 batches. BBM consistently outperforms UBM for *all* 20*6 batch-frequency combinations. The performance margin is more dramatic for less frequent queries, because the Bayesian modeling of document relevance in BBM significantly reduces the overall prediction risk. As tail queries constitute a significant portion of search instances (as shown in Table 1), we expect BBM would be more useful in practice than UBM.

4.3 Model Comparison on Time

Figure 5 compares the model training time for both BBM and UBM on each of the 20 batches. BBM is 57 times faster on average than its competitor. The speedup can be decomposed into two parts: (1) the iterative algorithm for learning UBM takes multiple iterations to converge. This number ranges from 16 to 27 (median = 24); (2) A single iteration

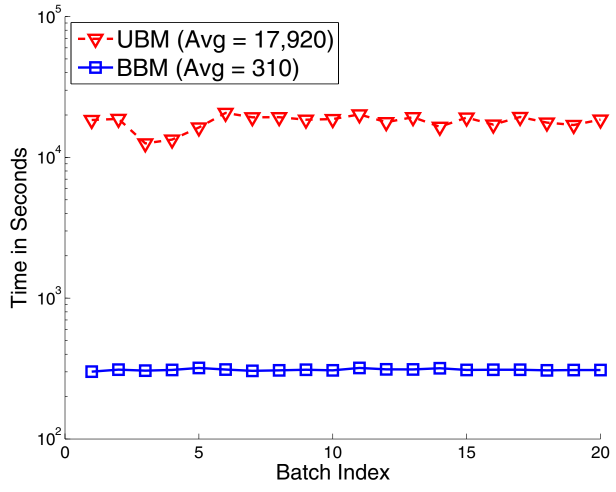


Figure 5: Comparison of Model Training Time. BBM **is 57 times faster than UBM on average.**

of UBM is still 2.5 times as expensive as BBM, because it has to scan through each query to optimize the relevance for each query-URL pair of interest (although we have already reduced the total number with position relevance computation).

The difference in computational cost would be more dramatic for larger-size data because all the counts can still be held in memory in this experiment and no expensive disk I/O is needed to perform multiple passes over the training set. But when the data becomes larger and the numbers can no longer be held in memory, the I/O cost of UBM would be prohibitive. We therefore discuss the parallel-computation alternative in the next section.

5. PETABYTE-SCALE EXPERIMENTS

In this section, we demonstrate how BBM scales to petabyte-scale data with a MapReduce infrastructure. Although UBM can also be implemented in a parallel fashion, through multiple iterations of `Map` and `Reduce`, the time and resource demand is much more expensive than BBM. We therefore focus on the scalability of BBM in this subsection.

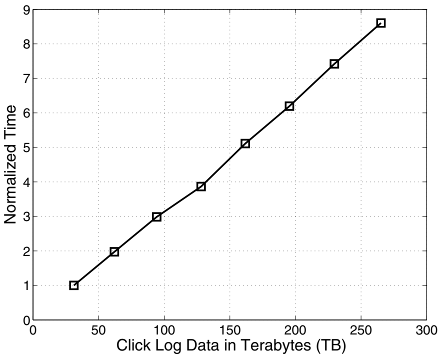
We collected 0.26PB (1PB = 2^{50} bytes) log data over a time interval of 8 consecutive time spans of the same length. Then we created 8 data sets with the kth data set consists of the first k time spans, respectively. Details of these data are summarized in Fig. 6(a); specifically, the largest data set contains 1.16 billion query-URL pairs for 103 million distinct queries.

We implemented BBM on a commercial MapReduce cluster and fit the model to each of the eight data sets. We recorded the total computational time, which is the sum of the CPU time spent on all machines, for each model learning job. Due to confidentiality, the computational time for each job was normalized by being divided by the time of the first job. The normalized computational time w.r.t. the size of the input log is plotted in Fig. 6(b), which exhibits the expected linear relationship because BBM only needs a single pass over the log.

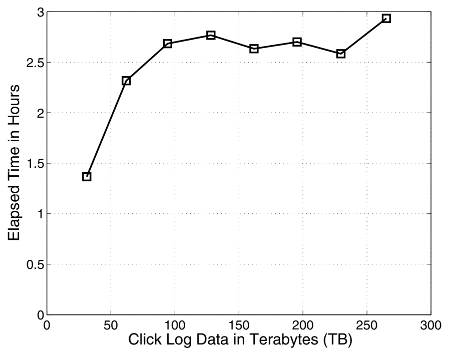
Figure 6(c), on the other hand, demonstrates the power of parallelism. On the cluster the first job takes 1.4 hours to

Job Index	Input Size (TB)	# Query (10^6)	#Query-URL (10^6)
1	31.2	16.3	169.0
2	62.1	30.7	322.9
3	94.3	42.9	454.1
4	128.1	53.9	575.0
5	161.8	63.8	686.4
6	195.5	75.4	816.6
7	229.7	86.3	954.8
8	265.2	103.0	1,155.7

(a) Data Summary



(b) Normalized Computational Load



(c) Actual Elapsed Time

Figure 6: Scaling BBM to Petabyte-scale Data. The largest job of 0.26PB log data takes only 3 hours to finish.

finish, whereas all the rest jobs take no more than 3 hours, despite of their much larger computation loads (as shown in Figure 6(b)). The MapReduce system automatically allocates more machines for the `Map` function for a more demanding job with larger input size. Consequently, given a larger output rate from the `Map` function, a proportionally larger number of machines would be allocated to the `Reduce` function. For this reason as well as the inherent variations in MapReduce scheduling and load balancing, a larger job may end up with a comparable or even shorter elapsed time than a smaller job. However, when all machines in the cluster are exhausted (or busy with other jobs), larger jobs do take longer elapsed time.

6. DISCUSSION AND RELATED WORK

One of the earliest publications on large scale query log analysis appeared in 1999 [25], which presented interesting statistics as well as a simple correlation analysis from the Alta Vista search engine. Thereafter, search logs, especially the click-through data, have been utilized for various applications, *e.g.*, learning to rank [1, 6, 18, 29], query recommendation [5, 31], search boosting and result reordering [4, 24]. A central task in utilizing search log is to understand and model user search and browsing behaviors and click decision processes. Joachims and his collaborators pioneered this direction by presenting a series of studies around some eye-tracking experiments [19, 20], which inspired a series of models that interpret user behaviors with increasing capacity, namely, the cascade model [8], ICM and DCM [17], and UBM [11]. These models were reviewed with details in Section 2 because of their close relationship with the proposed BBM. What distinguishes BBM from the existing models is the combination of enhanced modeling capacity and the scalability that meets the requirements of real-world practice.

Because of the data stream nature of search log, this work also relates to mining data streams [3, 13]. There have been many scalable algorithms developed for different mining tasks on data streams, *e.g.*, classification [10], clustering [15], and frequent pattern mining [14]. In lieu of these studies, the exponent vector $\mathbf{e}$ in BBM is actually the synopsis that is maintained over the stream of search log. Since this synopsis fully characterizes the relevance posterior, it can be utilized by various downstream applications. Note that although this synopsis tremendously reduces the data size, it exactly describes the relevance posterior with no approximation.

BBM can be utilized for a number of downstream applications. In the first place, quantities such as mean and standard deviation can be extracted from the relevance posterior, and can be used as new features to train a better search ranker, in a much similar way as [1, 2]. In the second place, the preference probability from BBM (Eq. 27) can substitute or supplement preferences that are heuristically derived from clicks or human ratings. For example, Joachims and his colleagues proposed a set of rules like "click $\succ$ skip above" to derive preferences from click-throughs within a search instance [18], and later extended them to across multiple instances [22]. The derived preferences are taken as constrains to train a Ranking SVM [18]. With BBM, the preference probabilities can be computed outright from the relevance posteriors, and can actually serve as soft constraints in Ranking SVM, *i.e.*, one URL is preferred to another with a certain probability. We foresee that the soft preferences will lead to a cost-sensitive Ranking SVM, which we will investigate in the future. As a matter fact, the utility of preference probability is not limited to augmenting preferences derived from click-through data or Ranking SVM; instead, we expect most pairwise preference-based learning to rank algorithms (*e.g.*, RankNet [7], RankBoost [12] and FRank [27]) can be married with BBM preference probabilities with appropriate algorithmic designs. For example, RankNet tries to learn a ranking function that agrees the most with known preferences as derived from human ratings. We could replace the human rating preferences with preference probability, and study how well the preference probabilities derived from feedbacks of millions of users compare with well-trained judges.

7. CONCLUSION

In this paper, we proposed BBM, the Bayesian Browsing Model, together with a single-pass exact inference algorithm that can be parallelized. BBM is both effective and efficient: it outperforms a state-of-the-art click model by 29.2% in log-likelihood, while being 57 times faster in a real world click data set. We further implemented the model on a MapReduce cluster, and the model was able to compute the relevance for 1.15 billion query-URL pairs within 3 hours by processing 0.26 petabyte click log. There are many topics to be further explored in the future, among which the usage of preference probability in downstream applications is in the particular spotlight.

Acknowledgements

We would like to thank Wenjie Fu for his great help on preparing this manuscript. Fan Guo and Christos Faloutsos are supported in part by the National Science Foundation under Grant No. DBI-0640543. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR '06*, pages 19–26, 2006.

[2] E. Agichtein, E. Brill, S. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *SIGIR '06*, pages 3–10, 2006.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS '02*, pages 1–16, 2002.

[4] R. Baeza-Yates. Applications of web query mining. *Advances in Information Retrieval*, pages 7–22, 2005.

[5] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *International Workshop on Clustering Information over the Web*, 2004.

[6] M. Bilenko and R. W. White. Mining the search trails of surfing crowds: identifying relevant websites from user activity. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 51–60, 2008.

[7] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

[8] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM '08: Proceedings of the first international conference on Web search and data mining*, pages 87–94, 2008.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, 2004.

[10] P. Domingos and G. Hulten. Mining high-speed data streams. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[11] G. E. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *SIGIR '08*, pages 331–338, 2008.

[12] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal Machine Learning Research*, 4:933–969, 2003.

[13] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, 2005.

[14] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining frequent patterns in data streamsat multiple time granularities. *Next Generation Data Mining*, 2003.

[15] S. Guha, A. Meyerson, N. Mishra, and R. Motwani. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15, 2003.

[16] F. Guo, L. Li, and C. Faloutsos. Tailoring click models to user goals. In *WSCD '09: Proceedings of the workshop on Web search click data*, 2009.

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

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

[19] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05*, pages 154–161, 2005.

[20] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems*, 25(2):7, 2007.

[21] T. Minka. Expectation propagation for approximate bayesian inference. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 362–369, 2001.

[22] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *KDD '05*, pages 239–248, 2005.

[23] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 521–530, 2007.

[24] M. Shokouhi1, F. Scholer, and A. Turpin. Investigating the effectiveness of clickthrough data for document reordering. In *ECIR'08*, pages 591–595, 2008.

[25] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[26] A. Trotman. Learning to rank. *Information Retrieval*, 8(3):359–381, 2005.

[27] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. FRank: a ranking method with fidelity loss. In *SIGIR '07*, pages 383–390, 2007.

[28] S. Wedig and O. Madani. A large-scale analysis of query logs for assessing personalization opportunities. In *KDD '06*, pages 742–747, 2006.

[29] G.-R. Xue, H.-J. Zeng, Z. Chen, Y. Yu, W.-Y. Ma, W. Xi, and W. Fan. Optimizing web search using web click-through data. In *CIKM '04*, pages 118–126, 2004.

[30] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Trans. Inf. Theory*, 51(7):2282–2312, 2005.

[31] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *WWW'06*, pages 1039–1040, 2006.