PKUICST at TREC 2014 Microblog Track: Feature Extraction for Effective Microblog Search and Adaptive Clustering Algorithms for TTG

Chao Lv Feifan Fan Runwei Qiang Yue Fei Jianwu Yang*
{lvchao, fanff, qiangrw, feiyue, yangjw}@pku.edu.cn
Institute of Computer Science and Technology
Peking University, Beijing 100871,China

ABSTRACT

This paper describes our approaches to temporally-anchored ad hoc retrieval task and tweet timeline generation (TTG) task in the TREC 2014 Microblog track. In the ad hoc search, we apply a learning to rank framework which utilizes not only the various content relevance of a tweet, but also the quality of a tweet. External evidences are well incorporated in our approach with Web-based query expansion and document expansion techniques. In the TTG task, we apply star clustering and hierarchical clustering algorithm on the retrieved tweets from ad hoc retrieval task. Experimental results show that our learning to rank methods with many state-of-the-art features achieve good retrieval performance with respect to MAP and P@30 metrics. Besides, our systems for TTG task also obtain convincing recall and precision scores.

1. INTRODUCTION

Information retrieval in microblogging environment has attracted increasing attention with the growing popularity of microblog. To explore the search behavior and boost the retrieval performance in the real-time environment, TREC first introduced Real-Time Search task in 2011 [5], where a user's information need is represented by a query at a specific point in time. The Microblog track in 2014 will use the "evaluation as a service" (EaaS) model, where teams interact with the official corpus via a common API. Tweet Timeline Generation (TTG) is a new task for this year's Microblog track with a putative user model as follows: "I have an information need expressed by a query Q at time t and I would like a summary that captures relevant information." In this year's task, the summary is operationalized by a list of non-redundant, chronologically ordered tweets that occur before time t.

In the ad hoc search, we apply a learning to rank framework with the help of the official API. Hundreds of features, including semantic score features, semantic expansion features and document quality features, are extracted to obtain good retrieval performance in microblogosphere. For the semantic score and semantic expansion features, we utilize different retrieval models (i.e. language models, BM25 and TFIDF) along with query expansion and document expansion techniques, in order to compute the relevance score of a given topic and tweet from different perspectives. In the query side, aside from the traditional pseudo relevance feedback based on top ranked tweets, external evidences from the Google search results are also utilized in our retrieval models to better understand the user's search intent. In the document side, we use the topic information of the shorten URLs embedded in tweets as the external evidence, and form a new document for relevance computation. For the document quality features, we use several quality features such as the number of hashtags, the term number in the tweet, the time difference between the query issue time and the tweet post time, etc. Topics for TREC'13 Microblog track are used for model training. Finally, we re-rank the results from the API and select the top 1000 tweets as our ad hoc search results.

The submitted results for ad hoc search results (i.e. the run labeled as PKUICST3) are used as the input source of our TTG system. Two strategies are adopted to determine how many results to use as TTG candidates. One strategy is to select the top N tweets or the tweets whose ranking scores are greater than a threshold score, which is determined by preliminary experiments on the training topics. Another strategy is to select the top ranked tweets manually for each query. After selecting the candidate tweets, clustering algorithms (i.e. star clustering and hierarchical clustering) are adopted to further detect and eliminate redundant tweets in the candidate set. The star clustering algorithm is a graph partition based approach, and adopts a tuned parameter α to determine whether a tweet would generate a new cluster. We choose the central tweet of each cluster as the representative tweet and eliminate other reluctant tweets. For the hierarchical clustering algorithm, a tuned cluster distance threshold is utilized to determine the final returned cluster count. For each cluster, the tweet with the highest score is selected as the representative tweet. All the representative tweets are then collected to form our final results for TTG task.

The remainder of the paper is organized as follows: we first presents our approach for ad hoc search task in Section 2. In Section 3, we describe our system for TTG task in detail. Section 4 presents our experimental results. At last, we conclude the paper in Section 5.

2. AD HOC SEARCH TASK

In this section, we first briefly introduce our system architecture for temporally-anchored ad hoc search task. Then, the learning to rank framework is described in detail.

*Corresponding author.

At last, all the extracted features for the ad hoc search task are presented.

2.1 System Overview

As mentioned above, the Microblog track use the 'evaluation as a service' (EaaS) model, where teams interact with the official corpus via a common API. In this section, we mainly discuss the architecture of our system, which is shown in Figure 1. From the figure, we can see that our system mainly contains three components:

1. **Candidate Generation Component**, which submits topics to TREC-API [1] to generate the candidate set. Additionally, we use the query expansion techniques to retrieve more topic related tweets.

2. **Feature Generation Component**, which generates the state-of-the-arts features for the candidate tweets. In our system, three groups of features are generated, i.e. semantic score features, semantic expansion features and quality features.

3. **Re-Ranking Component**, which re-ranks candidate tweets with a pairwise learning to rank algorithm [3]. The re-ranked top 1000 relevant tweets are selected as the final results of our system.

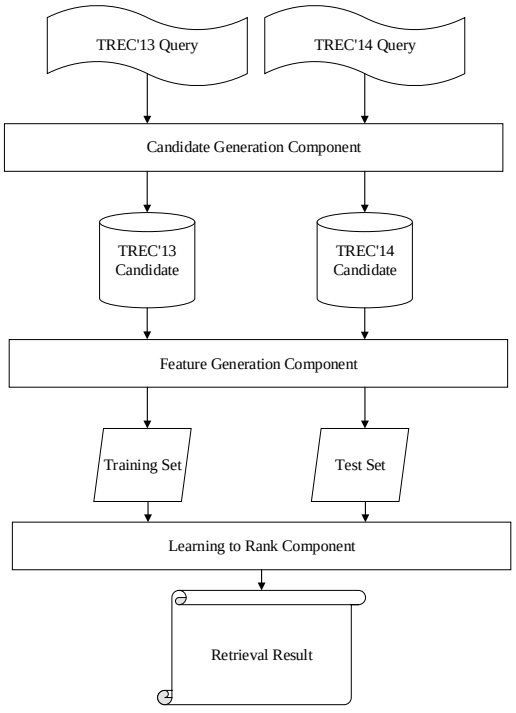

Figure 1: System Framework

The preprocessings we adopted on the queries and corpora are described as follows:

- **Non-English Filtering:** We discarded the non-English tweets by using a language detector with infinity-gram, named ldig[2].

- **Simple Retweet Elimination:** We eliminated tweets that begin with 'RT' with the consideration that these tweets have no extra information beyond the original ones.

- **Stemming and Stopword Filtering:** Each tweet was stemmed using the Porter algorithm. Moreover, stopwords were removed using InQuery words stoplists.

2.2 Learning to Rank Framework

Learning to rank is a data-driven approach which integrates a bag of features in the model effectively. Our system adopts the similar framework that Duan *et al* [1] proposed except that we extract much more features for the ad hoc search task. The learning to rank framework is shown in Figure 2. In order to train an effective model, adequate training data and useful feature set are required. Our training set is generated from the official result set of TREC'13 Microblog track. Besides, features in our proposed approach take both the similarity of query-document and the quality of the document into consideration. RankSVM algorithm [3] is utilized to train a ranking model from the training data.

2.3 Feature Generation

Several features have been proved effective in the prior work [2, 6]. However, these features are not fully utilized to further improve the performance of learning to rank approach in the microblogosphere. In this section, we describe the features used in RankSVM in detail. We classify all the feature into three groups as *semantic score features* , *semantic expansion features* and *document quality features*.

2.3.1 Semantic Score Features

Semantic score features refer to the features that describe the relevance between the query and tweets by analyzing the content of tweets. These semantic score features are listed as follows:

- **TFIDF Model Score** (*QueryTFIDFTweet*): This feature calculates the cosine similarity distance between a query and a tweet in the *Vector Space Model* with the TFIDF weighting method. *Vector Space Model* is an algebraic model for representing text documents as vectors of identifiers. We express the query and tweet as vectors:

$$\overrightarrow{Q_i} = (w_{1q}, w_{2q}, w_{3q}, \cdots w_{nq})$$

$$\overrightarrow{T_i} = (w_{1i}, w_{2i}, w_{3i}, \cdots w_{ni})$$

The TFIDF weighting scheme is adopted as the term weight and the Cosine Similarity Metric is used to evaluate the relevance between tweets and query. The Cosine Similarity Metric is defined as Eq.1.

$$Sim = \frac{\overrightarrow{T_i} \cdot \overrightarrow{Q}}{\|\overrightarrow{T_i}\| \cdot \|\overrightarrow{Q}\|} \quad (1)$$

[1]https://github.com/lintool/twitter-tools/wiki/TREC-2013-API-Specifications

[2]https://github.com/shuyo/ldig

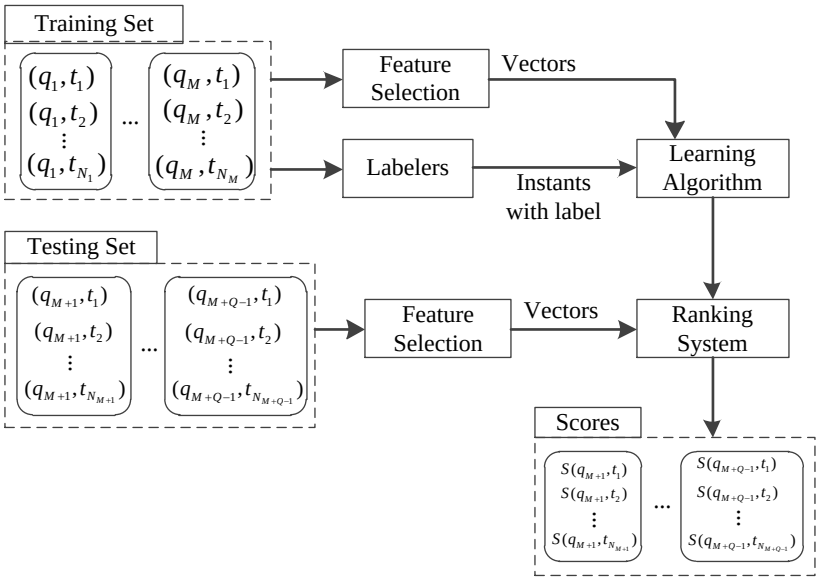

Figure 2: Learning to Rank Framework

- **Okapi BM25 Score** (*QueryBM25Tweet*): The standard Okapi BM25 weighting function is also adopted to measure the content relevance between query Q and tweet T. Okapi BM25 model is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document (e.g., their relative proximity). The similarity of a document D to query Q is defined as Eq.2.

$$Sim = \sum_{q_i \in Q} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \quad (2)$$

  where $f(q_i, D)$ is q_i's term frequency in the document D, $|D|$ is the length of the document D in words, and $avgdl$ is the average document length in the text collection from which documents are drawn. k_1 and b are free parameters.

- **Language Model Score** (*QueryLMDIRTweet, QueryLMJMTweet, QueryLMABSTweet*): We utilize the KL-divergence language model based retrieval method to measure the relevance between query language model $\widehat{\theta}_Q$ and tweet language model $\widehat{\theta}_T$. The smoothing methods we use for language model are: (1) DIR (Bayesian Smoothing with Dirichlet Priors) smoothing, (2) JM (Jelinek-Mercer method) smoothing and (3) ABS (Absolute Discounting Smoothing).

$$LMSocre(T, Q) = \sum_{w \in Q} P(w|\widehat{\theta}_Q) \cdot logP(w|\widehat{\theta}_T) \quad (3)$$

Henceforth, we can generated 5 basic semantic score features in total.

2.3.2 Semantic Expansion Features

As microblog retrieval suffers severely from the vocabulary-mismatch problem (i.e. term overlap between query and tweet is relatively small), different semantic expansion techniques can be leveraged to improve the retrieval performance. In this section, we introduce several semantic expansion features on basis of query expansion and document expansion.

To extract features related to query expansion, we first name the origin query offered by TREC'14 *OriginQuery*. For a certain *OriginQuery*, we use two strategies to extend it: (1) twitter corpus based query expansion and (2) web-based query expansion. In twitter corpus based query expansion, we first use TREC-API to get the top ranked tweet set. Then, noun and verb terms from the top ranked tweet are recognized as extension terms to generate a new query (i.e. *IssueQuery*). Then, we generate the *MergeQuery* by interpolating the *OriginQuery* and *IssueQuery*.

$$MergeQuery = \alpha \cdot OriginQuery + (1 - \alpha) \cdot IssueQuery \quad (4)$$

where α is an interpolation parameter and we set it as 0.4 in our system. In web-based query expansion, we submit the query to Google Search Engine API [3] with time limitation (before the query issue time). Noun and verb terms from the returned top 5 tweets are extracted as extension terms to generate a new query (i.e. *WebQuery*). To conclude, we have four different queries: *OriginQuery*, *IssueQuery*, *MergeQuery* and *WebQuery*.

To get a corpus from TREC API, we submit *OriginQuery*, *IssueQuery* and *MergeQuery* to TREC-API to obtain three retrieval result sets. We merge the three candidate result sets and filter the same tweet to generate a corpus for the re-ranking algorithm. We name it *OriginCorpus*. Regarding to the importance of the URLs in tweet, we collected all the external URLs contained in *OriginCorpus* and extracted their title information for our document expansion process [4]. Note that web pages might be deleted as time elapses, we have only crawled a portion of the external URL set. We name the title information corpus *TitleCorpus*. When

[3]http://developers.google.com/web-search

adding the title information to the original corpus, we name the newly generated corpus *DocExCorpus*. After preprocessing, we will have three different corpora: *OriginCorpus*, *TitleCorpus* and *DocExCorpus*.

Now, we can generate lots of features via combining different queries, corpora and retrieval models. In our method, 120 features ($4 \times 3 \times 5 \times 2$) have been generated. Here, 4 stands for query count, 3 stands for corpus count, 5 stands for retrieval model count and 2 stands for whether to use pseudo relevance feedback or not.

Note that we label the corpus generated by TREC as *API api corpus*. Besides, we also crawled a local copy of tweets from 1 February,2013 to 31 March,2013 via Tweet API, and name it *local corpus*. That means we could generate another 120 features on local corpus and we use all of them in our PKUICST3 run.

2.3.3 Document Quality Features

Unlike semantic score features and semantic expansion features which are query-biased, document quality features are tended to estimate the quality of a tweet. Some specific features of social network services (SNS) can be used to measure the quality and potential popularity in the entire social network. Based on the assumption that users prefer those tweets that are related with their query or popular in the social network, we can conclude the following features:

- **Time Difference** (*TimeDiff*): this feature represents the time difference between the post time of the tweet and the query issue time. A short time difference usually indicates the highly temporal relevance between the tweet and the query.
- **Mention Count** (*MentionCnt*): '@' symbol followed with a user's screen name stands for mentions and replies. A tweet with more '@' means this tweet may attract more persons' attention.
- **Hashtag Count** (*HashtagCnt*): '#' symbol (i.e. hashtag) is used for organizing tweets into a particular topic. A symbol '#' marks the tweet as belonging to a particular topic.
- **Shortened URL Count** (*URLCnt*): the number of shortened URLs may imply the tweet importance since URLs are likely to provide additional information for the origin tweets.
- **Word Count of Tweet** (*WordCnt*): this feature represents the number of terms in a tweet (after stopword removal) and it may suggest the quality of tweets since longer tweet is likely to be more informative.
- **Length of Tweet Text** (*TweetLen*): this feature represents the length of tweet text (after stopword removal) and it may suggest the quality of tweets since longer tweet is likely to be more informative.

3. TTG TASK

3.1 System Overview

Our approach for TTG task mainly contains two steps: (1) retrieve adequate relevant documents for each query, and (2) utilize clustering algorithm to eliminate redundant tweets. The architecture of our system is shown in Figure 3. Search results are clustered so that tweets about the same/similar topic are grouped together, and for each cluster only the informative tweets are kept.

The submitted results for ad hoc search results (i.e. the run labeled as PKUICST3) are used as the input source for our TTG system, and we apply two strategies to choose the relevant tweets for each query. One strategy is to select the top N tweets whose ranking scores are greater than a threshold score, which is determined by preliminary experiments on the training topics. Another strategy is to select the top ranked tweets manually for each query. After selecting the relevant tweets, two clustering algorithms are adopted in our system, one utilizing the star clustering algorithm and the other taking classic hierarchical clustering algorithm. Finally, we choose the most representative tweet from each cluster for each topic as the final result of our TTG system.

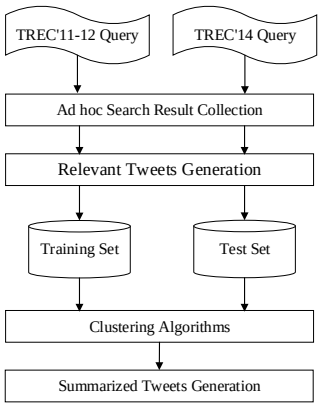

Figure 3: TTG System Framework

3.2 Clustering Algorithms

3.2.1 Star clustering algorithm

Given a query Q, star clustering algorithm first constructs a pairwise similarity graph on the top N retrieved results based on the *Vector Space Model*. For each pair of tweets d_i and d_j, their similarity score is computed by using cosine score of their corresponding TF vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, that is

$$sim(d_i, d_j) = cos(\mathbf{v}_i, \mathbf{v}_j) = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{|\mathbf{v}_i| \cdot |\mathbf{v}_j|} \quad (5)$$

$G = (V, E)$ is a similarity graph constructed by using a similarity threshold parameter σ. The top N documents compose vertex collection of G. If the similarity score between d_i and d_j is no less than σ, there would be a weighted edge connecting them. Star clustering algorithm [7] based on graph G is then utilized to generate the summarized tweet timeline as described in **Algorithm** 1. σ is the threshold parameter to determine whether document d_i and d_j have an edge in graph G. A large σ enforces that the connected tweets have high similarities, and thus the clusters tend to be small. Each cluster is star-shaped, and we treat the center document as the most representative tweet for the whole cluster.

Algorithm 1 star clustering algorithm

Input:
G = (V, E): vertex-weighted undirected graph.
Output:
Centers of star-shape clusters set S.

```
1: S = ∅
2: T = ∅
3: while T != V do
4:    v* = nil
5:    maxDegree = -1
6:    for v ∈ V - T do
7:       degree(v) = ‖ {v′ | (v′, v) ∈ E} \ T ‖
8:       if maxDegree < degree(v) then
9:          v* = v
10:         maxDegree = degree(v)
11:      end if
12:   end for
13:   S = S ∪ {v*}
14:   T = T ∪ {v″ | (v″, v*) ∈ E}
15: end while
16: return S
```

3.2.2 *Hierarchical clustering algorithm*

Given a query Q, we first get relevant tweet collection from the input source. Here we say a tweet is relevant when its score computed by the learning to rank component is greater than a score threshold α, then we apply the agglomerative hierarchical clustering algorithm with parameter β which controls the clustering terminal condition to generate the summarized tweet of each cluster. The algorithm is shown in **Algorithm** 2.

Algorithm 2 hierarchical clustering algorithm

Input:
Relevant tweet collection R
Cluster merging threshold β
Output:
Clusters collection C

```
1: C = {R_1, R_2, ··· , R_n}
2: repeat
3:    (C_i, C_j, MinDistance) ← GetMinDistancePair(C)
4:    MergeCluster(C_i, C_j)
5: until (MinDistance < β)
6: return C
```

Note that the distance between cluster c_i and c_j is computed by the following equation:

$$Distance(c_i, c_j) = 1 - sim(\underset{d_m \in c_i}{avg}\ (d_m),\ \underset{d_n \in c_j}{avg}\ (d_n)) \qquad (6)$$

where $sim(d_i, d_j)$ is the cosine similarity score between document d_i and d_j, and $\underset{d_m \in c_i}{avg}\ (d_m)$ is the average document in cluster c_i which has the whole term and average term frequency. Finally, we choose documents with the highest ranking score in each cluster as the summarized tweets.

4. RESULT ANALYSIS

Table 1 show the retrieval performance of our submitted four runs for ad hoc search task. The primary evaluation metric for this year's ad hoc search task is MAP (Mean Average Precision). Among all the runs, PKUICST1 uses learning to rank framework and adopts API corpus as candidate and API features as feature space; while PKUICST2 adopts local corpus as candidate and local features as feature space. PKUICST3 uses API corpus as candidate and API features plus local features as feature space. Unlike the previous three runs, PKUICST4 is an unsupervised run, which uses a language modeling framework with pseudo relevance feedback. More specifically, for the query modeling, we first combine *IssueQuery* and *WebQuery* with a interpolating coefficient. Then the combined query is further updated with the simple mixture model [8]. For the document modeling, we use the empirical word distribution on *DocExCorpus*, and choose Dirichlet smoothing method for model estimation.

From the table, we can observe that runs using learning to rank framework have a better retrieval performance than that only adopts language model (i.e. PKUICST4). Meanwhile, under learning to rank framework, runs using api corpus features (i.e. PKUICST1 and PKUICST3) perform better than the run just using local corpus features (i.e. PKUICST3) in terms of MAP score.

Table 1: Performance of submitted runs for ad hoc search

Run ID	MAP	P@30
PKUICST1	0.5834	0.7242
PKUICST2	0.5648	**0.7279**
PKUICST3	**0.5863**	0.7224
PKUICST4	0.5422	0.6958

For the TTG task, the primary evaluation metrics are unweighted_recall, weighted_recall (i.e. recall^w) and precision. The run TTGPKUICST1 applies star clustering method with tuned parameter $\sigma = 0.7$ and $N = 200$, while TTGPKUICST3 uses manually selected top N documents for each query. Both TTGPKUICST2 and TTGPKUICST4 apply hierarchical clustering method with distance threshold $\beta = 0.3$. Besides, the former one adopts score threshold $\alpha = 4.5$ to select relevant tweets while the latter one employs manually selected top ranked N tweets for each query.

Table 2 shows recall (unweighted and weighted), precision, and F_1 (unweighted and weighted) scores of different runs. Note that the ^{w} superscript indicates the weighted variant of the metric. The weighted version of the metrics attempts to account for the fact that some semantic clusters are (intuitively) more important than others. We can observe from the table that TTGPKUICST1 shows significant superiority in terms of unweighted_recall and weighted_recall over other runs, while it performs poorly in terms of precision and F_1 values. TTGPKUICST2 performs better in precision and F_1 values compared with other runs. Besides, TTGPKUICST3 and TTGPKUICST4 have medium and stable performance over all metrics. Both of them adopt manually selected top N parameter for each query.

Note that we utilize the ten subtopics' ground truth, which are provided by the official organization as our training set, to tune the threshold parameters for star clustering and hierarchical clustering algorithms. In the training set, there's no need to judge which tweet is relevant to the given query; while in the test set, our system has to determine how many retrieved tweets should be regarded as relevant tweets. Thus, the trained parameters may have

Table 2: Performance of submitted runs for TTG

Run ID	Auto/Manual	recall	recall^w	precision	F_1	$F_1{}^w$
TTGPKUICST1	Auto	**0.5221**	**0.7016**	0.2682	0.2691	0.3276
TTGPKUICST2	Auto	0.3698	0.5840	**0.4571**	0.3540	**0.4575**
TTGPKUICST3	Manual	0.4849	0.6583	0.3635	0.3496	0.4062
TTGPKUICST4	Manual	0.5174	0.6615	0.3664	**0.3579**	0.4057

a certain amount of deviation with optimal parameters for the test set. This may lead to the low precision of our four runs as it is hard for the system to trade off between the recall and precision. Further investigation and experiments are required to solve this issue.

5. CONCLUSION

In this paper, we present our systems for TREC 2014 Microblog track. In the ad hoc search, we apply a learning to rank framework which utilizes not only the various content relevance of a tweet, but also the quality of a tweet. In the TTG task, we apply some traditional clustering algorithm, i.e. hierarchical and star clustering on the retrieved tweets from ad hoc search task. Experimental results show the effectiveness of our systems for both tasks.

6. ACKNOWLEDGMENTS

The work reported in this paper was supported by the National Natural Science Foundation of China Grant 61370116.

7. REFERENCES

[1] Yajuan Duan, Long Jiang, Tao Qin, Ming Zhou, and Heung-Yeung Shum. An empirical study on learning to rank of tweets. In Chu-Ren Huang and Dan Jurafsky, editors, *COLING*, pages 295–303. Tsinghua University Press, 2010.

[2] Yajuan Duan, Long Jiang, Tao Qin, Ming Zhou, and Heung-Yeung Shum. An empirical study on learning to rank of tweets. COLING '10. ACL, 2010.

[3] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[4] Feng Liang, Runwei Qiang, and Jianwu Yang. Exploiting real-time information retrieval in the microblogosphere. In *Proceedings of the 12th ACM/IEEE-CS joint conference on Digital Libraries*, pages 267–276. ACM, 2012.

[5] Iadh Ounis, Craigand Macdonald, Jimmy Lin, and Ian Soboroff. Overview of the TREC-2011 Microblog Track. In *Proceedings of TREC 2011*, 2012.

[6] Runwei Qiang, Feng Liang, and Jianwu Yang. Exploiting ranking factorization machines for microblog retrieval. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1783–1788. ACM, 2013.

[7] Xuanhui Wang and ChengXiang Zhai. Learn from web search logs to organize search results. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 87–94. ACM, 2007.

[8] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.