Online Learning for Recency Search Ranking Using Real-time User Feedback

Taesup Moon, Lihong Li, Wei Chu
Yahoo! Labs
701 First Ave, Sunnyvale, CA 94089
{taesup, lihong, chuwei}@yahoo-inc.com

Ciya Liao
Microsoft
One Microsoft way, Redmond, WA 98052
cliao@microsoft.com

Zhaohui Zheng, Yi Chang
Yahoo! Labs
701 First Ave, Sunnyvale, CA 94089
{zhaohui, yichang}@yahoo-inc.com

ABSTRACT

Traditional machine-learned ranking algorithms for web search are trained in batch mode, which assume static relevance of documents for a given query. Although such a batch-learning framework has been tremendously successful in commercial search engines, in scenarios where relevance of documents to a query changes over time, such as ranking recent documents for a breaking news query, the batch-learned ranking functions do have limitations. Users' real-time click feedback becomes a better and timely proxy for the varying relevance of documents rather than the editorial judgments provided by human editors. In this paper, we propose an online learning algorithm that can quickly learn the best re-ranking of the top portion of the original ranked list based on real-time users' click feedback. In order to devise our algorithm and evaluate it accurately, we collected exploration bucket data that removes positional biases on clicks on the documents for recency-classified queries. Our initial experimental result shows that our scheme is more capable of quickly adjusting the ranking to track the varying relevance of documents reflected in the click feedback, compared to batch-trained ranking functions.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms

Keywords

Online learning, Recency ranking, User feedback

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

1. INTRODUCTION

Ranking a list of documents with respect to relevance to a given query is the central problem in various search applications of the Internet. The scale and dynamics of the Web requires the machineries for such ranking problems go beyond the traditional information retrieval methods, and many of the recent developments on the ranking problems are based on various techniques in machine learning. Machine-learned ranking algorithms have been shown very effective in generalizing to unseen data from the labeled training data and have been successful particularly in commercial web search engines. One of the main premise of such machine-learned ranking algorithms is to learn a ranking function in a *batch* mode. That is, based on collected labelled training data, a machine learning algorithm is trained once (and periodically retrained), then applied to unseen examples to predict relevance and do the ranking based on that predicted relevance scores. The main assumption of the batch approach is that the relevance of documents for a given query are more or less static over time, and thus, once the coverage of training set is wide enough, learning the ranking function single time from the training set would be sufficient to generalize to the unseen data in the future.

Although the static assumption on the relevance of documents holds for many cases, there is also an important category of applications where the relevance of documents to a query may change over time as well. One such example is the *recency ranking* in web search [3]. That is, when breaking news emerges, a document that would have been very relevant to a certain query may become less relevant than the recent documents that have the most recent news stories about the query. This kind of relevance drift can usually be well reflected in users' behavior, especially in click feedback in the temporal dimension. The key challenge of such problems is how to reflect such drift quickly to produce better ranking.

Recently in [3], a solid attempt, which also works in the batch mode, has been taken to address the above challenge by devising time-varying features and recency demoted labels provided by human editors. However, their batch approach still is limited in quickly tracking the varying relevance of documents in a timely manner since obtaining good quality of batch training data can be very hard and tracking unseen temporal dynamics in the actual serving stage would be still challenging even when the training data is obtained.

Instead of adhering to the batch learning paradigm to im-

prove the quality of recency ranking, this paper devises an online learning approach that can quickly refine the results of an existing ranking function based on users' click feedback. Our rationale is that, particularly for recency queries, instantaneous click trends on the top portion of the list seem to be more important and direct indicators of document relevance than human editors' judgment labels. That is, based on the relevance and freshness features, although we can get a decent idea about the relevance of documents to a recency sensitive query, it is apparent that it is only when the actual users provide with feedback in the form of clicks can we can accurately figure out the right ranking of the documents to the query. Moreover, it is almost impossible for human editors to predict this kind of subtle differences beforehand when they are making relevance and recency judgments.

Based on the above reasoning, we carry out the following: once the editorial judgment based, batch-learned recency ranking function generates the ranked list of highly relevant documents for a recency query, we devise a *linear* re-ranking function, which is a function of query-document features and the score of the batch-trained ranking function, and sequentially update the linear coefficients based on the observed users click feedback on the documents being explored in the top portion of the list. Moreover, we work in the feature space because there are a lot of tail queries and documents that do not have enough click feedback to learn separate models for them, and thus, we need to generalize from the feedback on other queries. In order to devise our algorithm and carry out technically sound experiments, we collected exploration data, which randomly shuffles top 4 documents for every recency-classified queries' search results, from a small portion of live traffic on the commercial search engine. This data is used for both designing and evaluating our online learning algorithms, since we can collect click data without positional impression biases and simulate the actual users' experience for various re-ranking strategies. Our initial experiments show that our online-learned re-ranking algorithm increases various click metrics on recency queries not only over the editorial-based recency ranking function of [3], but also over the batch-trained re-ranking function which is trained on hold-out part of the exploration bucket data. These results show the necessity of an online-learned ranking function to appropriately track time-varying relevance for recency queries.

The rest of the paper is organized as follows. Section 2 reviews previous work that is related to our work and describes the differences with ours. Section 3 outlines our algorithm and evaluation methodology. Section 4 summarizes our preliminary empirical results. Finally, concluding remarks are given in Section 5. Due to space limitation, we leave a detailed description of our method and results to an extended version of the paper.

2. RELATED WORK

As mentioned in Section 1, an extension of this framework on the recency ranking problem in web search was first proposed in [3], which remains in the batch training framework. Using users' click feedback to improve the ranking quality of search engines is not new and many previous research work has visited it. [4, 2] and many others, for example, developed user behavior models based on users' click log data and used the model output as input features to the batch training algorithms. Most of those features, however, were computed in an average sense and are hard to reflect the temporal variations of relevance of documents. [9] had more similar flavor as our work in that they used click data to directly modify the ranking based on the inference on the users' relative preferences on rankings. Taking temporal variation of relevance into account to produce better rankings has also been considered in the recommender systems literature. [5] devised a scheme to capture temporal dynamics of user ratings on items in a conventional collaborative filtering problem. Personalized recommendation of articles on the Web [1, 6] is another closely related problem.

3. OUR METHOD

Ranking problem for recency-classified queries has three main challenges: relevance drifting, dynamic content pool, and data sparsity. We use online learning, explore-exploit techniques, and feature-based scoring to address these issues, respectively. Moreover, we try to maximize CTR@1 of our algorithm, which we believe is a reasonable metric for recency-query ranking.

3.1 Exploration/Permutation Bucket

As described in Section 1, we first setup a bucket to collect exploration data for a small portion of live traffic from a commercial search engine between Jan 29, 2010 and Feb 4, 2010. Throughout these days, we collected $652,637$ search sessions that contained $82,590$ recency-classified queries. The ranked list for those queries were generated by the recency ranking function trained as described in [3] and the ranking score for each query-document was recorded. For each session, we randomly shuffled the top 4 results and logged the permutation id of each shuffled permutation and user clicks on the corresponding permuted ranking results. 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 of document relevance.

Moreover, our session data set is very sparse and long-tailed, in which 91.1% of queries were issued no more than 10 times. The reason for this sparsity is because 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, also classified as recency queries.

3.2 Notation

The online-learning-to-rerank problem is naturally modeled as a round-by-round process: at round t,

- A user arrives and types in a query q_t;
- The default recency ranking function generates an ordered list of s documents with highest relevance scores. Then, our re-ranking function re-orders these documents and present to the user the re-ordered ranked list $\{u_{t,1}, \ldots, u_{t,s}\}$. In our exploration bucket, $s = 4$.
- The user provides feedback $r_t = \{c_{t,1}, \ldots, c_{t,s}\}$ on our re-ranking result. Here, we define $c_{t,i} = 1$ if a user clicked on the document at potision i, and 0 otherwise.
- Finally, based on user feedback, we then update our re-ranking function.

In order to efficiently implement and update our re-ranking function, we construct a common feature vector for every query-document pair, (q, u), and denote it as $\mathbf{x}_{qu}$. Our online algorithm then maintains a re-ranking 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. Before describing our algorithm in detail, we first cover how we evaluate different algorithms.

3.3 Evaluation Methodology

A tricky part of our problem is that, unlike supervised learning, it is hard to evaluate and compare performance of algorithms using a static set of log data, since the click feedback in the log depends on the ranking results that the user observed when the log was collected. We follow the methodology established in [6] for online recommendation problems. First, we hold out the sessions for the latter three days in the exploration bucket data and use it as a test set. We then sort the test sessions in the order of time stamps. Let the number of clicks on the first position be C, which is initialized to 0. For $t = 1, 2, \ldots$,

1. We retrieve the t-th session in the test set, present the top s documents together with their features to the re-ranking function.
2. The re-ranking algorithm then proposes to display one of the documents in the first position by its re-ranking score. 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 r_t and update $C \leftarrow C + c_{t,1}$.
4. Otherwise, r_t is not revealed, and the value of C remains unchanged. Effectively, this session is ignored.

Finally, the overall CTR@1 of the algorithm in the evaluation process above is C/M, where M is the number of matched sessions. For each session in the 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 evaluation method above: it can be proved that the estimated CTR@1, C/M, of an online algorithm is an unbiased estimate of its true CTR *if we ran it to serve live user traffic* [6]. Therefore, algorithms that have higher CTR estimates using this evaluation method will have higher CTRs in live buckets as well. This crucial fact allows us to compare and evaluate various algorithms *offline* without the costs and risks of actually testing them with *live* users.

3.4 Online Algorithm

Given our goal of maximizing CTR@1 in the re-ranking results, it is tempting for an online algorithm to follow a *greedy* strategy: that is, it *always* picks (for the present query at hand) the document with the highest CTR estimate for the first position, and updates the function parameters solely based on user feedback r_t for the algorithm's re-ranked list. While this greedy approach is intuitively desirable, it can be detrimental in practice since when an algorithm mistakenly underrates a document, a greedy re-ranking strategy can prevent it from collecting user feedback for this document to correct this mistake. Consequently, the algorithm has to balance two conflicting goals: (a) "exploitation" — to display in the first position most relevant documents to maximize re-ranking quality (in our case, to maximize CTR@1), and (b) "exploration" — to display documents for the purpose of collecting data to further improvement. The exploration/exploitation tradeoff described above is a defining characteristic of a class of problems known as bandit problems [8], which has received considerable attention recently for Internet-related applications [7, 9].

Although many alternatives exist, we choose our re-ranking function to be linear in the feature vector $\mathbf{x}_{qu}$ and combine ϵ-greedy strategy to resolve the above explore-exploit tradeoff and implement our online algorithm. A total of 51 features were used, i.e., $\mathbf{x}_{qu} \in \mathbb{R}^{51}$, where those features include regular query-specific, document-specific, and query-document-specific features used in ordinary machine-learned ranking functions, and, more importantly, the ranking score generated by the default recency ranking function [3]. Since we try to maximize CTR@1, it is natural to find a function that estimates CTR@1 of a (q, u) pair for re-ranking.

Once the feature vector is given, we define the following *hybrid* model for CTR@1 of (q, u) pair:

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

where $\boldsymbol{\beta}$ is the common coefficient vector for all (q, u) pairs as before, but b_{qu} is a real-valued latent feature associated with the (q, u) pair. We also call b_{qu} a bias term for (q, u). One might wonder the necessity of using a linear model if we simultaneously allow a bias term for every (q, u) pair in the hybrid model (1). The reason is as follows: if the features in $\mathbf{x}_{qu}$ are reasonably good in predicting CTR@1(q, u), the linear model part $\boldsymbol{\beta}^\top \mathbf{x}_{qu}$ will be able to capture much information of CTR@1(q, u). Consequently, the shared coefficients $\boldsymbol{\beta}$ makes it possible to generalize CTR from one query-document pair to others, and therefore makes learning faster. At the same time, the bias terms b_{qu} can make the hybrid model more powerful since for popular (q, u) pairs, we will have enough training data so that b_{qu} can correct the estimate of $\boldsymbol{\beta}^\top \mathbf{x}_{qu}$. As usually, we need to put regularization penalty on both $\boldsymbol{\beta}$ and b_{qu} to prevent overfitting.

Our online re-ranking algorithm uses (1), which sequentially updates $\boldsymbol{\beta}$ and b_{qu} based on observed click feedback, to re-rank top 4 documents generated by the default recency ranking function. The target click feedback for updating is obtained by the ϵ-greedy strategy with $\epsilon = 1$, and we use the standard ridge regression for learning; for each time t, we compute the optimal model parameters $(\boldsymbol{\beta}_t, \{b_{qu,t}\})$ to minimize

$$f_t(\boldsymbol{\beta}, \{b_{qu}\}) \triangleq \sum_{i=1}^{t} \left(c_i - \boldsymbol{\beta}^\top \mathbf{x}_i - b_{q_i u_i} \right)^2 + \lambda_1 \left\| \boldsymbol{\beta} - \boldsymbol{\beta}^{(0)} \right\|_2^2 + \lambda_2 \sum_{(q,u)} \left\| b_{qu} - b^{(0)} \right\|_2^2, \tag{2}$$

where λ_1 and λ_2 are regularization parameters, $\boldsymbol{\beta}^{(0)}$ and $b^{(0)}$ are the prior values, and $\| \cdot \|_2$ is the ordinary ℓ_2-norm. The efficient sequential update formula for $(\boldsymbol{\beta}_t, \{b_{qu,t}\})$ can be obtained by matrix inversion lemma.

4. EXPERIMENTAL RESULTS

In this section, we report preliminary empirical evidence for the usefulness of the proposed online ra-ranking method for recency queries. We first describe the models we compared, and then show the performance of each model in our test set.

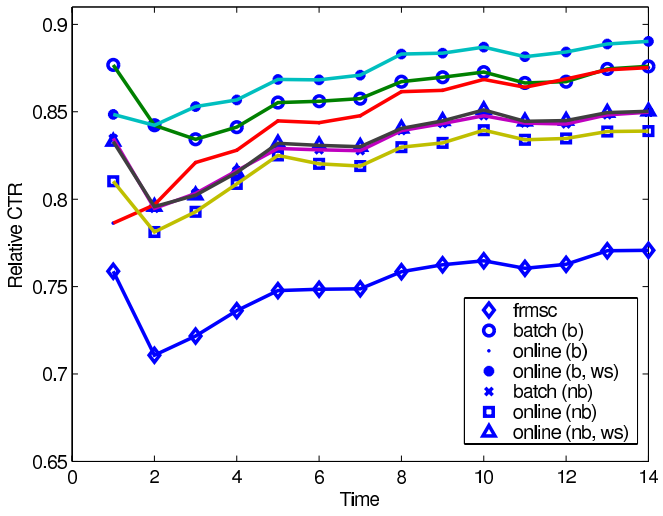

Figure 1: Cumulative (normalized) CTR@1 of the four models on the test data. Each point was measured for an approximately six-hour period.

4.1 Models

In order to demonstrate the effectiveness of our *click-based, online* learning method, we have to compare our scheme not only with the *editorial judgment-based, batch-trained* recency ranking function, but also with the *click-based batch-trained* re-ranking model. For such comparison, we partitioned our exploration bucket data into training and test set as described in Section 3.3, and following four models were compared on the test set.

- **frmsc**: This is the baseline scheme using a deployed recency ranking function [3].
- **batch(b)**: This is the linear model obtained by minimizing (1) on the training set.
- **online(b)**: This is the model described in Section 3, where $\boldsymbol{\beta}^{(0)}$ and $b^{(0)}$ are initialized to zero.
- **online(b,ws)**: This is the same as **online(b)** except that $\boldsymbol{\beta}^{(0)}$ and $b^{(0)}$ are initialized to the solution obtained by **batch(b)**, thus the name "warm start" (ws).

For each of the last three models, we also try a variant that does *not* use the bias terms b_{qu} in (1). The corresponding models are **batch(nb)**, **online(nb)**, and **online(nb,ws)**.

4.2 Result Analysis

We focus on CTR@1 in empirical evaluation. Figure 1 plots cumulative CTR@1 on the test data set for seven different models described in the previous subsection. The unit length on the x-axis corresponds roughly to 6 hours. A few interesting observations are in order. First, it is apparent that using user feedback, either in batch or online models, helps improve CTR@1 metrics, yielding significant improvement over the **frmsc** baseline. The improvement also implies a noticeable discrepancy between editorial judgments and users' click feedback on recency queries. Second, **online(b,ws)** dominates all other models almost during all three days, verifying the importance of online learning with warm start and bias terms. Finally, it is clear that using the bias terms in (1) helps improve CTR@1, confirming our conjecture that they are useful to correct the linear model's prediction error of CTR@1 for popular queries. In other words, the bias term serves as a latent feature for each query-url, which turns out to be a strong signal.

5. DISCUSSION AND CONCLUSION

In this work, we proposed a novel framework for using online learning algorithms to do Web search re-ranking based on real-time user feedback. Our contributions are three-fold. First, we demonstrate the need for using online learning as a flexible machine learning paradigm to adapt a ranking system to non-stationary document relevance. Second, our evaluation method is novel—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 algorithm *offline*. Third, we proposed a novel and principled algorithm for doing online re-ranking. This algorithm can be efficiently implemented and easily incorporated into existing ranking engines. Empirical results suggest effectiveness of our proposed solutions, and we plan to report more detailed evaluation results of our re-ranking method in a full version of this paper.

6. REFERENCES

[1] D. Agarwal, B. Chen, and P. Elango. Explore/exploit schemes for web content optimization. *Proceedings of the International Conference on Data Mining (ICDM), 2009* .

[2] O. Chapelle and Y. Zhang. A dynamic bayesian network click model for web search ranking. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 1–10, New York, NY, USA, 2009. ACM.

[3] A. Dong, Y. Chang, Z. Z, G. Mishne, J. Bai, R. Zhang, K. Buchner, C. Liao, and F. Diaz. Towards recency ranking in web search. *Proceedings of the third International ACM Conference on Web Search and Data Mining (WSDM), 2010.*

[4] G. Dupret and C. Liao. Cumulated relevance: A model to estimate document relevance from the clickthrough logs of a web search engine. In *Proceedings of the third International ACM Conference on Web Search and Data Mining (WSDM)*, 2010.

[5] Y. Koren. Collaborative filtering with temporal dynamics. *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD), 2009.*

[6] L. Li, W. Chu, J. Langford, and R. Schapire. A contextual bandit approach to personalized news article recommendation. *Proceedings of 19th World Wide Web (WWW) Conference, 2010.*

[7] F. Radlinski and T. Joachims. Active exploration for learning rankings from clickthrough data. *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD), 2007.*

[8] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[9] Y. Yue and T. Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. *Proceedings of the 26 th International Conference on Machine Learning, Montreal (ICML), Canada, 2009.*