Smoothing Clickthrough Data for Web Search Ranking

Jianfeng Gao*, Wei Yuan#, Xiao Li *, Kefeng Deng$, Jian-Yun Nie#
*Microsoft Research, Redmond, USA, Email: {jfgao; xiaol}@microsoft.com;
$Microsoft, Search Technology Center, Beijing, China, Email: kefengd@microsoft.com;
#University of Montreal, Canada, Email: {yuanwei; nie}@iro.umontreal.ca

ABSTRACT

Incorporating features extracted from clickthrough data (called *clickthrough features*) has been demonstrated to significantly improve the performance of ranking models for Web search applications. Such benefits, however, are severely limited by the data sparseness problem, i.e., many queries and documents have no or very few clicks. The ranker thus cannot rely strongly on clickthrough features for document ranking. This paper presents two smoothing methods to expand clickthrough data: query clustering via Random Walk on click graphs and a discounting method inspired by the Good-Turing estimator. Both methods are evaluated on real-world data in three Web search domains. Experimental results show that the ranking models trained on smoothed clickthrough features consistently outperform those trained on unsmoothed features. This study demonstrates both the importance and the benefits of dealing with the sparseness problem in clickthrough data.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.2.6 [**Artificial Intelligence**]: *Learning*

General Terms

Algorithms, Experimentation

Keywords

Clickthrough Data, Smoothing, Random Walk, Discounting, Learning to Rank, Web Search

1. INTRODUCTION

We consider the task of ranking Web search results, i.e., a set of retrieved Web documents (URLs) are ordered by relevance to a query issued by a user. In this paper we assume that the task is performed using a ranking model (also called *ranker* for short) that is learned on labeled training data, i.e., human-judged query-document pairs. The ranking model is a function that maps the feature vector of a query-document pair to a real-valued relevance score. Such a learned ranking model is shown to be superior to classical retrieval models [6, 11] largely due to its ability to integrate both traditional criteria such as TF-IDF and BM25 values, and non-traditional features such as hyperlinks.

In general Web search, a document can be described by multiple text streams. Some of the most useful text streams for Web search are (1) a *content stream* consisting of all the title and body texts in a page, (2) an *anchor stream* consisting of all the anchor texts of a page's incoming links, and (3) a *clickthrough stream* consisting of all the user queries that have click(s) on the document. Recent research shows that incorporating features extracted from the clickthrough stream (called *clickthrough features*) could significantly improve the performance of ranking models for Web search because the clickthrough stream can provide complementary information about a user's intention [1].

However, clickthrough data typically suffer from the sparseness problem. Two related aspects are involved. First, for a query, users only click on a very limited number of documents, thus the clicks are not complete. We refer to it as the *incomplete click* problem. Second, for many queries and documents, no click at all is made by users. We call this the *missing click* problem. As a consequence, the clickthrough streams for most of documents are either short or empty. Although one can use such raw text streams to extract some clickthrough features as in previous studies (e.g., [1, 6, 7]), their potential is severely limited because of the following reasons: First, with incomplete clicks, the click-related features that we can generate for a document-query pair are also incomplete and unreliable. Second, no clickthrough features can be generated for pairs without clicks. In the rankers used in most previous studies [1, 6, 7], this is equivalent to assigning zero values for clickthrough features. In ranker training, the zero-valued features make a categorical difference between the documents with and without clicks, and severely penalize the documents without clicks. However, in reality, the "true" difference between these documents may be much smaller because a document could be unclicked for a variety of reasons even if the document is relevant.

The missing click problem bears a strong resemblance to the problem of determining the frequency or probability of an unseen event, which has been well-studied in the context of estimating n-gram language models [8]. Various smoothing techniques have been proposed and successfully used to deal with this problem, including clustering (by grouping observations on similar n-grams) and discounting (by assigning some counts to unseen n-grams) [8, 14]. In the case of clickthrough data, we can consider a click for a document-query pair as an n-gram. Then clickthrough data can also be smoothed in two directions: by clustering similar queries or by assigning non-zero values to the clickthrough features of unclicked documents through discounting. In this paper, we propose to perform query clustering via Random Walk on click graphs, and a discounting method inspired by the Good-Turing estimator [13]. The Random Walk method is intended to address the incomplete click problem. In some particular settings, such as image retrieval [9] and query classification [21], it has been shown that expanding clicks to similar documents and queries via Random Walk can lead to significant improvements. However, to our knowledge, no

1. Message Web Design – Home
www.message.uk.com
2. **MSN Web Messenger**
webmessenger.msn.com
2008-11-01 15:01:15
3. High School Baseball Web
www.hsbaseballweb.come/message_boards.htm
4. Send a Wireless Web Message
messaging.sprintpcs.com
5. SprintPCS 2Way SMS
messaging.sprintpcs.com/sml/guestcompose.do
6. Email on the Web
webmail.netzero.net
7. USA MOBILITY
www.arch.com/message
8. Message Boards – rootsweb.com
boards.rootsweb.com
9. **Yahoo! Messenger – Chat, Instant message**
messenger.yahoo.com
2008-11-01 15:00:59
10. Yahoo! Message Boards – Home
messages.yahoo.com

Figure 1. The query session for the query "web message". Marked in bold are the links the user clicked on.

msn web	0.6675749
webmensseger	0.6621253
msn online	0.6403270
windows web messanger	0.6321526
talking to friends on msn	0.6130790
school msn	0.5994550
msn anywhere	0.5667575
web message msn com	0.5476839
msn messager	0.5313351
hotmail web chat	0.5231608
messenger web version	0.5013624
instant messager msn	0.4550409
browser based messenger	0.3814714
im messenger sign in	0.2997275
msn web browser download	0.0926431
msn passport	0.0035466
download msn messenger 6	0.0027844
install msn toolbar	0.0027248
msn people	0.0025993
…	…

Figure 2. Fragments of the clickthrough stream for the link *http://webmessenger.msn.com*

study has been carried out on general Web search applications showing a similar improvement. Our experiments show that the expanded clickthrough data is noisy, and it should be used with caution. Effective improvement is possible only when we extract those features that are robust to noise for ranking. Notice that documents and queries with no click cannot be enriched through Random Walk.

Thus, inspired by the Good-Turing method [13, 20], we present a discounting method to estimate the values of the clickthrough features for the documents without clicks.

Our experiments will show that both smoothing techniques can significantly improve the retrieval effectiveness compared to the utilization of raw clickthrough data. In particular, the simple discounting method will prove to be effective on all the three test datasets. This series of experiments strongly indicate that sparseness is a crucial problem in clickthrough data, and an appropriate solution to this problem allows us to better take advantage of clickthrough data.

In the rest of the paper, Section 2 describes background information on clickthrough data and rankers. Section 3 presents two smoothing techniques. Section 4 presents experiments. Related work and conclusions are presented in Sections 5 and 6.

2. BACKGROUND

In this section, we first describe the clickthrough data we use and the way a Web document is represented by a clickthrough stream. Then, we present the clickthrough features to be incorporated in ranking models. Finally, we review the ranking model used in our experiments. Notice that we focus on clickthrough features in this paper. The features extracted from other text streams will remain unchanged and be used in the same manner as before.

2.1 Clickthrough Streams for Documents

Clickthrough data used in this study consists of a set of query sessions that were extracted from one-year log files of a commercial Web search engine. A query session contains a query issued by a user and a rank list of (top-10) links browsed by the same user (with or without click). Following the notations in [18], a query session is represented by a triplet (q, r, c) consisting of the query q, the ranking r presented to the user, and the set c of links (documents) the user clicked on. Figure 1 shows a query session for the query "web message". The documents #2 and #9 are clicked by the user, and the dates and times of the two clicks are also recorded.

Previous work has utilized clickthrough data as implicit feedback for Web search ranking in two different ways. The first approach is to derive training data from clickthrough data directly [18, 19, 26]. In particular, [19] argued that relative preferences derived from clicks are reasonably accurate. For example, in Figure 1, the document #2 is assumed to be more relevant to the query "web message" than #1 because #2 is clicked, and #1, though ranked higher than #2, is not clicked. By doing so, one could derive a large amount of preference pairs. Then a ranking algorithm, such as LambdaRank [6], can be trained on such preference pairs.

The second category of work is to derive features from the clickthrough data and incorporate them into a ranking model [1, 28]. Our approach belongs to this category. The method is based on the assumption that all the queries that have clicks on a document form a description of the document from users' perspective. One can see an example of such a clickthrough stream in Figures 2 for the document "*webmessenger.msn.com*". It consists of all the queries that have one or more clicks on the document. In Figure 2, each line in a clickthrough stream consists of a query and a clickthrough score *Score*(d, q), which can be considered as the importance of the query q in describing the document d, similarly to the TF-IDF scores. The score can be derived from raw click information recorded in log files heuristically. In our experiments, one of the simplest functions that work well across all data sets is:

StreamLength_w	# of words in CS
StreamLength_q	# of queries in CS
WordsFound	Ratio between # of words in *q* that occur in CS and # of words in *q*
CompleteMatches	Sum of the scores of the queries in CS all of whose words are included in *q*
PerfectMatches	Sum of the scores of the queries in CS that match *q* (as a single string)
ExactPhrases	Sum of the scores of the queries in CS that contain *q* as a substring
Occurrences_i	Sum of the scores of the queries in CS that contain the *i*-th (i = 1…N) word of *q*
Bigrams	Sum of the scores of the queries in CS that contain any word-pair in *q*
InorderBigrams	Sum of the scores of the queries in CS that contain any word-bigram in *q*
… …	

Figure 3. Some clickthrough features used in ranking models, where *q* is the input query, containing *N* query words (stop words are removed); CS is the clickthrough stream that consists of a set of query-score pairs.

$$Score(d,q) = \frac{C(d,q,click) + \beta * C(d,q,last\_click)}{C(d,q)}, \quad (1)$$

where $C(d,q)$ is the number of times that *d* is shown to the users when *q* is issued (so, $C(d,q)$ is sometimes called the number of *impressions*), $C(d,q,click)$ is the number of times that *d* is clicked for *q*, and $C(d,q,last\_click)$ is the number of times that *d* is the temporally last click of *q* in clickthrough data. For example, in Figure 1 the documents #2 and #8 are the clicks of the query, but only #2 is the last click. Here, the weight β is a scaling factor, empirically tuned (β = 0.2 in our experiments). Intuitively, if a document is the last click of a query, there is a higher chance that the user is satisfied by this document and no additional document is necessary. Therefore, we boost the score of the last-clicked documents in the above formula.

2.2 Clickthrough Features for Ranking

In modern Web search engines, search results are ranked based on a large number of features extracted from query-document pairs. Since a document is described by multiple text streams, multiple sets of features can be extracted, one from each stream (with respect to the query). Therefore, using clickthrough data for ranking is equivalent to incorporating the clickthrough features, which are extracted from the clickthrough steam, in the ranking algorithm. As described in [1], during training, the ranker can be learned as before but with additional features. At runtime, the search engine would fetch the clickthrough features associated with the given query-document pair and determine a relevance score.

Figure 3 lists some of the most important clickthrough features we used in our experiments, and describes how their values are computed from the clickthrough scores of the matched queries (to an input query). Let us illustrate this by an example. Consider a clickthrough stream consisting of 4 query-score pairs, as follows.

Query	Score
A B C D	*S1*
B C A	*S2*
E A B C D F	*S3*
B A E	*S4*

Now, given a 4-word input query A B C D, the values of the clickthrough features are as follows.

StreamLength_w	*16*
StreamLength_q	*4*
WordsFound	*1*
PerfectMatches	*S1*
CompleteMatches	*S1* + *S2*
ExactPhrases	*S1* + *S3*
Occurrences_1	*S1* + *S2* + *S3* + *S4*
…	…

2.3 Ranking Model and Quality Measure in Web Search

Many rankers can be used to incorporate a set of features, such as RankSVM [18], or RankNet [7]. In this study, we will use LambdaRank. Details can be found in [6]. We only sketch it here.

We assume that training data is a set of input/output pairs (**x**, *y*). **x** is a feature vector extracted from a query-document pair, where the document is represented by multiple text streams as described in Section 2.1. We use about 300-400 features extracted from content and anchor text streams, including dynamic ranking features such as term frequency and BM25 value and static features similar to PageRank, as well as a set of (44 or 30) clickthrough features. *y* is a human-judged relevance score, from 0 to 4, with 4 as the most relevant.

LambdaRank is a neural net ranker that maps a feature vector **x** to a real value *y* that indicates the relevance of the document given the query. For example, a linear LambdaRank simply maps **x** to *y* with a learned weight vector **w** such that $y = \mathbf{w} \cdot \mathbf{x}$. Several non linear functions are provided in LambdaRank. LambdaRank is particularly interesting to us due to the way **w** is learned. Typically, **w** is optimized with respect to a cost function using numerical methods if the cost function is smooth and its gradient with respect to **w** can be computed easily. In order for the ranker to achieve the best performance in document retrieval, the cost function used should be the same as, or as close as possible to, the measure used to assess the final quality of the system. In Web search, *Normalized Discounted Cumulative Gain* (NDCG) [17] is widely used as quality measure. For a query *q*, NDCG is computed as follows:

$$\mathcal{N}_i = N_i \sum_{j=1}^{L} \frac{2^{r(j)} - 1}{\log(1+j)}, \quad (2)$$

where $r(j)$ is the relevance level of the *j*-th document, and the normalization constant N_i is chosen so that a perfect ordering would result in $\mathcal{N}_i = 1$. Here L is the ranking truncation level at which NDCG is computed. The $\mathcal{N}_i$ are then averaged over a query set. However, NDCG, if it were to be used as a cost function, is either flat or discontinuous everywhere. It thus presents particular challenges to most optimization approaches that require the computation of the gradient of the cost function.

LambdaRank solves the problem by using an implicit cost function whose gradients are specified by rules. These rules are

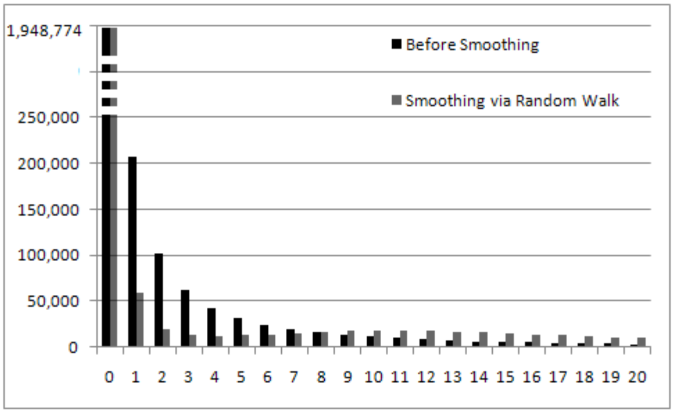

Figure 4. Length distribution of the clickthrough streams (with StreamLength_q ≤ 20) in the Japanese training data, where *x*-axis is the stream length, and *y*-axis is the number of training samples; Bars at *x* = 0 shows the number of documents without click. Black bars correspond to the raw click counts and grey bars to the smoothed counts using Random Walk.

called *λ-functions*. Burges et al. [6] studied several λ-functions that were designed with the NDCG cost function in mind. They showed that LambdaRank with the best λ-function outperforms significantly a similar neural net ranker, RankNet [7], whose parameters are optimized using the cost function based on cross-entropy. In this paper, we will use LambdaRank with a sigmoid function, as our ranker and we explore different ways to integrate clickthrough features in it.

3. TWO SMOOTHING TECHNIQUES

An analysis of the data sets in all the three search domains of our study reveals a severe sparseness problem of the clickthrough data. Take the Japanese training data as an example. Around 75% of 2.62 million samples (i.e., query-document pairs) do not have any click (see Figure 4). That is, the clickthrough features of about 1.95 million samples are assigned a zero value (the missing click problem). For the rest of the data, the lengths of the clickthrough streams have a very skewed distribution, with a majority of samples having very short (< 5 queries) clickthrough streams, as illustrated in Figure 4 (the incomplete click problem).

These sparseness problems are largely attributable, on the one hand, to the bias of the search results retrieved by an imperfect search engine (i.e., most users only see a few top, typically 10, search results and do not see the others), and on the other hand, to the incomplete clicks by the user even if many relevant documents are shown to the user. Both sparseness problems have their counterparts defined and studied in the machine learning research community. The missing click problem can be viewed as a particular example of the missing data problem [22], and the incomplete click problem is related to confidence-weighted learning presented in [10]. We will return to the related work in Section 5.

This situation makes it clear that the raw clickthrough data is imperfect and unreliable (in the sense that an unclicked document is not necessarily non-interesting). Instead of using the raw clickthrough data, a better approach is to derive a new clickthrough data, which contains the "expected" clickthrough features, in which the raw clickthrough features are generalized or expanded to other documents. This idea is very similar to smoothing in statistical language modeling (SLM). Many studies showed that the model trained with expected counts can better capture the language usage than the raw counts. It can then be expected that a similar processing on raw clickthrough data could produce a similar effect.

Inspired by the smoothing techniques for SLM, we propose two methods to smooth clickthrough data: clustering and discounting.

3.1 Random Walk

Clustering techniques have been widely used in language modeling to improve the reliability of probability estimation [5, 14, 15]. Consider a large text corpus containing N words in which a word w_1 occurs once and another word w_2 occurs twice. If a unigram model were built using maximum likelihood estimation without smoothing, the model would say that the probability that w_2 occurs in a new text, $P(w_2)$, is twice as large as that of w_1, $P(w_1)$. However, these probabilities are not reliable because they are estimated on few samples. Now suppose that we could group similar words into clusters. Assume that W_1 and W_2 are the clusters of w_1 and w_2, respectively. If W_1 occurs 200 times, and W_2 400 times, then one is more confident to say that $P(W_2)$ is twice as large as $P(W_1)$.

The same idea can also be used to smooth clickthrough features. Consider the StreamLength feature as an example. We would not be confident to say that a document d_1, with StreamLength_q = 2, is twice as popular as a document d_2, with StreamLength_q = 1. However, if we could expand the stream with "similar" queries that are likely to click the same document but are not recorded in log data for some reason, and observe that the expanded streams of d_1 and d_2 are 200 and 100 in StreamLength_q, respectively, then we are more confident to say that d_2 is more popular. Similar idea applies to other clickthrough features.

Now, the question is how to determine similar queries that should have clicked on the document. One can use a similarity defined according to query terms. However, this would unlikely add very different queries into the click stream. Another solution is to exploit co-clicks: queries for which users have clicked on the same documents can be considered to be similar. This principle has been successfully used in several studies [e.g., 3, 9, 27]. We follow the same principle here, but use a different approach: instead of defining a static function of similarity according to the number of co-clicks, we use random walk to derive it dynamically. This approach has been successfully used in [9]. Figure 5 gives an example that illustrates this idea.

Formally, we construct a click graph which is a bipartite-graph representation of clickthrough data. We use $\{q_i\}_{i=1}^{m}$ to represent a set of query nodes and $\{d_j\}_{j=1}^{n}$ a set of document nodes. We further define an $m \times n$ matrix W in which element W_{ij} represents the click count associated with (q_i, d_j). This matrix can be normalized to be a query-to-document transition matrix, denoted by A, where $A_{ij} = p^{(1)}(d_j|q_i)$ is the probability that q_i transits to d_j in one step. Similarly, we can normalize the transpose of W to be a document-to-query transition matrix,

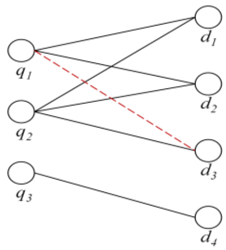

Figure 5. Before expansion, document d_3 has a clickthrough stream consisting of query q_2 only; after expansion, the clickthrough stream is augmented with query q_1 which has a similar click pattern as q_2.

denoted by B, where $B_{j,i} = p^{(1)}(q_i|d_j)$. It is easy to see that using A and B we can compute the probability of transiting from any node to any other node in k steps. There are various ways of evaluating query similarities based on a click graph, e.g. using *hitting time* [25]. In this work, we use a simple measure which is the probability that one query transits to another in 2s steps; and the corresponding probability matrix is given by $(AB)^s$. Although longer transitions could be used, the most effective transitions are the first ones, and longer transitions also raise the problem of efficiency. So, in our experiments, we limit s to 1.

Based on this measure, we propose two heuristics for clickthrough stream expansion. For each query q in the original clickthrough stream, we select up to 8 similar, previously absent queries to be added into the expanded stream. A newly added similar query q' must satisfy $p^{(2)}(q'|q) > \alpha$, where α is tuned empirically on validation data ($\alpha = 0.01$ in all the experiments in Section 4). Alternatively, we can select similar queries if $p^{(2)}(q|q') > \alpha$. Empirical experiments show no significant difference in using these two heuristics. In Figure 4, one can observe the effect of Random Walk smoothing. The length of clickthrough stream is generally increased and we can expect more reliable clickthrough features to be extracted. However, we observe that the number of streams of length 0 remains the same because they are not affected by Random Walk smoothing. The technique of discounting described in the next subsection aims to solve this problem.

3.2 Discounting

Many smoothing methods have been proposed in SLM to deal with unseen words [13, 20]. Our method is inspired by the Good-Turing estimator, which will be reviewed briefly.

Let N be the size of a sample text, and n_r be the number of words which occur in the text exactly r times, so that

$$N = \sum_r r n_r. \tag{3}$$

Good-Turing's estimate P_{GT} for a probability of a word that occurred in the sample r times is

$$P_{GT} = \frac{r^*}{N} \tag{4}$$

where

$$r^* = (r+1)\frac{n_{r+1}}{n_r}. \tag{5}$$

The procedure of replacing an empirical count r with an adjusted count r^* is called *discounting*, and the ratio r^*/r is a discount coefficient. When applying Good-Turing discounting to estimating n-gram language model probabilities, Katz [20] suggested not discounting high values of counts, considering them as reliable. That is, for $r > k$ (typically k = 5), we have $r^*=r$.

Notice that $(r+1)n_{r+1}$ is the total count of words with frequency r+1. Let us denote it by C_{r+1}. Then Equation (5) can be rewritten as:

$$r^* = \frac{C_{r+1}}{n_r}. \tag{6}$$

One of the most straightforward manners of applying the Good-Turing method to our case is to replace a raw click count, such as $C(d,q,last)$ and $C(d,q,click\_last)$ in Equation (1), with its adjusted count according to Equation (5). However, this does not work here. While the clickthrough scores are derived from the raw click counts, the values of the clickthrough features are computed based on not only the clickthrough scores but also the specific words in the clickthrough stream, as illustrated in Figure 3. If we were to adjust the raw click counts, we would have expanded the clickthrough stream of a document to an infinitely large set by assigning a non-zero score to any possible query that does not have a click on the document. This would make most of the features, whose values are based on word or n-gram matching, meaningless. Therefore, instead of discounting raw click counts as in the Good-Turing estimator, we have developed a heuristic method, inspired by the Good-Turing estimator, which directly discounts the clickthrough feature values.

Let f_r be the value of a clickthrough feature in a training sample whose clickthrough stream is of length r, where the length is measured as the number of the queries that have click(s) on the document (i.e., StreamLength_q in Figure 3). Assume that the feature values f_r, for $r > 0$, have been smoothed using the Random Walk based method described in Section 3.1. To address the missing click problem, we only need to estimate an adjusted clickthrough feature value f_0*. Obviously, we have $f_0 = 0$ for all the raw clickthrough features.

Let $f_{1,i}$, $i = 1 \ldots n_1$, be the value of a feature in the i-th training sample whose clickthrough stream is of length 1. The sum of $f_{1,i}$ over all the training samples is $\sum_{i=1}^{n_1} f_{1,i}$. Then, similar to Equation (6), f_0* is computed as

$$f_0^* = \frac{\sum_{i=1}^{n_1} f_{1,i}}{n_0}. \tag{7}$$

where n_0 is the number of the samples whose clickthrough streams are empty. Notice that the average value of f_1 over all training samples is $f_1' = 1/n_1 \sum_{i=1}^{n_1} f_{1,i}$. Since $n_0 \gg n_1$, as shown in Figure 4, we have $f_1' \gg f_0^* > f_0 = 0$. That is, for each type of clickthrough features, Equation (7) assigns a very small non-zero constant if the feature is in a training sample whose clickthrough stream is empty (i.e., the raw feature value is zero). This will prevent the ranker from considering unclicked documents to be categorically different from clicked ones. As a consequence, the ranker can rely more on the smoothed features. Before we empirically test the impact of the smoothing method in the next section, here is an example of illustrating why this simple method might work.

Assume that given a query q, two documents, d_1 and d_2, have been retrieved based on their content streams. Now, we want to adjust their ranks based on their clickthrough streams (i.e., using their clickthrough features such as PerfectMatches in Figure 3). Assume that d_1 has a lot of clicks and d_2 has no click

Coll.	Description	# qry.	# doc/qry
en-click	*aggregated 1-year clickthrough data*	35,374,184	3.4
name-train	*human-labeled training data*	5,752	85
name-valid	*human-labeled validation data*	476	154
name-test	*human-labeled test data*	4,370	84

Table 1. Data sets in the name query domain experiments, where # qry is number of queries, and # doc/qry is number of documents per query.

Coll.	Description	# qry	# doc/qry
en-click	*aggregated 1-year clickthrough data*	35,374,184	3.4
long-train	*human-labeled training data*	6,255	93
long-valid	*human-labeled validation data*	532	159
long-test	*human-labeled test data*	5,785	123

Table 2. Data sets in the long query domain experiments.

Coll.	Description	# qry.	# doc/qry
jp-click	*aggregated 1-year clickthrough data*	3,958,820	4.7
jp-train	*human-labeled training data*	47,919	55
jp-valid	*human-labeled validation data*	4,730	119
jp-test	*human-labeled test data*	3,959	178

Table 3. Data sets in the Japanese query domain experiments.

because d_2 is a new URL and we have not collected enough click data for d_2 yet. If PerfectMatches = 0 for both d_1 and d_2, intuitively d_2 should be ranked higher because the fact that q does not match any queries, collected previously, which have clicks on d_2 seems to provide a piece of evidence that d_1 might be irrelevant, whereas there is no evidence about the (ir)relevance of d_2. Using the discounting smoothing method of Equation (7), d_2 would be ranked higher, in agreement with our intuition.

4. EXPERIMENTS

4.1 The Data

We evaluated the two smoothing methods in three Web search domains, namely (1) a person name search domain, which consists of only person name queries, (2) a long query domain, which consists of queries containing four or more words, and (3) a Japanese query domain, which consists of queries users submitted to the Japanese search market. The statistics of these data sets are shown in Tables 1 to 3. We chose English name queries and long queries for our experiments because we've collected large amounts of clickthrough data in these domains and we believe that the clickthrough features, if their values could be properly estimated, should lead to a significant improvement. We also evaluated our methods on Japanese queries because our Japanese clickthrough data is an order of magnitude smaller than English log data, but the human-labeled Japanese training data is almost an order of magnitude larger than the training data sets in the first two domains. We expect that the different settings could help us know in what case our methods perform well.

For each domain, we used two different data sets. They contain queries that are sampled from the query log files of a commercial Web search engine of two non-overlapping periods of time. We used the more recent one as test set, and split the older data set into two non-overlapping data sets: training and validation sets. This setting provides a good simulation to the realistic Web search scenario, where the ranking models in use are usually trained on previously collected data.

#	Models	NDCG@1	NDCG@3	NDCG@10	AveNDCG
1	**λ-Rank-374**	0.4981	0.5130	0.5716	0.5363
2	**λ-Rank-418**	0.5021	0.5163	0.5723	0.5381
3	**2 + GT**	0.5151	0.5240	0.5776	0.5452
4	**2 + RW-44**	0.5151	0.5198	0.5737	0.5409
5	**2 + RW-02**	0.5187	0.5275	0.5787	0.5472
6	**3 + RW-44**	0.5219	0.5242	0.5752	0.5448
7	**3 + RW-02**	0.5398	0.5403	0.5879	**0.5595**

Table 4. Test results on name-test. λ-Rank-374 is a ranker trained using LambdaRank with 374 features; GT stands for the discounting method inspired by the Good-Turing estimator; RW-44 is the query smoothing method based on Random Walk, using 44 clickthrough features, while RW-02 uses 2 clickthrough features.

In the name query and the long query experiments, we used 44 clickthrough features and other 374 features. These data are extracted from the same en-click data, which is generated from 1-year query sessions as follows. Each query is associated to a set of documents (URLs) clicked for it, together with the click counts. For each query-document pair, we computed the counts $C(d,q)$, $C(d,q,click)$, and $C(d,q,last\_click)$, as listed in Equation (1). We only kept the pairs with $C(d, q) \geq 5$ and we computed the clickthrough scores according to Equation (1). In the Japanese query experiments, we used 30 clickthrough features and other 263 features. The clickthrough data set, jp-click in Table 3, is generated from 1-year Japanese query sessions using the same procedure as that of en-click.

In all the human-labeled data sets, each sample is labeled on a 5-level relevance scale, 0 to 4, with 4 as the most relevant. The performance of all the ranking models in our experiments is measured by NDCG on the test sets. We report NDCG scores at positions 1, 3 and 10, and the averaged NDCG score (AveNDCG), which is the arithmetic mean of the NDCG scores at 1 to 10. We also performed significance test, i.e., t-test with a significance level of 0.05. In the results reported in Tables 4 to 6 in Section 4.2, the difference between any pair of different rankers is statistically significant.

4.2 Results

Table 4 shows the results of the name query experiments. All the human-labeled data sets (in Table 1) consist of only person name queries. Row 1 in Table 4 is the result of the baseline ranker which is a 2-layer LambdaRank model with 10 hidden nodes and a learning rate of 10^{-5}, trained on name-train. It uses 374 features, i.e. without clickthrough features.

Row 2 is a LambdaRank model trained using the same parameter setting, but with an additional set of 44 clickthrough features extracted from the raw data. We incorporated these clickthrough features as follows. For each document in the three human-labeled data sets (i.e., name-train, name-valid and name-test), we built a clickthrough stream as shown in Figure 2. Then for each query-document pair, we extracted the 44 features by matching the query to the clickthrough stream of the document, and computed the values of these features, as described in Section 2.2. Finally, we appended these new clickthrough features to each query-document pair.

Row 3 is the model trained on name-train where all the 44 clickthrough features have been smoothed using the Good-Turing inspired discounting method, described in Section 3.2.

Rows 4 and 5 are the models trained on the expanded clickthrough features through Random Walk. Considering that the query clusters generated by Random Walk are noisy, we also consider using only a subset of the expanded clickthrough features that are the most reliable as follows: We grouped these features into two categories - query-dependent features and query-independent features. The feature values of the former have to be computed by matching query words to the words in a stream, such as WordsFound and CompleteMatches in Figure 3. Since similar queries are generated automatically, the expanded stream may contain arbitrary queries that are irrelevant to the document. Therefore, the quality of the query-dependent features is very sensitive to the quality of the clustering algorithm, which unfortunately is by no means satisfactory on noisy log data. The second category contains only two StreamLength features, as in Figure 3 (number of words and number of queries in the clickthrough stream). Their values can reflect the popularity of a document from users' perspective, similar to BrowseRank [23]. More importantly, since the StreamLength features do not take into account any specific word in a stream, but simply measure its length, they are much more robust to noise. Row 4 is the model trained using all the 44 expanded clickthrough features, and Row 5 is the model trained using only the two StreamLength features.

Rows 6 and 7 are similar respectively to the models in Rows 4 and 5, except that all the clickthrough features are further smoothed using the discounting method.

Our results show that (1) as observed by other researchers, incorporating clickthrough features improves the ranker significantly (Row 2 vs. Row 1); (2) as expected, smoothing can further boost the ranking performance by a large margin in the name domain experiments (Row 7 vs. Row 2); (3) interestingly, the discounting method, though simple, brings a substantial improvement; and (4) the Random Walk method works well (Row 7 vs. Row 3) but not all the expanded query-dependent clickthrough features are reliable and they should be used with care[1] (Row 7 vs. Row 6). Overall, both smoothing methods work very well in the name domain experiments, and the combination of the two smoothing methods lead to a 4.0% relative improvement (or a 2.1% of absolute improvement) in AveNDCG (Row 7 vs. Row 2), which is very significant even in users' perception[2].

Table 5 shows the results of the long query experiments. All the models were built similarly to those in Table 4, except the parameters for the LambdaRank: here, we use a 2-layer LambdaRank models with 15 hidden nodes and a learning rate of 10^{-5}, trained on long-train. The results are consistent with those in the name query experiments. The combined smoothing method still substantially outperforms the unsmoothed model by 1.3% in AveNDCG (Row 7 vs. Row 2). There is, however, one noticeable difference from the results in Table 4: The contribution of the Random Walk method, which only uses the two StreamLength features, is smaller than that of the discounting method in the long query experiments: We see in Table 5 that the discounting method contributed to a 0.82% improvement in AveNDCG (comparing results in Row 3 vs. Row 2), which is almost twice as large as that of the Random Walk method, which is 0.46% (Row 5 vs. Row 2).

#	Models	NDCG@1	NDCG@3	NDCG@10	AveNDCG
1	**λ-Rank-374**	0.4302	0.4341	0.4642	0.4456
2	**λ-Rank-418**	0.4486	0.4432	0.4697	0.4539
3	**2 + GT**	0.4520	0.4478	0.4728	0.4576
4	**2 + RW-44**	0.4507	0.4415	0.4675	0.4525
5	**2 + RW-02**	0.4538	0.4462	0.4710	0.4560
6	**3 + RW-44**	0.4473	0.4405	0.4667	0.4511
7	**3 + RW-02**	0.4563	0.4500	0.4748	**0.4598**

Table 5. Test results on long-test.

#	Models	NDCG@1	NDCG@3	NDCG@10	AveNDCG
1	**λ-Rank-263**	0.5555	0.5427	0.5418	0.5424
2	**λ-Rank-293**	0.5589	0.5503	0.5525	0.5515
3	**2 + GT**	0.5658	0.5542	0.5500	**0.5528**
4	**2 + RW-30**	0.5595	0.5456	0.5425	0.5448
5	**2 + RW-02**	0.5603	0.5482	0.5471	0.5482
6	**3 + RW-30**	0.5639	0.5518	0.5456	0.5490
7	**3 + RW-02**	0.5631	0.5537	0.5485	0.5517

Table 6. Test results on jp-test.

Table 6 shows the results of the Japanese query experiments. All the models are trained similarly to those in Table 4. The baseline ranker (Row 1) is a 2-layer LambdaRank models with 10 hidden nodes and a learning rate of 10^{-5}, trained on jp-train, with 263 features. 30 clickthrough features are used. Comparing to the experimental settings of the other two search domains abovementioned, in the Japanese experiments, the human-labeled training data is much larger and clickthrough data is much smaller. This difference leads to some changes in the results: the discounting method produces a smaller improvement than on two other datasets and the Random Walk method fails to bring any improvement. This result suggests the following possible interpretations: (1) When the amount of human-judged document-query pairs is very large, the advantage of exploiting clickthrough data is reduced. (2) The smaller amount of clickthrough data leads to much noisier expansion by the Random Walk method. In this case, it is even better not to expand the data than to do it. Therefore, the impact of the Random Walk method is more subject to the amount of clickthrough data than that of the discounting method.

To sum up, our experimental results on the three datasets suggest: (1) Incorporating clickthrough features can improve the performance of rankers substantially. (2) Smoothing clickthrough features can reduce the sparseness problem of these features, and lead to some further, significant improvements. (3) The discounting method, inspired by the Good-Turing estimator, is simple and effective. It works very well across all the data sets we tested. (4) The Random Walk method also helps in some cases, but this depends more on the amount of raw clickthrough data.

5. RELATED WORK

Although the sparseness problem of clickthrough data has been reported in many recent studies [1, 9, 26, 28], no effective solution has been tested on real web search data. To the best of our

[1] It is possible that a subset of the query-dependent features (after some transformation) is useful for ranking. We have not fully exploited each individual query-dependent feature, with different transformations. We leave it to future work.

[2] A user study conducted by Microsoft Live Search (p.c.) shows that users start to sense the improvement of ranking when the NDCG improvement is larger than 0.5%.

knowledge, the Random Walk method reported in [9] is perhaps the closest work to ours. However, [9] only tested the method on the application of image search, leaving it unclear whether it can be extended to general Web search. Clickthrough data plays a much more important role in ranking images than in ranking text documents in general Web search because the content text streams of images are usually much less informative. Radlinski et al. [26] argued that missing click is due to the ranking bias of a search engine and proposed an active learning method to collect more click data by modifying the original ranking list. Given the large amount of missing clicks, the extent to which the method could alleviate the missing click problem is questionable.

The missing click problem can be viewed as a special case of the *missing data* problem that has been well-studied in the machine learning community (e.g., [4, 16, 22]). Various heuristics have been proposed. In general, missing feature values are replaced by integration over (e.g., the mean of) the corresponding features whose value is available, weighted (or discounted) by the appropriated distribution [2, 24]. Our discounting method is also a heuristic and shares some similarities with them. One area of our future work is to explore the problem in a more principled way such as the work presented in [12] (though their method cannot be applied to our case directly), where missing data in density estimation problems are dealt with by seeking a maximum likelihood solution using the expectation maximization algorithm.

Another area worth exploring in the future concerns the incomplete click problem. Incomplete click makes the feature values unreliable for training. The Random Walk method tries to improve the reliability of features via smoothing the feature values *before* training. An alternative strategy is to take into account the uncertainty of feature values *during* training. Dredze et al. [10] introduced a class of online learning methods, called *confidence-weighted learning*, where a measure of confidence (reliability) of each feature is maintained during training so that each feature weight can be updated separately according to its confidence score.

6. Conclusions

Clickthrough data have proven useful for document ranking in Web search. However, their sparseness prevents the ranker from strongly rely on these data. In this paper, we have presented two smoothing techniques for expanding clickthrough features: Discounting and Random Walk. We have demonstrated that they lead to significant improvements compared to the utilization of raw clickthrough data. In particular, the discounting method is simple, robust, and effective. This work demonstrates both the importance and the benefits of dealing with the sparseness problem of clickthrough data. In our future work we will refine the smoothing techniques to reach the full potential of clickthrough data.

7. REFERENCES

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

[2] Ahmad, S. and Tresp, V. 1993. Some solutions to the missing feature problem in vision. In *NIPS*, pp. 393-400.

[3] Baeza-Yates, R. and Tiberi, A. 2007. Extracting semantic relations from query logs. In *SIGKDD*, pp. 76-85.

[4] Bishop, C. M. 1995. *Neural networks for pattern recognition*. Clarendon Press, Oxford.

[5] Brown, P.F., Della Pietra, V. J., de Souza, P. V., Lai, J. C. and Mercer, R. L. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4): 467-479.

[6] Burges, C. J., Ragno, R., & Le, Q. V. 2006. Learning to rank with nonsmooth cost functions. In *NIPS*, pp. 395-402.

[7] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, and Hullender, G. 2005. Learning to rank using gradient descent. In *ICML*, pp. 89-96.

[8] Chen, S. and Goodman, J. 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University.

[9] Craswell, N. and Szummer, M. 2007. Random walk on the click graph. In *SIGIR*. pp. 239-246.

[10] Dredze, M., Crammer, K. and Pereira, R. 2008. Confidence-weighted linear classification. In *ICML*. pp. 264-271.

[11] Gao, J., Qin, H., Xia, X. and Nie, J-Y. 2005. Linear discriminative models for information retrieval. In *SIGIR*. pp. 290-297.

[12] Ghahramani, Z. and Jordan, M. I. 1994. Supervised learning from incomplete data via an EM approach. In *NIPS*, pp.

[13] Good, I. J. 1953. The population frequencies of species and the estimation of population parameters. *Biomerika*, 40 (3 -4): 237-264.

[14] Goodman, J. 2001. A bit of progress in language modeling (extended version). Technical Report MSR-TR-2001-72, Microsoft Research.

[15] Goodman, J. and Gao, J. 2000. Language model size reduction by pruning and clustering. In *ICSLP*, pp. 176-182.

[16] Hastie, T., Tibshirani, R. and Friedman, J. 2001. *The elements of statistical learning*. Springer-Verlag, New York.

[17] Jarvelin, K. and Kekalainen, J. 2000. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, pp. 41-48.

[18] Joachims, T. 2002. Optimizing search engines using clickthrough data. In *SIGKDD*, pp. 133-142.

[19] Joachims, T., Granka, L., Pan, B., Hembrooke, H. and Gay, G. 2005. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR*, pp. 154-161.

[20] Katz, S. M. 1987. Estimation of probabilities from sparse data for the language model of a speech recognizer. *IEEE Trans on Acoustics, Speech and Signal Processing*, ASSP-35(3): 400-401.

[21] Li, X., Wang, Y-Y. and Acero, A. 2008. Learning query intent from regularized click graphs. In *SIGIR*, pp. 339-346.

[22] Little, R. J. A. and Rubin, D. B. 1987. *Statistical analysis with missing data*. New York: John Wiley.

[23] Liu, Y., Gao, B., Liu, T., Zhang, Y., Ma, Z., He, S., and Li, H. 2008. BrowseRank: letting web users vote for page importance. In *SIGIR*, pp. 451-458

[24] Lowe, D. and Webb, A. R. 1990. Exploit prior knowledge in network optimization: an illustration from medical prognosis. *Network: Computation in Neural Systems*, 1(3):299-323.

[25] Mei, Q., Zhou, D. and Church, K. 2008. Query Suggestion Using Hitting Time. In *CIKM*, pp. 469-478.

[26] Radlinski, F., Kurup, M. and Joachims, T. 2007. Active exploration for learning rankings from clickthrough data. In *SIGKDD*.

[27] Wen, J. Nie, J.Y. and Zhang, H. 2002. Query Clustering Using User Logs, ACM TOIS, 20 (1): 59-81.

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