An Online Learning Framework for Refining Recency Search Results with User Click Feedback

TAESUP MOON, WEI CHU, LIHONG LI, ZHAOHUI ZHENG, and YI CHANG, Yahoo! Labs

Traditional machine-learned ranking systems for Web search are often trained to capture stationary relevance of documents to queries, which have limited ability to track nonstationary user intention in a timely manner. In recency search, for instance, the relevance of documents to a query on breaking news often changes significantly over time, requiring effective adaptation to user intention. In this article, we focus on recency search and study a number of algorithms to improve ranking results by leveraging user click feedback. Our contributions are threefold. First, we use commercial search engine sessions collected in a random exploration bucket for reliable offline evaluation of these algorithms, which provides an unbiased comparison across algorithms without online bucket tests. Second, we propose an online learning approach that reranks and improves the search results for recency queries near real-time based on user clicks. This approach is very general and can be combined with sophisticated click models. Third, our empirical comparison of a dozen algorithms on real-world search data suggests importance of a few algorithmic choices in these applications, including generalization across different query-document pairs, specialization to popular queries, and near real-time adaptation of user clicks for reranking.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.5 [**Information Storage and Retrieval**]: Online Information Services; I.6.0 [**Simulation and Modeling**]: General

General Terms: Design, Algorithms, Performance

ACM Reference Format:
Moon, T., Chu, W., Li, L., Zheng, Z., and Chang, Y. 2012. An online learning framework for refining recency search results with user click feedback. ACM Trans. Inf. Syst. 30, 4, Article 20 (November 2012), 28 pages.
DOI = 10.1145/2382438.2382439 http://doi.acm.org/10.1145/2382438.2382439

1. INTRODUCTION

Ranking a list of documents based on their relevance to a given query is the central problem in various search applications of the Internet. Machine-learned ranking algorithms have been shown highly effective for generalizing to unseen data from labeled training data and have been very successful especially in commercial Web search engines [Burges et al. 2005, 2007; Cortes et al. 2007; Freund et al. 2003; Joachims 2002a; Liu 2009; Zheng et al. 2008]. Usually, these algorithms learn a ranking function based on editorial judgments—relevance labels provided by human editors. A critical assumption here is that the relevance of documents for a given query is more or less stationary over time, and therefore, as long as the coverage of

T. Moon is currently affiliated with the University of California Berkeley and W. Chu is currently affiliated with Microsoft.
Authors' address: T. Moon, W. Chu, L. Li, Z. Zheng, and Y. Chang, 701 First Avenue, Sunnyvale, CA 94089; email: tsmoon@ymail.com.

DOI 10.1145/2382438.2382439 http://doi.acm.org/10.1145/2382438.2382439

training set is broad enough, the ranking function learned from the training set would be sufficient to generalize to unseen data in the future. This assumption is often valid in Web search, especially for popular queries like "*yahoo*," for which document relevance is indeed (almost) static.

However, there are other important categories of applications where document relevance to a query may change over time. One such example is the recency ranking problem in Web search: when breaking news emerges, a document that used to be most relevant to a query may be superseded by others that have more timely information about the news; see Section 4 for a concrete example. A key challenge for such problems is to track user intention in a timely fashion.

An interesting attempt was taken recently for tracking nonstationary document relevance [Dong et al. 2010a]. The authors devised time-varying features that reflect freshness of documents and utilized recency demoted labels provided by human editors that explicitly modify the relevance target values in the training set. Their results showed an improvement of ranking qualities for time-sensitive queries. However, their approach is still based on editorial judgments and so limited for two reasons. First, obtaining high-quality training data is hard. Implementing more fine-grained time-varying features, such as features from the time series of clicks that can accurately follow the relevance drifts is considerably subtle and complex since carefully testing and selecting good features is a long and complicated process. Also, obtaining laborious recency demoted labels from human editors not only is expensive, but also can be inaccurate in correctly representing the temporal variation of document relevance. Second, even when we can come up with such complex and expensive data to batch-train a ranking function, tracking actual user intention remains challenging due to the very unpredictable nature of how user intention evolves over time.

In this article,[1] we investigate how to leverage user click feedback to complement and improve such editorial-judgment based ranking systems. Our rationale is that, particularly for recency queries, instantaneous click trends on the top portion of the ranking list are important indicators of document relevance. Such signals allow us to extract subtle information that may be hard for human editors to foresee when they provide relevance judgments. In particular, we explicitly track the click-through rate (CTR) of a query-document pair using a linear combination of extracted features, including the editorial-judgment based ranking function's score. Based on search results returned by the current search engine, we propose a methodology for an online reranking to further improve search results for recency queries near real-time. We use user click as labels for training the CTR models in either batch or online mode.

In order to justify our rationale mentioned before and develop a methodology for exploration for learning and evaluation of different algorithms, an "exploration bucket" was set up for a small random portion of live traffic for recency-classified queries in a commercial search engine. Within the bucket, the top URLs returned by the search engine was randomly shuffled and the corresponding click feedback was logged. This bucket is integral to the work reported in this article, and the details on how we utilize this bucket data for learning and evaluation are described in the subsequent sections.

The rest of the article is organized as follows. In Section 2, we discuss several closely related previous research and highlight the difference and contributions of our work. In Section 3, we describe our exploration bucket, which is essential for both learning and evaluation. Using the data in this bucket, we present a motivating example in Section 4, showing the necessity of taking into account temporal variation of document relevance reflected in user click feedback. Our methods are detailed in Section 5

[1]The present article extensively augments our preliminary results reported in an extended abstract [Moon et al. 2010].

and empirically evaluated using the exploration bucket data in Section 6. Section 7 gives some possible future directions and concludes the article with a summary of our contribution.

2. RELATED WORK AND CONTRIBUTION OF THE ARTICLE

Our work is closely related to several previous researches. We now summarize the related work and point out the difference and contributions of our work.

2.1. Temporal Modeling for Ranking

Recently, several work proposed approaches regarding taking the temporal aspects of documents and queries into account to improve search ranking. For example, Elsas and Dumais [2010] considered the temporal dynamics of document content and applied that knowledge to improve search ranking for navigational queries, and Dai and Davison [2010] tried to capture the temporal variations in the link structures between Web pages and reflect them in improving the relevance and freshness of ranking. Jones and Diaz [2007] and Kulkarni et al. [2011], on the other hand, focused on the query side of temporal variations and tried to detect and classify the profiles of time-sensitive queries.

Moreover, as mentioned in Section 1, Dong et al. [2010a] show how machine-learned ranking system can take advantage of different signals that capture the temporal variations of relevance of documents to recency queries. They introduced query classifiers to detect time-sensitive queries, implemented time-varying features that reflect document freshness, and recency demoted labels provided by human editors. More recently, improvements were made by introducing additional click or micro-blogging (e.g., Twitter) service related time-varying features [Dong et al. 2010b; Inagaki et al. 2010]. Note that the results in Elsas and Dumais [2010] and Dai and Davison [2010] can be also used as time-varying document features, and those in Jones and Diaz [2007] and Kulkarni et al. [2011] can be used as additional signals for the recency query classifier in Dong et al. [2010a], as well.

The difference between our work and these examples is that we explicitly use the click feedback as target to learn the temporal variation of the relevance of documents for recency queries, whereas the given papers do not. While such an approach would not be beneficial for all queries, we show in Section 4 how the click feedback can be helpful in reflecting relevance of documents for recency queries via a motivating example. Furthermore, our approach complements prior work; it can be applied to the rankings that are generated by any of the work and further improve the quality of rankings by utilizing the click feedback.

2.2. Click Models and Click-Based Reranking

Using users' click feedback to improve ranking quality of a search engine has been extensively studied before as well. Various click models [Chapelle and Zhang 2009; Dupret and Liao 2010; Guo et al. 2009; Zhu et al. 2010] are developed based on click log data, of which goals were to either remove positional biases of clicks in the log data or extract some relevance signal so that they can be used as useful features for batch-trained ranking functions. Several other works also used click feedback data to directly modify search rankings by reranking; Ji et al. [2009], Kang et al. [2011], and Liu et al. [2009] extracted preference information on rankings from the past click log data and used it in a machine-learned reranking system.

There are several differences between our work and these examples. First, all of that work is based on batch schemes that are trained on *static* training data, and

thus the relevance features or the reranking functions obtained in that work are often computed in the average sense and are hard to reflect the temporal variations of document relevance. In contrast, the reranking algorithm in our work can update its model on-the-fly as click data accumulate and can quickly track the temporal variations reflected in the click feedback.

Second, the schemes in the given work are all based on the click feedback only from the default search ranking, whereas our scheme explores the click feedback on other rankings as well to find the potentially better ranking quickly. Finding better ranking by exploration has been also considered in Radlinski and Joachims [2007] and Yue and Joachims [2009], too, but the settings in those papers are different from ours. Radlinski and Joachims [2007] devised a strategy to actively explore to collect the click feedback that is helpful in improving the ranking quickly, but the results were based on simulated user clicks rather than real ones. Yue and Joachims [2009] developed the dueling bandit approach, but it required a special functionality of the retrieval system to interleave two different ranking results.

Third, most of the previous click models do not generalize across different queries, but instead compute a single number for each query-document pair. Here, our models can generalize relevance information among different query-document pairs by working in a common feature space; it thus allows one to deal with tail/infrequent queries with sparse training data. Furthermore, our models also provide a mechanism so that highly accurate relevance predictions can be made for popular/frequent queries. It should be noted that an interesting click model was recently proposed [Zhu et al. 2010] that also relied on features for cross-query generalization. While the focus there was to understand position bias and user click behavior, their click model may naturally be used within our online-learning framework, which is left as an interesting direction for future work.

2.3. Collaborative Filtering and Personalized Article Recommendation

Using feature-based models to track temporal variations of relevance reflected on the user feedback has been considered in several collaborative filtering and personalized content recommendation problems. Koren [2009] devised a scheme to capture temporal dynamics of user ratings on items in a collaborative filtering problem. Using techniques to balance exploration-exploitation has been also considered in the problems of personalized article recommendation on Web portals. While models in earlier works [Agarwal et al. 2009] did not use features so that the scheme can generalize to unseen articles, there have been efforts on developing feature-based models more recently, such as the LinUCB algorithm [Li et al. 2010] that uses a similar linear model as ours, and the warm-start solution by Agarwal et al. [2010].

The difference of our work from these is as following; Koren [2009] focused rather on long-term dynamics and did not consider the cold-start problem, which is critical to our recency ranking problem given the large volume of of tail queries as shown in Section 3. Our work, on the other hand, incorporates a warm-start model. The work on personalized article recommendation work [Agarwal et al. 2010; Li et al. 2010] maintain models for a small number of articles/items, and so have not demonstrated capacities of learning with a much larger content pool, as in our space of query-document pairs. Another difference is that their model was item-specific, whereas our model consists of both a global model that applies to all queries and documents as well as a specific bias term for each query-document pair. Stern et al. [2009] independently considered a similar idea for large-scale personalized recommendation, but imposed a Bayesian framework that is different from ours.

2.4. Our Contribution

In summary, the contributions of our article are the following.

(1) We use a unique exploration bucket data collected from a real, commercial search engine and propose a framework of learning and evaluating various reranking schemes for recency queries.
(2) We use click feedback as a direct target to track the varying relevance of documents for recency queries and use online reranking model that utilizes explore-exploit techniques.
(3) We use mixture of feature-based parametric model and nonparametric bias terms so that the reranking model can generalize to tail queries and specialize to popular queries simultaneously.

Moreover, as also mentioned in the Introduction, this article significantly augments and clarifies our conference version [Moon et al. 2010]. Followings are the summary of the additional contribution of this journal version:

— significantly enlarged baseline schemes;
— analyses on the exploration bucket data;
— thorough result analyses.

In the following sections, we describe the details about the exploration bucket data, our methods, and experimental results.

3. EXPLORATION BUCKET DATA

As described in the previous section, we set up a bucket to collect exploration data from a small portion of live traffic at a commercial search engine. The bucket started on Jan. 29, 2010 and ended on Feb. 4, 2010. Throughout these days, we collected 399, 880 search sessions that contained 61,904 recency classified queries, after removing nonrandom sessions corrupted by business rules. We used the recency query classifier described in Dong et al. [2010a, Section 4]. The ranked list for those queries were generated by the recency ranking function trained as described in Dong et al. [2010a] and the ranking score for each query-document was recorded. For each session, we randomly shuffled the top four results and logged the permutation id of each shuffled permutation (a total of 4! = 24 of them) and user clicks on the corresponding permuted ranking results.

The collected data is very sparse and long-tailed, as shown in Figure 1, in which 92.4% of queries were issued no more than 10 times and more than half of queries were issued just once. The reason for this sparsity is that the recency query classifier utilizes some language model to determine the queries that are related to each other, which causes some recency-related idiosyncratic, less popular queries, such as different word orderings or typographically wrong queries, to be classified as recency queries.

By doing the random shuffling, we are able to collect user click feedback on each document without positional bias, and such feedback can be thought of as a reliable proxy on relevance of documents. Note that the effect on user experience of shuffling would not be as severe as that for navigational queries, since the relevance differences of top-ranked documents to recency queries would not be as dramatic as those for navigational queries. Also, we chose a reasonably small number, 4, in order to limit the negative impact on user experience in the exploration bucket.

Another byproduct of our exploration bucket is that we can accurately observe the positional biases of clicks, which are not easy to obtain from published works. Specifically, for the top four URLs in each session, we can infer the original ranking of the search engine simply by the recorded ranking scores. For each session, the URL with

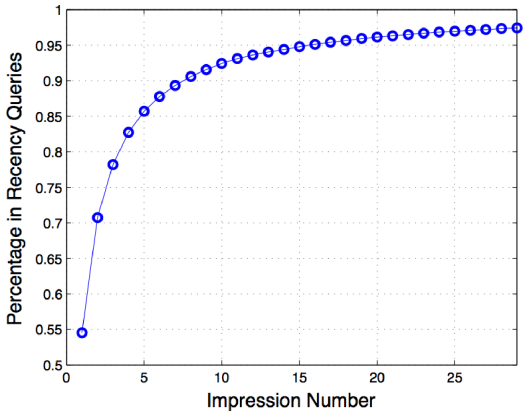


Fig. 1. Recency query impression.

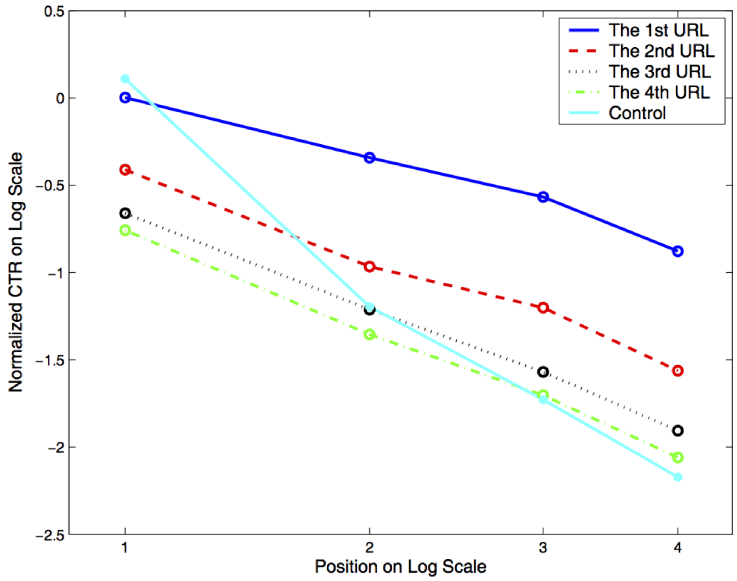


Fig. 2. Marginal nCTRs of four types of URL at the top four positions in exploration bucket, along with the nCTRs on the original order as control. All are on logarithm scale.

a highest ranking score is called the "1st URL." These URLs were displayed an equal number of times in all four positions in the exploration bucket because of the random shuffling. We can then estimate the aggregate Click-through rate (CTR) of the 1st URLs in each of the four positions, as depicted by the blue line in Figure 2.[2] Such nCTRs are marginal since we have taken all possible layouts (of other URLs) into account, thanks to the uniform randomness in the exploration bucket. Similarly, marginal nCTRs of the 2nd, 3rd, and 4th URLs of all sessions are also plotted in Figure 2.

Interestingly, lines of these four marginal nCTRs are almost parallel to each other, which implies the user click patterns follow the well-known power-law distribution.

[2]To protect business-sensitive information, the article reports only the *normalized CTR* or *nCTR*, which is the CTR multiplied by a constant. We will use them interchangeably if there is no confusion.

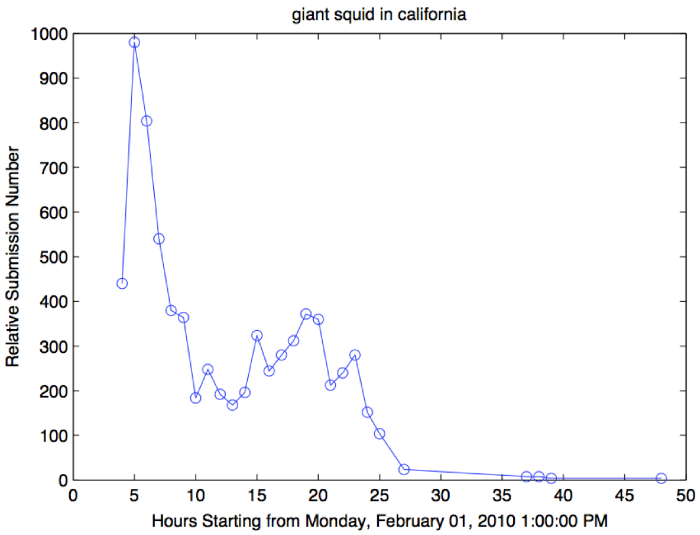


Fig. 3. Impression count in the exploration bucket of the query "giant squid in California."

The slope indicates the intrinsic positional biases in the displayed layout of search results. To further illustrate the conditional effect on user click patterns, we also present the nCTRs of the original display order. It corresponds to the steeper straight line of Figure 2, indexed by "Control" in cyan. We observed that the nCTR of the 2nd URL at the 2nd position conditioned on the 1st URL at the 1st position is much lower than the marginal CTR of the 2nd URL at the 2nd position. The drop indicates a negative conditional effect from the 1st URL at the 1st position. For the 3rd URL at the 3rd position and the 4th URL at the 4th positions, we observed a similar conditional effect.

The apparent positional biases shown in Figure 2 would be taken into account in devising our reranking algorithm. Moreover, the fact that the four lines other than the "Control" do not cross each other shows that, on average, the original ranking is doing a decent job in ranking the URLs also with respect to CTRs. However, in this article, we show that we can do better than this by reranking the URLs appropriately so that the overall CTRs of reranked results can be further improved.

In Section 5, we show how we use the exploration bucket data in our learning method, and in Section 6.1 we show how the data is used for unbiased evaluation of various algorithms.

4. MOTIVATION

Before describing the technical details of our method, let us first look at a concrete example found in exploration bucket to illustrate our motivation, and then summarize the challenges that we confront in recency search results. This example will be revisited in our discussion of experimental results.

In our exploration bucket, "giant squid in California," is a typical recency query, which appeared on February 1, 2010, and then disappeared after two days in our exploration bucket data. Figure 3 shows the impression statistics of the query with respect to time, which clearly shows nonstationary temporal statistics.[3] This query is

[3]To avoid revealing business-sensitive data, we normalize the query submission number by multiplying it with a positive number.

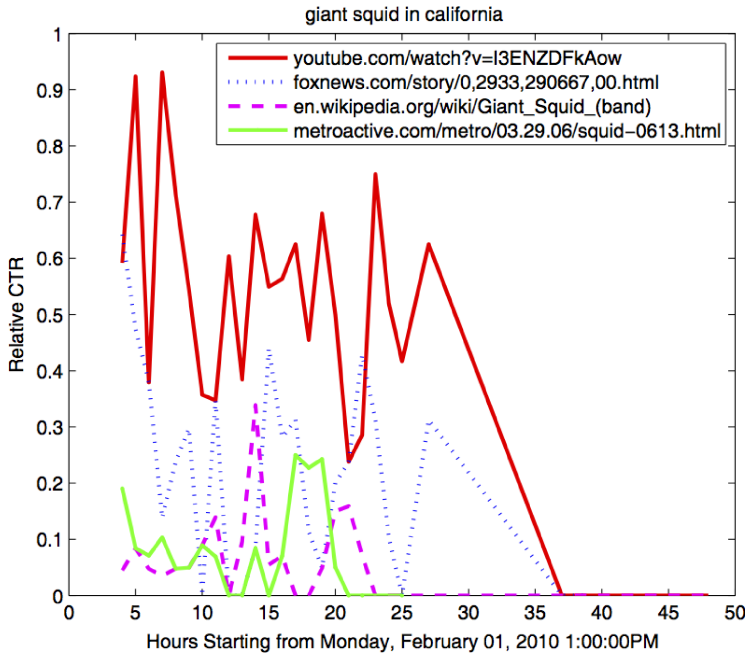


Fig. 4. Hourly CTR of the top 4 URLs associated with the recency query "giant squid in California." After the 30th hour, no more click of this query was observed in the exploration bucket.

related to local news in California. At that time, a number of giant squids weighing up to 60 pounds had swum into waters off the Californian coast and were caught by sport fishermen by the hundreds. It seems apparent that many people submitted the query "giant squid in California" to find related materials of the local news from the search engine.

To study user click patterns on the URLs associated with the query, we again examined the top four URLs in the exploration bucket. The four URLs were retrieved by the default ranking function for recency queries in the search engine, and the corresponding default ranking on the top four URLs was:

(1) `foxnews.com/story/0,2933,290667,00.html`,
(2) `en.wikipedia.org/wiki/Giant_Squid_(band)`,
(3) `metroactive.com/metro/03.29.06/squid-0613.html`,
(4) `youtube.com/watch?v=I3ENZDFkAow`.

Based on their contents, the four Web pages can be categorized as "a news story page," "a background knowledge page," "a relevant page," and "a video page." As we randomly shuffled the display order of the top 4 URLs in the exploration bucket, each URL had the same chance to be displayed at each position. Therefore, position bias on clicks for these URLs was removed. The nCTRs of the four URLs observed in our exploration bucket session data are presented in Figure 4. Clearly, although the initial nCTRs were similar, the "video" content ended up receiving most clicks, while the "news story" was runner-up. This shows that while our recency ranking function made a reasonable decision (by putting the "video" content within top 4), it yet failed to accurately predict the ranking with respect to the users' preference reflected in

the CTR patterns on the URLs. Moreover, we note that unless we actually see this click patterns, it would be extremely hard for human editors to predict such relevance patterns for these top 4 URLs.

Based on these observations, two challenges are identified for the recency ranking problem.

— *Relevance Drifting*. As illustrated by the previous case, document relevance may vary significantly over time. These examples show the limitations of the editorial judgment based, batch learning framework in tracking such temporal dynamics. In general, it is very difficult to design features that can correctly reflect the temporal variances of relevance and for editors to predict the relevance labels before observing the actual user behaviors. How to rapidly track such drift would be a major challenge of recency ranking.
— *Data Sparsity*. Due to the reason specified in Section 3, many recency queries have few impressions. Hence, learning across queries (i.e., generalization) would be important.

In addition, we note that keeping track of dynamic content for recency queries to generate reasonably good top documents is also a critical challenge. However, as mentioned in Section 2.1, the topic is outside the scope of this article; various approaches in Section 2.1 can be applied to generate good top documents with respect to the editorial labels, and our focus is to propose a method that further refines the ranking generated from any such approaches by tracking the temporal variation of relevance reflected in the click feedback.

5. OUR METHOD

To address the two challenges in the previous subsection, it is necessary for a ranking module to detect and track the nonstationarity of user interests reflected by click feedback patterns.

— *Relevance Drifting*. In contrast to waiting for editorial judgments to update ranking results for a recency query, our algorithm updates relevance scores near real-time based on user click feedback on the ranked list of documents. Not only can it avoid the expensive editorial judgments, but it can also quickly adapt to the varying relevance that truthfully reflects real user intent.
— *Data Sparsity*. The ranking model in our method works in a common feature space shared by all query–document pairs. It is then able to generalize click feedback of a pair to other pairs via feature values. Furthermore, we will also show the benefits of maintaining bias terms, or latent features, dedicated to popular query–document pairs in the experimental results.

Ideally, if we knew the true relevance of every document for a recency query, we would be able to display the optimal ranked list of documents. However, estimating the relevance of all query-document pairs is unnecessarily difficult: similar to general Web search in which most of the relevance metric focus on the top portion of the ranking, it is more important to get better estimates for high-quality documents for recency queries. Therefore, assuming that a baseline, the default recency ranking function, already retrieves reasonably good quality documents at the top, we propose a *reranking approach* that estimates the (possibly time-varying) relevance of top-ranked documents returned by the default ranking function and optimizes the reranking by those estimates.

To do so, our method assumes the availability of a real-valued reward signal, derivable from user clicks, to refine its reranking model. While our technique is general

enough and can be combined with essentially any such click-based signals [Radlinski et al. 2008], in this work, we focus on CTR at position 1 (CTR@1) as the reward for the sake of concreteness. The use of this metric can be justified as follows.

— First, CTR@1 is a widely adopted, critical metric when deploying general machine-learned ranking function for Web search in practice. Indeed, our empirical findings in Section 6.3.4 also confirms that lifts in CTR@1 are consistent with lifts in other important click metrics that reflect the relevance of the whole ranked list of documents.
— Second, our example in Section 4 shows that CTR@1 can be a more objective metric than editorial judgment-based metrics for the recency queries, for which relevance judgments are difficult to obtain in advance.
— Third, our CTR@1 based reranking would not hurt other traditional editorial judgment-based metrics, for instance, NDCG (Normalized Discounted Cumulative Gain) [Järvelin and Kekäläinen 2002], too much, either. The reason is that we are only focusing on the very top portion of the ranked list of the baseline ranking function, and we are assuming that those metric values on the baseline ranking are already reasonably high.

As mentioned in Section 2.2, there have been a lot of previous research on click models and click-based reranking. However, as pointed out in Section 2.2, most of those work are batch trained schemes on click feedback data from the default rankings, and feature-based generalization has seldom been used. In contrast, our method is based on online scheme together with explore-exploit strategy, and uses a hybrid model with both feature-based parametric model and nonparametric bias terms. We will also describe in Section 6.3.2 how such click model-based reranking method compares to our method, and why directly comparing the two is not the focus of this article.

Finally, our approach is readily applicable for optimizing other real-valued metrics like session length and revenue. It may be possible to be extended to more sophisticated signals like skip-above [Joachims 2002b].

5.1. Settings

We consider the following reranking framework, naturally modeled as a round-by-round process. At round t, the process proceeds as follows.

(1) A user arrives and types in a query q_t.
(2) The default recency ranking function generates an ordered list of s documents with highest relevance scores. Then, our reranking function reorders these s documents and presents to the user the reordered ranked list $(u_{t,1}, \ldots, u_{t,s})$.
(3) The user then provides feedback $r_t = (c_{t,1}, \ldots, c_{t,s})$ on our reranking result, where $c_{t,i} = 1$ if a user clicked on the document at potision i, and 0 otherwise.
(4) Based on the user feedback r_t, the reranking function is updated and is used for the next round $t + 1$.

From the described process, we see that our reranking function is inherently an online algorithm that updates its logic on the fly from the sequential observation of click feedback. In order to efficiently implement and update our reranking function, we implement a common feature vector for every query-document pair, (q, u), and denote it as $\mathbf{x}_{qu} \in \mathbb{R}^d$. In our experiment, a total of $d = 51$ features were used. These features include regular query-specific (e.g., number of words in a query), document-specific (e.g., spam classification score of a document), and query-document-specific (e.g., number of times a query appears in a given document) features used in ordinary machine learned ranking functions, and more importantly, the ranking score generated by the

default, editorial judgment-based recency ranking function. Our reranking function is then defined to be a function that predicts the CTR@1 of each (q, u) as a function of $\mathbf{x}_{qu}$ and possibly of some latent features, and the function gets updated based on observing users' click feedback in an online fashion. The detailed function form and update formula are described in the subsequent two sections.

Given our goal of maximizing CTR@1 in the reranking results, it is tempting for an online algorithm to follow a *greedy* strategy: that is, it always ranks (for the present query at hand) the documents in the order of the highest CTR@1 estimates and updates the function parameters solely based on the user feedback for the algorithm's ordered list. While this greedy approach is intuitively desirable, it can be detrimental in practice. This is because, as can be seen in the interactive round-by-round process described before, the reranking algorithm obtains user feedback *only* from the orderings that it has displayed to the user. Therefore, if an algorithm mistakenly orders the documents, a greedy reranking strategy can prevent it from collecting user feedback for other (potentially better) rankings and correcting its mistake to find the most relevant document on the top for maximizing CTR@1. Consequently, the algorithm has to balance two conflicting goals: (a) "exploitation" — to display in the first position most relevant documents to maximize reranking quality (in our case, to maximize user clicks), and (b) "exploration" — to display documents for the purpose of collecting data to further improvement. The exploration/exploitation tradeoff described before is a defining characteristic of a class of problems known as bandit problems [Robbins 1952], which has received considerable attention recently for Internet-related applications [Radlinski and Joachims 2007; Yue and Joachims 2009].

In the following Section 5.2 and Section 5.3, we present the basic function form of our online reranking function and its update formula provided the user feedback r_t is given, respectively. Then, in Section 5.4, we describe how we vary our scheme in order to cope with the explore/exploit tradeoff explained previously and accelerate learning speed.

5.2. CTR@1 Estimation Model

Although many alternatives exist, we choose our reranking function to be linear in the feature vector $\mathbf{x}_{qu}$. This choice allows us to derive exact update rules and simplify the exposition. Other nonlinear models may also be used, although numerical approximation is unavoidable in general when optimizing their model parameter. In particular, we have tried logistic and probit regression [Graepel et al. 2010], and observed similar performance as the linear model.

Since we try to maximize CTR@1, it is natural to find a function that estimates CTR@1 of a (q, u) pair for reranking. Once the feature vector $\mathbf{x}_{qu}$ of length d is given for a (q, u) pair, a linear combination of them is used to estimate CTR@1. In fact, we will use the most general form that captures all variants useful in our experiments:

$$\text{CTR@1}(q, u) = \boldsymbol{\beta}^\top \mathbf{x}_{qu} + b_{q,u}, \tag{1}$$

where the vector $\boldsymbol{\beta} \in \mathbb{R}^d$ contains the coefficients shared by all query-document pairs, and $b_{q,u} \in \mathbb{R}$ is a (q, u)-specific *bias* term. Both $\boldsymbol{\beta}$ and $\{b_{q,u}\}$ are to be learned by our algorithm.

Clearly, user click feedback on any query-document pair may be used to estimate $\boldsymbol{\beta}$, which in turn can be used to predict CTR@1 for other query-document pairs. Therefore, the linear part of $\boldsymbol{\beta}$ in Equation (1) addresses the data sparsity challenge by allowing generalization across different queries and documents. However, a linear model in the features may not be sufficiently accurate to capture the real CTR@1. The bias

terms thus provide a mechanism to correct the residuals and to yield more accurate estimates.

Because of these bias terms, it may appear that Equation (1) uses too many free parameters. However, as will be cleared in the next subsection, we use regularization to control the magnitude of these terms, so the bias terms will be essentially zero except for popular query-document pairs. Consequently, these terms can be used to yield a highly accurate CTR@1 estimate for popular (q, u), while for unpopular (q, u) (which suffer the data sparsity issue most) we essentially use the linear estimate $\boldsymbol{\beta}^\top \mathbf{x}_{qu}$. Such a dichotomy is done automatically within the regularization framework.

In summary, when optimized with regularization, our hierarchical linear model can adapt to the data in a smooth fashion by automatically balancing model complexity (which bias terms are used effectively) and frequencies of query-document pairs.

5.3. Parameter Update Rule

This subsection addresses the problem of parameter updates for model (1). We first describe how to fit the parameters if we are given a static set of data, then extend the update rule to the online case when data arrive sequentially, and finally discuss a few practical issues when deploying the update rules in large-scale ranking systems.

5.3.1. Batch Parameter Fitting. Suppose we are given a set $\mathcal{D}$ of t data in the form of $\{(q_i, u_i, c_i)\}_{i=1,2,\ldots,t}$, where $c_i \in \{0, 1\}$ is the click feedback for (q_i, u_i) provided by the i-th user. Let $\mathcal{P}$ be the set of distinct (q, u) pairs observed in $\mathcal{D}$, and $N = |\mathcal{P}|$. For brevity, denote the feature vector for the (q_i, u_i) pair as $\mathbf{x}_i$.

A standard approach to learn the parameters in Equation (1) for CTR@1 estimation is the ridge regression by using $\{c_i\}$'s as targets: we seek the optimal parameters that minimize a regularized square loss:

$$f_t(\boldsymbol{\beta}, \{b_{q,u}\}) \stackrel{\text{def}}{=} \sum_{i=1}^{t} \left(c_i - \boldsymbol{\beta}^\top \mathbf{x}_i - b_{q_i u_i}\right)^2 + \lambda_1 \left\|\boldsymbol{\beta} - \boldsymbol{\beta}^{(0)}\right\|_2^2 + \lambda_2 \sum_{(q,u)\in\mathcal{P}} \left\|b_{qu} - b_{qu}^{(0)}\right\|_2^2, \quad (2)$$

where λ_1 and λ_2 are positive regularization parameters provided by users, $\boldsymbol{\beta}^{(0)}$ and $b_{qu}^{(0)}$ are the prior values, and $\|\cdot\|_2$ is the ordinary ℓ_2-norm. In practice, values of λ_1 and λ_2 are often determined by cross-validation on the training data, while $b_{qu}^{(0)}$ are often set to zero to favor simple models. As usual, regularization is applied to avoid overfitting and to ensure numerical stability. In our specific application, it also has desired consequences as explained in the previous subsection.

Since (2) is a least-squares problem with $d + N$ many parameters, one may think it is intractable to solve for the exact solution since the computation complexity is $O((d+N)^3)$ and N is often very large. Fortunately, using matrix algebra, we can derive a closed-form solution for the minimizer of (2), whose complexity is cubic in d and only linear in N.

Specifically, we partition the index set $\{1, 2, \ldots, t\}$ into $I_1 \cup I_2 \cup \cdots \cup I_N$, so that I_j contains indices in $\mathcal{D}$ that corresponds to the j-th distinct (q, u) pair. For every $j \in \{1, 2, \ldots, N\}$, we define the following quantities,

$$\begin{aligned} a_j &\stackrel{\text{def}}{=} \lambda_2 + |I_j|, \\ \mathbf{b}_j &\stackrel{\text{def}}{=} \sum_{i\in I_j} \mathbf{x}_i, \\ d_j &\stackrel{\text{def}}{=} \lambda_2 b^{(0)} + \sum_{i\in I_j} c_i. \end{aligned}$$

In addition, we define

$$\mathbf{A}_0 \stackrel{\text{def}}{=} \lambda_1 \mathbf{I} + \sum_{i=1}^{t} \mathbf{x}_i \mathbf{x}_i^\top,$$
$$\mathbf{d}_0 \stackrel{\text{def}}{=} \lambda_1 \boldsymbol{\beta}^{(0)} + \sum_{i=1}^{t} c_i \mathbf{x}_i.$$

Now the optimal solution to the least-squares problem must satisfy the first-order optimality condition:

$$\frac{\partial f_t(\boldsymbol{\beta}, \{b_{q,u}\})}{\partial \boldsymbol{\beta}} = \mathbf{0},$$
$$\frac{\partial f_t(\boldsymbol{\beta}, \{b_{q,u}\})}{\partial b_j} = 0,$$

for each $j \in \{1, \ldots, N\}$. Solving this system of linear equations immediately gives the regularized least-squares solution:

$$\boldsymbol{\beta}^* = \left(\mathbf{A}_0 - \sum_{j=1}^{N} a_j^{-1} \mathbf{b}_j \mathbf{b}_j^\top\right)^{-1} \left(\mathbf{d}_0 - \sum_{j=1}^{N} a_j^{-1} d_j \mathbf{b}_j\right) \tag{3}$$
$$b_j^* = a_j^{-1} \left(d_j - \mathbf{b}_j^\top \boldsymbol{\beta}^*\right), \quad \text{for each } j \in \{1, 2, \ldots, N\}. \tag{4}$$

In other words, the complexity of solving the least-squares problem now becomes $O(d^3 + dN)$, a substantial improvement over the $O((d + N)^3)$ complexity of the naive approach.

5.3.2. Online Parameter Updates. More importantly, these formulas suggest that we only need to maintain a set of sufficient statistics ($\mathbf{A}_0$, $\mathbf{d}_0$, a_j, $\mathbf{b}_j$, and d_j) to obtain the exact solution when a new data is added to the set $\mathcal{D}$, without the need to recomputing all quantities.

When a new example $(q_{t+1}, u_{t+1}, c_{t+1})$ arrives, all these sufficient statistics can be updated efficiently in an incremental fashion. In particular, let j_{t+1} be the index of (q_{t+1}, u_{t+1}) in $\mathcal{P}$, then $O(d^2)$ time is needed for the updates:

$$\begin{aligned}
\mathbf{A}_0 &\leftarrow \mathbf{A}_0 + \mathbf{x}_{t+1} \mathbf{x}_{t+1}^\top \\
\mathbf{d}_0 &\leftarrow \mathbf{d}_0 + c_{t+1} \mathbf{x}_{t+1} \\
a_{j_{t+1}} &\leftarrow a_{j_{t+1}} + 1 \\
\mathbf{b}_{j_{t+1}} &\leftarrow \mathbf{b}_{j_{t+1}} + \mathbf{x}_{t+1} \\
d_{j_{t+1}} &\leftarrow d_{j_{t+1}} + c_{t+1}.
\end{aligned}$$

With these updated sufficient statistics, we can now apply Equations (3) and (4) to compute the exact solutions, which again requires $O(d^3 + dN)$ time. However, there are acceleration techniques that can reduce the complexity to $O(d^2)$ and even $O(d)$, as explained in the next section.

5.3.3. Implementation Issues in Practice. While the update rules we have derived are reasonably efficient, we would still like greater acceleration for large d and large N, so that the response time of the whole reranking system can be further reduced.

First of all, the most time-consuming part is the inversion of the matrix in Equation (3), which takes $O(d^3)$ time. Fortunately, since every new data results in a rank-one update on the matrix, $\mathbf{A}_0 - \sum_{j=1}^{N} a_j^{-1}\mathbf{b}_j\mathbf{b}_j^\top$, a straightforward variant of the famous Sherman-Morrison formula may be applied to reduce the complexity to $O(d^2)$.

Second, we may also ignore off-diagonal elements of the matrix, $\mathbf{A}_0 - \sum_{j=1}^{N} a_j^{-1}\mathbf{b}_j\mathbf{b}_j^\top$, and so inversion can be done very efficiently in $O(d)$ time. According to our experience (not reported in the present article), this approximation is quite effective, yielding a good trade-off between solution quality and time requirement.

Third, we note that it is unnecessary to update all N bias terms b_j^* every time a new example arrives. In fact, these bias terms can be updated independently, provided that $\boldsymbol{\beta}^*$ is given. Therefore, we may delay their updates until the moment they are used. Specifically, for a new example $(q_{t+1}, u_{t+1}, c_{t+1})$, we may only update $b_{j_{t+1}}^*$, where j_{t+1} is the index of (q_{t+1}, u_{t+1}) in $\mathcal{P}$. This lazy-update trick completely removes the time dependency on N, a significant improvement when N is large.

Finally, as mentioned in Section 5.2, it may be impossible and unnecessary to explicitly maintain a bias term for every (q, u) pair, since only a small fraction of them are popular queries and thus are expected to take advantage of those bias terms. While the regularization helps as previously mentioned, we can also take advantage of a few techniques such as the hashing trick [Langford et al. 2007] to limit the effective number of bias terms.

5.4. Model Variations

Given the model form and update formula in Section 5.2 and Section 5.3, there are a couple of choices to try for our online reranking function, which we describe as follows.

Exploration and ϵ-greedy. In Section 5.3, we did not describe how the click feedback for the data $\mathcal{D}$ is collected. In order to explore rankings other than the output of our reranking model and collect balanced click feedback in our data set $\mathcal{D}$, we use ϵ-greedy strategy. The ϵ-greedy is a simple strategy to handle the explore-exploit tradeoff described in Section 5.1. It collects the feedback from the randomly permuted ranking with probability ϵ and from the reranked result by the function (1) with probability $1-\epsilon$. Thus, by controlling ϵ, we can balance the exploration and exploitation for our online learning, and our exploration bucket data enables us to realize this strategy. More details on the methodology of using our exploration bucket data are described in the next section.

Warm start. When we are sequentially learning $(\boldsymbol{\beta}, \{b_{qu}\})$ as described in Section 5.3, we need not learn them from scratch solely based on online learning (which is known as *cold-start*), but learn a starting point from some already available click logs (i.e., *warm-start*). The effectiveness of such warm-start models could be critical in terms of improving the performance and learning speed of our reranking function as presented in the next section.

Using clicks on multiple positions. In Section 5.3, we inherently assumed that the click feedback $\{c_i\}$'s are the ones received by the user when the document was displayed in the first position for the query, since we used them as a target for our CTR@1 function in (2). However, although our goal is maximizing CTR@1, we may not limit ourselves to use the click feedback only on position 1, that is, $c_{t,1}$ in the $r_t = \{c_{t,1}, \ldots, c_{t,s}\}$ defined in Section 5.1, but use clicks on multiple positions for learning our reranking function. In that case, we can enlarge the data set $\mathcal{D}$ to $\{(q_i, u_{i,p}, c_{i,p})\}_{i=1,2,\ldots,t}^{p=1,\ldots,s}$, and we

introduce additional bias terms $\{b_p\}_{p=1}^{s}$ to correct the positional biases in the click feedback on position p. Then, we model CTR@p as

$$\text{CTR@}p(q,u) \;=\; \text{CTR@1}(q,u) + b_{\,p} \tag{5}$$

while modifying the loss function as

$$f_t(\boldsymbol{\beta}, \{b_{qu}\}, \{b_p\}) \stackrel{\text{def}}{=} \sum_{i=1}^{t}\sum_{p=1}^{s}\left(c_{i,p} - \boldsymbol{\beta}^{\top}\mathbf{x}_{i,p} - b_{\,q_{i,p}u_{i,p}} - b_p\right)^2$$
$$+\;\lambda_1\left\|\boldsymbol{\beta} - \boldsymbol{\beta}^{(0)}\right\|_2^2 + \lambda_2\sum_{(q,u)\in\mathcal{P}_t}\left\|b_{qu} - b^{(0)}\right\|_2^2 + \lambda_3\sum_{p=2}^{s}\left\|b_p - b_p^{(0)}\right\|_2^2, \tag{6}$$

where λ_3 and $b_p^{(0)}$ are regularization coefficient and prior for the positional bias terms $\{b_p\}$. Note that we set $b_p = 0$ when $p = 1$, and our reranking function is still CTR@1$(q,u) = \boldsymbol{\beta}^T\mathbf{x}_{qu} + b_{qu}$ learned by minimizing (6). In this way, we can utilize more click feedback than only using the clicks on position 1 to learn the reranking function (1). In Section 6, we will show how useful this approach is for building the warm start model described before. For the online updates, however, in order to control the number of experiments to compare, we remain to use only the clicks at the first positions and use the loss function and update formula in Section 5.3 for all of the online schemes in our experiments.

6. EXPERIMENTAL RESULTS

This section reports our experiments on various algorithms for recency search reranking using the exploration bucket data described in Section 3. Section 6.1 describes an unbiased offline evaluation method we will adopt in the experiments. Section 6.2 describes a number of representative algorithms for comparison. These algorithms are selected to demonstrate benefits of various algorithmic choices described in Section 5.4. Section 6.3 presents and analyzes the experiment results in details. Finally, Section 6.4 revisits the query examined in Section 4, illustrating how our algorithm adapts to user click feedback to rerank the top documents and yield better results.

6.1. Unbiased Offline Evaluation

A tricky part of our problem is that, unlike in supervised learning, it is hard to evaluate and compare performance of algorithms using a static set of log data. The reason is that the click feedback in the log depends on the ranking results that the user observed when the log was collected; consequently, we do not know what that user might have clicked if the algorithm we evaluate ranked the results differently. Fortunately, our exploration bucket data can be used for reliable offline evaluation of different algorithms, including both batch or online ones.

We follow the "replaying" evaluation method studied by Li et al. [2011] for interactive applications like the reranking problem considered here. First, we hold out the sessions for the latter three days in the exploration bucket data and use it as a *test* set. The first three days' data may be used as a training set for batch learning or warm start model described in Section 5.4. We then sort the test sessions in the order of time stamps. To evaluate an algorithm's CTR@1 on the test set, we maintain two

quantities, C and M, which are interpreted as the number of clicks at position 1 and number of search sessions, respectively. Both C and M are initialized to 0.

(1) We retrieve the t-th session in the test set, present the top s documents together with their features to the reranking function.
(2) The reranking algorithm then proposes to display one of the documents in the first position based on its reranking scores. We call it a "match" if this proposed document is the same as the one displayed in the first position in the retrieved test session.
(3) If a match happens, we reveal the user feedback c_t (1 for click and 0 otherwise) to the algorithm, and perform the updates: $C \leftarrow C + c_t$ and $M \leftarrow M + 1$.
(4) Otherwise, c_t is not revealed, and the values of C and M are unchanged. Effectively, this session is ignored.

Finally, the overall CTR@1 of the algorithm in this evaluation process is C/M.

For each session in our test set, the probability that a match happens is $1/s$ for any ranking algorithm, since the top s documents are randomly shuffled in our exploration bucket data. Therefore, for a test set of L sessions, M equals L/s on average. In our experiments, since L is large, M is almost constant across different runs. The following key property justifies the soundness of the given evaluation method: it can be proved that the estimated CTR@1, C/M, of an online algorithm is an unbiased estimate of its true CTR@1 as if we were able to run it to serve live user traffic [Li et al. 2011, Theorem 2]. Therefore, algorithms that have higher CTR@1 estimates using this evaluation method will have higher CTR@1 in live buckets as well. This important fact allows us to reliably compare and evaluate various algorithms offline without the costs and risks of actually testing them with live users.

6.2. Models

There are various options to leverage user click feedback to adjust a reranking function. For instance, one may expect better adaptation to user interest if a reranking system can adjust its ranking function in real time based on user feedback; it may also be interested in understanding how reranking performance is affected by the CTR model, such as the ability to generalize (via the linear features) and specialize (via the bias terms) in our model (1).

In the following, we describe a few representatives, chosen carefully to demonstrate the benefits of various algorithmic choices. The models are grouped into four categories. When there is **@4** symbol in the model name, it means that the data $\mathcal{D}$ used for the model update consists of click feedback from top 4 positions in the click log. Otherwise, by default, $\mathcal{D}$ for the following models consists of click feedback only from the first position.

(1) The first is a baseline that is based entirely on editorial judgments and does not leverage user clicks at all.
   — *frmsc(baseline)*. We used the recency ranking function [Dong et al. 2010a] deployed in our search engine as a baseline. This function was trained using time-varying recency features and recency demoted labels provided by human editors. This method does not use click feedback.
(2) The second category contains methods that learn CTR@1 from the first three days' training data (as specified in Section 6.1), and then, do not online update in the test phase. Following three batch methods will be compared to their online-learning counterpart.
   — *batch(b)*. This is the linear model in (1) trained on the training set, and then deployed on the test set without any online updates. Note that there is no

positional biases in the click feedback in the training set for this model due to the exploration bucket.

— *batch(nb)*. This is the same as before, but does not use the bias terms in (1). In other words, only a linear combination of features is used to compute a CTR@1 estimate. This model is used to show the benefits of the bias terms.

— *batch@4(counting)*. This scheme sets $\boldsymbol{\beta} = \mathbf{0}$ and only train the bias terms from the training set using click feedback from all 4 positions. That is, this is equivalent to simply computing the ratio of cumulative clicks and views for each query-document in the training set separately. Since this scheme is not using any features, when it sees a new query-document pair in the test set, it does not have any information for the pair to do the reranking; in this case, it switches to *frmsc(baseline)*. Moreover, note that this scheme uses clicks from all four positions, so the size of data it uses is four times larger than these two batch models. Furthermore, we do not utilize the position features described in Section 5.4 since the scheme is based on the exploration bucket and the position biases will be removed as shown in Section 3.

(3) The third category contains online learning methods in Section 5.3 for reranking with ϵ-greedy strategy mentioned in Section 5.4. We realize the ϵ-greedy strategy in our online learning method by utilizing the exploration bucket data again. That is, while we use the exploration bucket data for an unbiased evaluation of performances of various schemes as in Section 6.1, we use the data once more to incrementally train the online schemes, as in Section 5.3. More concretely, at time t, the click feedback at the first position $c_{t,1}$ is revealed to the online schemes for the model updates, no matter whether there is a "match" or not for the schemes so that the reranking function can observe the feedbacks for all possible randomly served documents in the top position to correctly learn the reranking based on CTR@1. Note that this is effectively simulating the ϵ-greedy strategy with $\epsilon = 1$ and a separate deployment test bucket for evaluation. Ideally, to fully realize the ϵ-greedy strategy rigorously, we need to set $\epsilon < 1$ and also learn from the exploitation feedback of the reranking schemes. However, implementing the procedure would reduce the data size for learning significantly (since we need to simulate the exploitation via "matching" of sessions, and it will cause large variance in our results). Thus, in our experiment with the online learning method will learn solely from the random exploration of rankings. Note that when our online schemes get deployed in the live system, then there would be no issue with realizing ϵ-greedy with $\epsilon < 1$. Also, a clear but subtle point is that we are revealing the click feedback after the data point was used for the "replay" evaluation so that we are not training and testing with the same data point.

Moreover, in practice, we note there is usually a time delay between delivery of the ranking result and the receipt of user feedback. To make our evaluation and online learning process closer to reality, we do not reveal user feedback $c_{t,1}$ to the reranking algorithm immediately. Rather, these signals are revealed every five minutes (based on the time stamps of the test sessions) for our simulation. Figure 5 summarizes the whole procedure of batch training for warm start, online update, and testing using our exploration bucket data.

Based on a few algorithmic choices, we tested following variations to see the effect of online schemes. Similarly as in the previous batch models, *online@4(counting,ws)* uses clicks from all four positions, and the rest only uses clicks from the first positions.

— *online(b)* and *online(nb)*. These are the online algorithms that optimize the parameters in (1) incrementally based on user click feedback, with and without

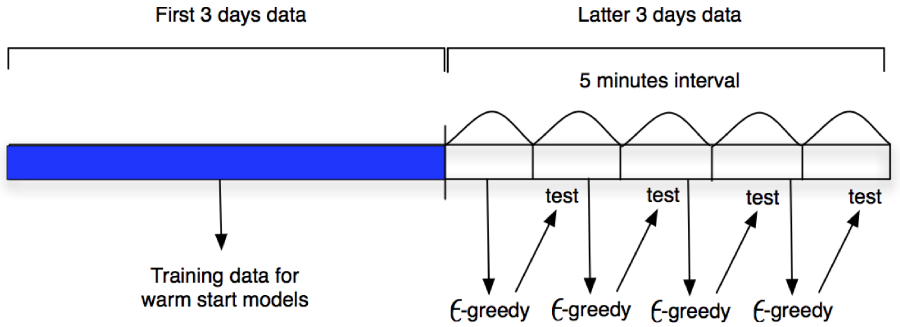


Fig. 5. Summary of the usage of the exploration bucket data for batch training, online update, and testing. ϵ was set to 1 in our experiments.

bias terms, respectively. Note that both algorithms learn the parameters $\boldsymbol{\beta}$ *from scratch*.

— *online(b,ws)* and *online(nb,ws)*. These are the same as *online(b)* and *online(nb)* but use warm-start initialization of the parameters. Specifically, we used the batch-learned parameters (in *batch(b)* and *batch(nb)*, respectively) as $\boldsymbol{\beta}^{(0)}$ and $b_{qu}^{(0)}$ in (1). These methods thus combine the prior knowledge extracted from previous data with the ability to learn online. While the regularization parameters λ_1 and λ_2 can be set by cross validation, we set the parameters to 10 based on our previous experience in related problems (such as in experiments of the cited reference [Li et al. 2010]), in which values between 1 and 10 seemed to work stably and reasonably well.[4]

— *online(b,ws,w0)*. This method is similar to *online(b,ws)* except that the weight vector $\boldsymbol{\beta}$ is fixed to the warm-start $\boldsymbol{\beta}^{(0)}$ learned by *batch(b)*. Thus, this model performs limited online updates and is useful to demonstrate the benefits of online update of $\boldsymbol{\beta}$.

— *online@4(counting, ws)*. This scheme does the warm start from the bias terms learned in *batch@4(counting)* and continues the learning, or equivalently, counting clicks and views, in the test set using click feedback from all 4 positions. This scheme is also using four times more data than these online schemes. Again, since this scheme does not utilize query–document features for generalization, it may still suffer from the "cold-start" problem on new or tail queries.

(4) Batch learning of (warm-start) parameters in the previous two categories is trained on the first three days of exploration bucket data. However, although the exploration bucket gives the unbiased CTR@1 of the documents as in Section 4, in practice, it is not realistic to always require such expensive data in order to build the batch model as a starting point for online models. Therefore, we use the *controlled log* from nonexploration buckets (e.g., production bucket) for the same period of time to build the batch models and compare them with the models built from the exploration bucket. Note that the controlled log is very cheap to attain, but have large positional biases in clicks, so it is not clear to see how well the batch models trained on the controlled logs would perform. We include following three variations regarding the batch model with the controlled log, which contained 485, 135

[4]As a sanity check, we tried varying the parameters in a grid of parameters in a neighboring range, and found the performance to be quite insensitive (less than 0.6% relative difference).

Table I. Overall Cumulative nCTR@1 on the Test Set

algorithms	nCTR@1	lift over *frmsc(baseline)*
frmsc(baseline)	0.770	0%
batch(b)	0.877	13.90%
batch(nb)	0.849	10.26%
batch@4(counting)	0.854	10.91%
online(b)	0.875	13.64%
online(nb)	0.839	8.96%
online(b,ws)	**0.901**	**17.01%**
online(nb,ws)	0.851	10.52%
online(b,ws,w0)	0.891	15.71%
online@4(counting)	0.872	13.25%
batch@1(control)	0.883	14.68%
batch@4(control,np)	0.856	11.17%
batch@4(control)	0.885	14.94%

sessions from production logs that overlaps with the first three days of exploration bucket.[5]

— *batch(control)*. This method learns the model in (1) using the clicks at the first positions in the controlled log.
— *batch@4(control,np)*. This method learns the model with the loss function in (6) and using top four positions' clicks in the controlled log. However, it ignores the position biases, $\{b_p\}$'s are all set to zero, so data from all four positions are not distinguished.
— *batch@4(control)*. This method improves on *batch@4(control,np)* by considering position biases and including nonzero $\{b_p\}$'s as described in Section 5.4.

6.3. Experimental Results

We ran the algorithms described in the previous subsection, whose overall nCTR@1 results are summarized in Table I. The lifts over the baseline's nCTR@1 are also computed.

To visualize how instantaneous nCTR@1 evolves over time, we also computed aggregated clicks in every 6-hour period for the algorithms. Only five algorithms are included in Figure 6 to ensure legibility.

6.3.1. Results Analysis. A number of important observations are in order based on the results reported in Table I and Figure 6.

(1) The advantage of leveraging user click feedback is obvious from the lift of all algorithms over the baseline that does not use click feedback at all. The click lift, which ranges from 8.96% to 17.01%, is statistically significant given the size of our data set. Even the batch-learning methods that do not perform online updates are quite strong, capable of achieving at least 10% lift.
(2) We can see the additional benefits brought by the bias terms in both batch and online algorithms. It is true in all cases that an algorithm is better than its counterpart without bias terms. In the case between *online(b,ws)* and *online(nb,ws)*, the bias terms account for about 6.49% lift.
(3) While batch algorithms have quite strong lift, greater lifts are achieved by algorithms that adjust their reranking functions online. This benefit is expected since

[5]Note that the online methods described here may also be combined with warm-start models learned from control logs. We do not include them for comparison in the article to simplify the presentation of results.

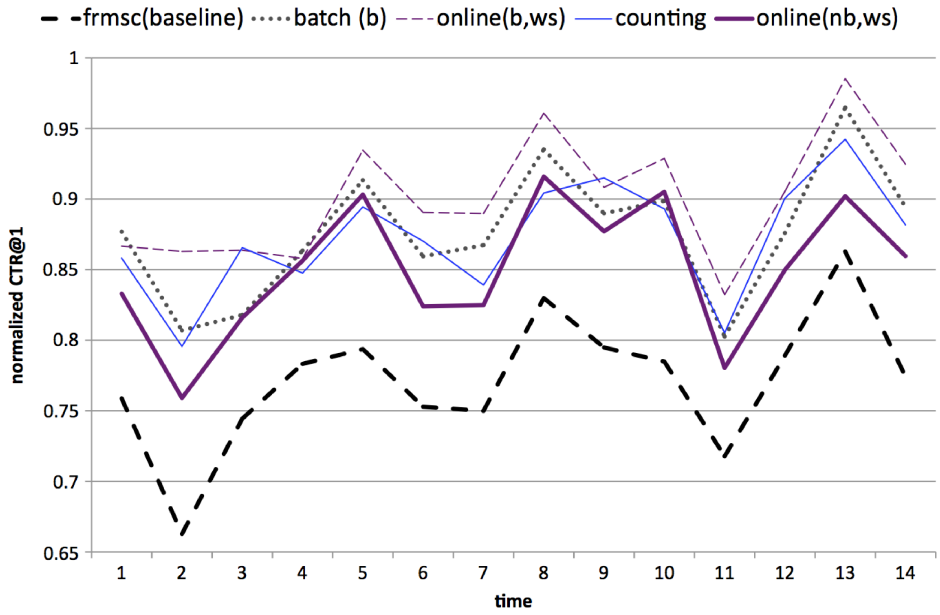


Fig. 6. Instantaneous (normalized) CTR@1 of the four models on the test data. Each point was measured for an approximately six-hour period. The *counting* scheme in this plot refers to *online@4(counting)* scheme.

the online algorithms are able to extract more information from online click feedback, in addition to the batch-learned models. In addition, it is worth pointing out a practically important fact that it is compatible to use batch-learned models as warm-start models for online methods. Of all the algorithms in Table I, the greatest lift is achieved by *online(b,ws)*—the online method that uses batch-learned warm-start models and bias terms. Even for the coefficient vector $\boldsymbol{\beta}$ that is shared by all query-document pairs, updating it online is still helpful, which is justified by the gap between *online(b,ws)* and *online(b,ws,w0)*. A larger gap is possible if the time span of our test data is larger.

(4) We examine the role of generalization in our CTR@1 model. As discussed earlier, the bias terms are essentially zero except for popular query-document pairs, due to the regularization we used in the optimization step. Therefore, the linear part in (1) determines the CTR@1 estimates of tail queries for which we observed one or few sessions. The result in Table I confirms our conjecture: the method *online@4(counting)* which uses the bias terms alone for reranking yielded a lower click lift than *online(b,ws,w0)* or *online(b,ws)*. The reason is that the CTR model in *online@4(counting)* cannot make good prediction in "cold-start" situation and so little lift was achieved on the tailed queries. Moreover, note that *online@4(counting)* is using three times more data than the others, which emphasizes the significance of the feature-based generalization.

(5) Our results suggest it is possible to use control log to build a competitive warm-start model. It should be noted that the control log has much more data than the exploration data, thus the strong performance. However, position bias has to be considered when clicks from multiple positions are used, as demonstrated by the gap between *batch@4(control)* and *batch@4(control,np)*. We believe the performance of learning from controlled logs could be further improved by advanced click models. We plan to investigate this direction in future work.

6.3.2. Discussion on the Relations with Click Models and Click-Based Reranking. We deliberately have not included the results of the possible reranking schemes that apply CTR@1 estimation methods of various click models in the literature, because the focus of our work and those click models is different. That is, as mentioned in Section 2.2, one of the main motivations for studying click models is to remove positional biases of the clicks in the *controlled* log and obtain unbiased estimate of *static* CTR@1 of each query-document pair. Moreover, most of the click models operate in a batch mode, which requires significant time and complexity for collecting the data and training the model. In contrast, in our work, the focus is to show the effectiveness of online learning framework that quickly updates the model and tracks the CTR@1 variations of documents for recency queries by utilizing the feature based models and the exploration bucket, in which the positional biases are already removed. Therefore, applying the click models in our exploration bucket to obtain CTR@1 estimates for reranking in an online manner would be not only computationally expensive, due to the complex training process, but also not necessary, since the positional biases are already removed in the exploration bucket.

We may still try to apply the click models in our reranking scenario by obtaining the CTR@1 estimates from the controlled logs and use the estimates for reranking in the test set. However, we can expect that this scheme would not perform much better than *batch@4(counting)* in Table I, since *batch@4(counting)* is estimating the CTR@1 from the exploration bucket in which the positional biases are already removed, although the exploration bucket data is smaller than the controlled log data. For the comparison purpose and sanity check, we have implemented several click models, for instance, Dynamic Bayesian Network (DBN) model [Chapelle and Zhang 2009], Cumulated Relevance model [Dupret and Liao 2010], and COEC model [Zhang and Jones 2007], on the previously described controlled log, and as expected, their test results were similar to or slightly worse than *batch@4(counting)*. We have not included the numbers in order to prevent from causing unnecessary confusion to the readers.

From these intuitions and the strong generalization performances of the feature based models in our experiment, one may also conceive of online reranking schemes for recency queries that combine feature-based model with click models as in Zhu et al. [2010], which would be an interesting topic for future research. Again, the main purpose of this article is to propose a framework for online reranking method in recency search problem, not to find the best click models to estimate CTR@1, thus our findings can be applied to those future work as well.

6.3.3. Lift Distribution. We now examine lift distribution over queries with different impression and lengths, respectively. Results of two representatives, *batch(b)* and *online(b,ws)*, are reported.

Figure 7 presents CTR@1 lifts over *frmsc* aggregated over queries with different impressions. We notice that the *online(b,ws)* model is doing very well on popular recency queries, whereas the *batch(b)* model gives more lift on queries with less than 2 impressions. For queries with very limited impressions, for instance, less than 2, the *online(b,ws)* model cannot gain much advantage over the *batch(b)* model. This problem might be mitigated when applying the *online(b,ws)* model to larger traffic. This observation also suggests a practical solution, that is, employing *batch(b)* models for queries with scarce impressions while using *online(b,ws)* models for popular recency queries only. It is expected that the *online(b,ws)* model achieves much more lift on popular recency queries, since the *batch(b)* model cannot specialize well on such cases despite the relatively large number of impressions. For example, the query "giant squids in California" was a popular recency query. A *batch(b)* model well trained on historical click events still fails to foresee the popularity and high relevance of the youtube video,

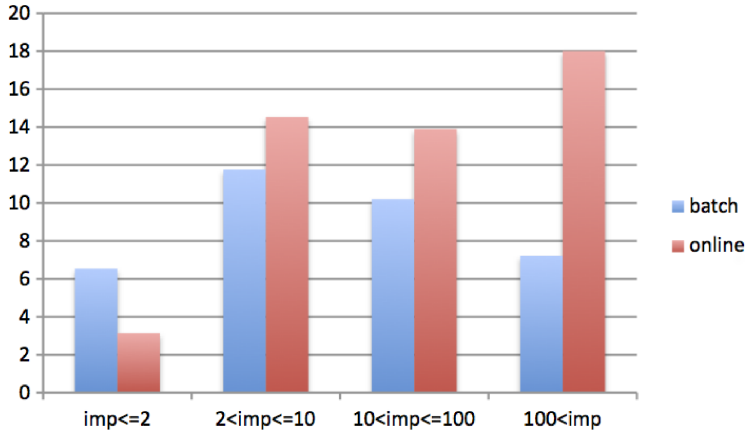


Fig. 7. Relative CTR@1 lift in percentage (%) over the baseline model (frmsc) for queries with different impressions.

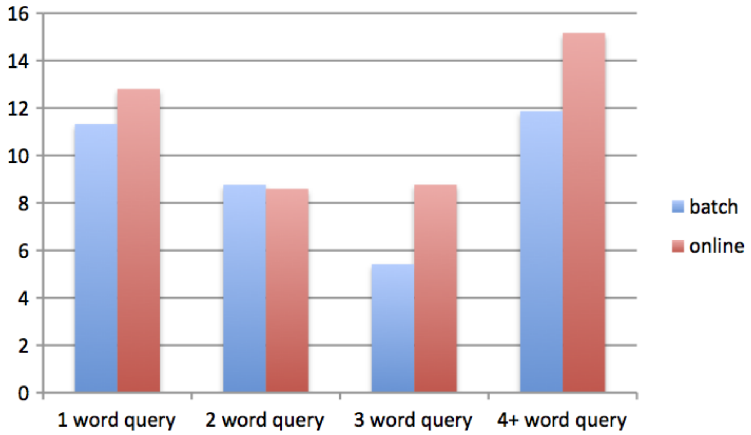


Fig. 8. Relative CTR@1 lift in percentage (%) over the baseline model (frmsc) for queries with different lengths.

whereas the *online(b,ws)* model does so correctly by adapting to users' click feedback; see Section 6.4 for more details of this example.

Figure 8 presents CTR@1 lifts over *frmsc* for queries with different lengths. Except for a tie in two-word queries, the *online(b,ws)* model consistently outperforms the *batch(b)* model. The results suggest the online reranking method is robust to queries of various lengths.

6.3.4. Comparison of Other Click Metrics. Although we have focused on training and evaluating our online reranking function based on CTR@1, we also compare our results with following other click metrics for ranking proposed in Radlinski et al. [2008].

— *Query CTR* is the average number of clicks for each query
— 1–*Abandonment Rate* is the probability of a session receiving a click
— *Max RR (Reciprocal Rank)* is the reciprocal rank of the highest ranked result clicked on
— *Mean RR (Reciprocal Rank)* is the average of clicked documents' reciprocal ranks

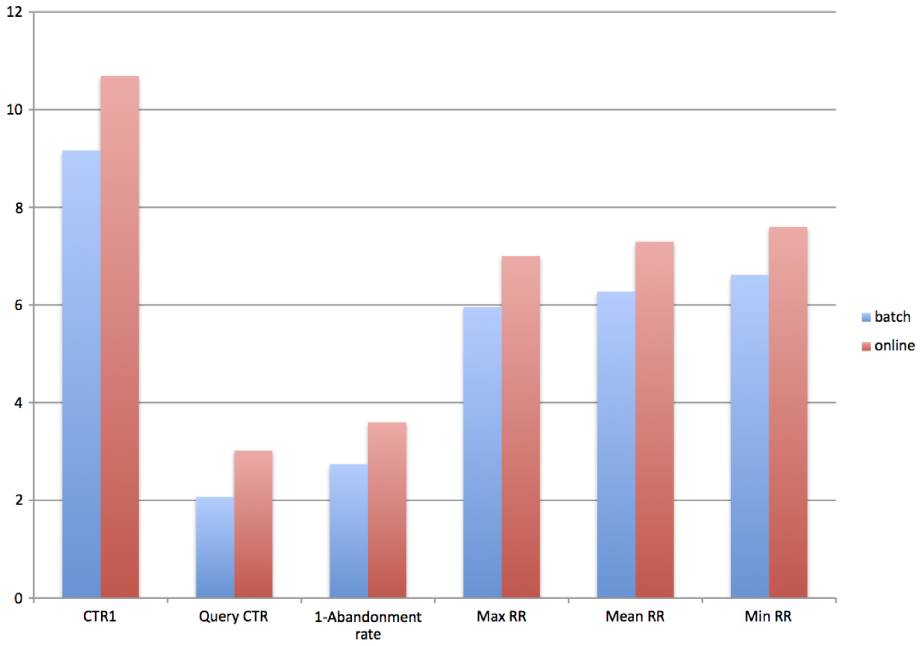


Fig. 9. Relative lift in percentage (%) over the baseline model (frmsc) on various click metrics.

—*Min RR (Reciprocal Rank)* is the reciprocal rank of the lowest ranked results clicked on.

Thus, for all metrics, higher values are assumed to indicate better ranking qualities.

In Figure 9, both *batch* and *online* models show significant lifts over the *frmsc* baseline on all click metrics listed before. Moreover, the *online(b,ws)* method consistently gives about 2% more lift than *batch(b)*. Thus, although our algorithms focus on maximizing CTR@1, it also gives simultaneous lifts on other click metrics as well. This fact shows the easily measurable CTR@1 is a good surrogate to optimize, and also justifies our choice of using click at the top position as user feedback.

Remark. We have done similar experiments on the nonrecency queries as well, but our online reranking scheme did not show too much gain for those queries as in the recency queries, of which results are omitted here. The absence of improvements for nonrecency queries is expected; the relevance of documents with respect to such queries does not change dramatically over time, so the reranking based on the click feedback may not be too different from the original ranking.

6.3.5. Effect of Model Update Frequency. A critical parameter in our online-learning framework is the update frequency, that is, how often we use click feedback to update the reranking function. Usually, more frequent updates ensure better ability to adapt to relevance changes, but, at the same time, they impose greater engineering requirements on the search engine. It is of practical significance to study how performance of an online-learning algorithm is affected by model update frequencies.

Our results so far have used 5 minutes as the update cycle, which was determined somewhat arbitrarily. Figure 10 plots the cumulative nCTR@1 of *online(b,ws)* with varying model update periods, ranging from 5 minutes to 3 days. As expected, the longer the period, the lower nCTR@1 is achieved due to lack of swift adaptivity. When

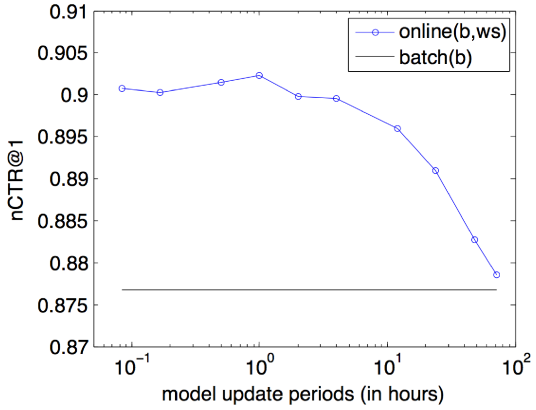


Fig. 10. Cumulative nCTR@1 of *online(b,ws)* with various model update periods. The leftmost result on the blue line is for 5 minutes, and the rightmost for 3 days. The straight line is for *batch(b)* that does not perform any online model updates. It should be noted that our data for online learning contained slightly more than 3 days' data, thus the small difference in the performance of *batch(b)* and *online(b,ws)* with a 3-day model update period.

the model is updated every 2 or 3 days, the nCTR@1 is very close to the *batch(b)* scheme that does not do online updates at all.

An interesting observation is that our method is quite robust to this parameter. Within the range of six hours, model update periods do not affect the performance much. This is also consistent with our intuition that, when a breaking news occurs, a few hours' delay may be good enough for online learning to wait for enough click signals to identify the relevance changes. In general, however, we expect greater benefits of online learning with higher traffic volume.

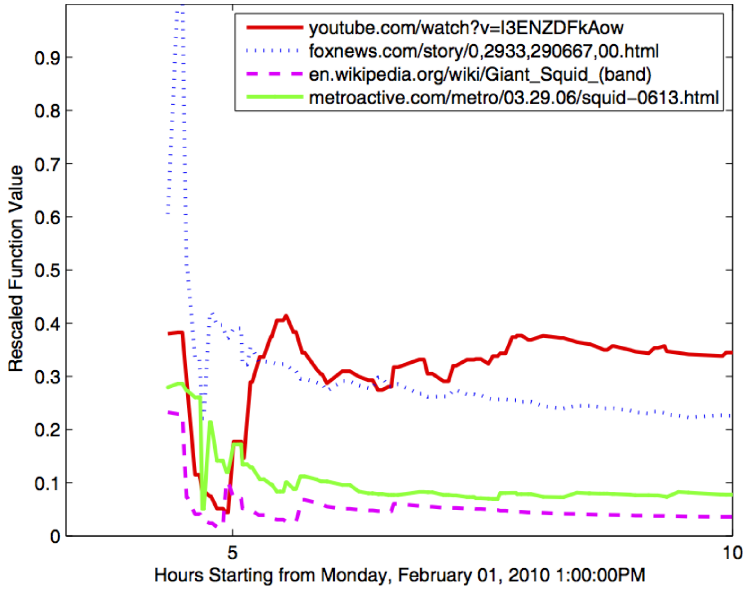
6.4. Case Study

We now revisit our example query "giant squid in California" in Section 4 to illustrate how our online reranking function can adapt to the click feedback quickly and track the best reranking. Coincidentally, the query happened to only appear in our test set. We ran our online model, *online(b,ws)*, on the test set, and recorded the function values of the 4 URLs. Figure 11(a) shows the function values of 4 URLs for the first 10 hours, and Figure 11(b) presents the entire temporal curves for those function values in the lifetime of "*giant squid in California*."

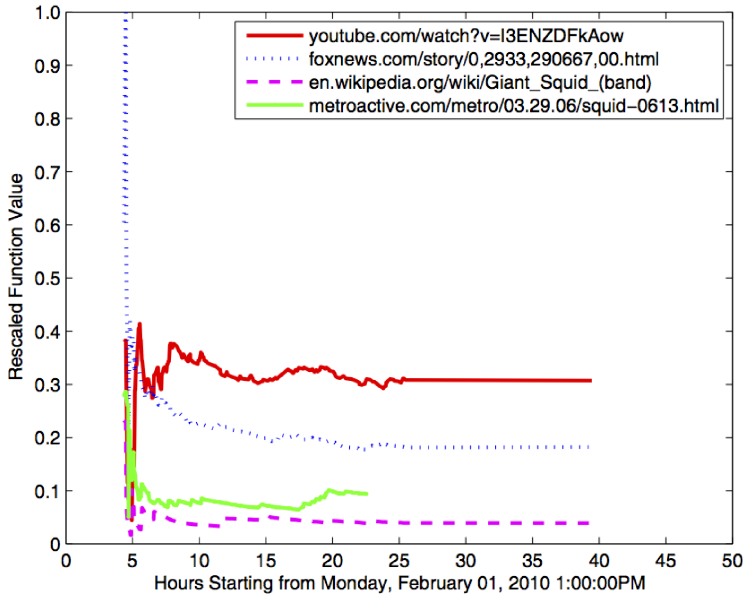
Since the initial reranking function *online(b,ws)* is (almost) identical to the fixed one of *batch(b)*, we can see from Figure 11(a) that when the query appears around the 5th hour, *batch(b)* orders the four URLs as

(1) `foxnews.com/story/0,2933,290667,00.html`
(2) `youtube.com/watch?v=I3ENZDFkAow`
(3) `metroactive.com/metro/03.29.06/squid-0613.html`
(4) `en.wikipedia.org/wiki/Giant_Squid_(band).`

Note that this ranking is different from the *frmsc* ranking presented in Section 4. That is, although the *batch(b)* has not observed the query "*giant squid in California*" in its training set sessions, from the sessions of other queries in the training set, it was able to predict based on the query–document features that the "video" page will attract many clicks for the query and improve the original ranking. Nonetheless, we see that it still fails to accurately predict the users' click behaviors.



(a) Zoom-in plot



(b) Whole period

Fig. 11. Function values in the online model on the 4 URLs associated with the recency query "*giant squid in California*." Note that the URL of metroactive.com was replaced by another URL (not shown here) after the 23rd hour.

On the other hand, given the users' click patterns in Figure 4, the *online(b,ws)* promptly learns from them and put the "video" content with the highest CTR to the top rank within an hour. The ranking was then maintained for the rest of the time. Then, after the 25th hours, when the impression of the query quickly decreased toward 0 as

shown in Figure 3, the function values of *online(b,ws)* were kept intact, as can be seen in Figure 11(b).

Thus, this example indeed illustrates how reranking algorithms may benefit from user click feedback to improve ranking results. Furthermore, by real-time adaption *online(b,ws)* can quickly learn from users' click patterns and outperforms not only the editorial-based batch recency ranking, *frmsc*, but also the click-based batch reranking, *batch(b)*, for recency queries.

7. CONCLUSIONS

In this work, we investigated various learning algorithms to reranking recency search results based on real-time user feedback. Our contributions are threefold. First, our evaluation method is novel for Web search—a random exploration bucket was used to collect user feedback, which not only removed positional bias but also allowed one to reliably evaluate online learning algorithms offline. Second, we proposed a reranking approach to improve current search results for recency queries, and carried out extensive empirical results for a dozen of variants. Third, we demonstrated the need for using online learning as a flexible machine learning paradigm to adapt a ranking system to time-varying document relevance.

In future work, we would like to investigate on how we can combine the feature-based model and click models in the recency search reranking problem. Also, incorporating other metrics from search sessions, such as dwell time of a click, session length, or revenue, into our objective function so that the learned reranking model can optimize for those more sophisticated objective would be another topic to pursue. Finally, in this work, we focused on ranking documents based on individual document's CTR estimate, and it would be more challenging to design algorithms for the best permutation of a set of documents, in which interactions between documents can be taken into account, so that the diversity of the ranking can be taken into account as well.

REFERENCES

AGARWAL, D., CHEN, B., AND ELANGO, P. 2009. Explore/exploit schemes for web content optimization. In *Proceedings of the International Conference on Data Mining*.

AGARWAL, D., CHEN, B., AND ELANGO, P. 2010. Fast online learning through offline initialization for time-sensitive recommendation. In *Proceedings of the ACM SIGKDD International Conference On Knowledge Discovery and Data Mining.*.

BURGES, C., SHAKED, T., RENSHAW, E., LAZIER, A., DEEDS, M., HAMILTON, N., AND HULLDENDER, G. 2005. Learning to rank using gradient descent. In *Proceedings of the International Conference on Machine Learning*.

BURGES, C. J., LE, Q. V., AND RAGNO, R. 2007. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems*, vol. 19, B. Schölkopf, J. Platt, and T. Hofmann Eds.

CHAPELLE, O. AND ZHANG, Y. 2009. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th International World Wide Web Conference*. ACM, 1–10.

CORTES, C., MOHRI, M., AND RASTOGI, A. 2007. Magnitude-preserving ranking algorithms. In *Proceedings of the 24th International Conference on Machine Learning*.

DAI, N. AND DAVISON, B. D. 2010. Freshness matters: In flowers, food and web authority. In *Proceedings of the 3rd ACM SIGIR Conference*.

DONG, A., CHANG, Y., ZHENG, Z., MISHNE, G., BAI, J., ZHANG, R., BUCHNER, K., LIAO, C., AND DIAZ, F. 2010a. Towards recency ranking in web search. In *Proceedings of the 3rd International ACM Conference on Web Search and Data Mining*.

DONG, A., ZHANG, R., KOLARI, P., BAI, J., DIAZ, F., CHANG, Y., AND ZHENG, Z. 2010b. Time is of the essence: Improving recency ranking using twitter data. In *Proceedings of the 19th International ACM Conference on World Wide Web*.

DUPRET, G. AND LIAO, C. 2010. Cumulated relevance: A model to estimate document relevance from the click-through logs of a web search engine. In *Proceedings of the 3rd International ACM Conference on Web Search and Data Mining*.

ELSAS, J. AND DUMAIS, S. 2010. Leveraging temporal dynamics of document content in relevance ranking. In *Proceedings of the 3rd International Conference on Web Search and Data Mining*.

FREUND, Y., IYER, R., SCHAPIRE, R., AND SINGER, Y. 2003. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Resear. 4*, 933–969.

GRAEPEL, T., CANDELA, J. Q., BORCHERT, T., AND HERBRICH, R. 2010. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In *Proceedings of the 27th International Conference on Machine Learning*. 13–20.

GUO, F., LIU, C., KANNAN, A., MINKA, T., TAYLOR, M., WANG, Y., AND FALOUTSOS, C. 2009. Click chain model in web search. In *Proceedings of 18th International World Wide Web Conference*.

INAGAKI, Y., SADAGOPAN, N., DUPRET, G., LIAO, C., DONG, A., CHANG, Y., AND ZHENG, Z. 2010. Session based click features for recency ranking. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*.

JÄRVELIN, K. AND KEKÄLÄINEN, J. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst. 20*.

JI, S., ZHAO, K., LIAO, C., ZHENG, Z., XUE, G., CHAPELLE, O., SUN, G., AND ZHA, H. 2009. Global ranking by exploiting user clicks. In *Proceedings of the 28th International ACM SIGIR Conference*. 35–42.

JOACHIMS, T. 2002a. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, New York, NY, 133–142.

JOACHIMS, T. 2002b. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*. ACM.

JONES, R. AND DIAZ, F. 2007. Temporal profiles of queries. *ACM Trans. Inf. Syst. 25*.

KANG, C., WANG, X., CHEN, J., LIAO, C., CHANG, Y., TSENG, B., AND ZHENG, Z. 2011. Learning to re-rank web search results with multiple pairwise features. In *Proceedings of the 4th International Conference on Web Search and Web Data Mining*.

KOREN, Y. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

KULKARNI, A., TEEVAN, J., SVORE, K. M., AND DUMAIS, S. 2011. Understanding temporal query dynamics. In *Proceedings of the 4th International Conference on Web Search and Web Data Mining*.

LANGFORD, J., LI, L., AND STREHL, A. 2007. Vowpal wabbit online learning project. http://hunch.net/?p=309.

LI, L., CHU, W., LANGFORD, J., AND SCHAPIRE, R. 2010. A contextual bandit approach to personalized news article recommendation. In *Proceedings of 19th World Wide Web Conference*.

LI, L., CHU, W., LANGFORD, J., AND WANG, X. 2011. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the 4th International Conference on Web Search and Web Data Mining*.

LIU, C., LI, M., AND WANG, Y.-M. 2009. Post-rank reordering: Resolving preference misalignments between search engines and end users. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*.

LIU, T. Y. 2009. *Learning to Rank for Information Retrieval*. Now Publishers.

MOON, T., LI, L., CHU, W., LIAO, C., ZHENG, Z., AND CHANG, Y. 2010. Online learning for recency search ranking using real-time user feedback. In *Proceedings of the 19th International Conference on Knowledge Management*.

RADLINSKI, F. AND JOACHIMS, T. 2007. Active exploration for learning rankings from click-through data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

RADLINSKI, F., KURUP, M., AND JOACHIMS, T. 2008. How does click-through data reflect retrieval quality? In *Proceedings of 17th ACM Conference on Information and Knowledge Management*.

ROBBINS, H. 1952. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc. 58*, 5, 527–535.

STERN, D., HERBRICH, R., AND GRAEPEL, T. 2009. Matchbox: Large scale online bayesian recommendations. In *Proceedings of the 18th International World Wide Web Conference*.

YUE, Y. AND JOACHIMS, T. 2009. Interactively optimizing information retrieval systems as a dueling bandits problem. In *Proceedings of the 26th International Conference on Machine Learning*.

ZHANG, V. AND JONES, R. 2007. Comparing click logs and editorial labels for training query rewriting. In *Query Log Analysis: Social and Technological Challenges. A Workshop at the 16th International World Wide Web Conference*.

ZHENG, Z., ZHA, H., ZHANG, T., CHAPELLE, O., AND CHEN, K. 2008. A general boosting method and its application to learning ranking functions for web search. In *Advances in Neural Information Processing Systems*, vol. 20. MIT Press, Cambridge, MA.

ZHU, Z. A., CHEN, W., MINKA, T., ZHU, C., AND CHEN, Z. 2010. A novel click model and its applications to online advertising. In *Proceedings of the 3rd International Conference on Web Search and Web Data Mining*.

Received March 2011; revised September 2011, January 2012; accepted March 2012