Improving Tweet Timeline Generation by Predicting Optimal Retrieval Depth

Maram Hasanain[1], Tamer Elsayed[1], and Walid Magdy[2]

[1] Computer Science and Engineering Department
College of Engineering, Qatar University, Doha, Qatar
{maram.hasanain,telsayed}@qu.edu.qa
[2] Qatar Computing Research Institute, Doha, Qatar
wmagdy@qf.org.qa

Abstract. Tweet Timeline Generation (TTG) systems provide users with informative and concise summaries of topics, as they developed over time, in a retrospective manner. In order to produce a tweet timeline that constitutes a summary of a given topic, a TTG system typically retrieves a list of potentially-relevant tweets over which the timeline is eventually generated. In such design, dependency of the performance of the timeline generation step on that of the retrieval step is inevitable.
In this work, we aim at improving the performance of a given timeline generation system by controlling the depth of the ranked list of retrieved tweets considered in generating the timeline. We propose a supervised approach in which we *predict* the optimal depth of the ranked tweet list for a given topic by combining estimates of list quality computed at different depths.
We conducted our experiments on a recent TREC TTG test collection of 243M tweets and 55 topics. We experimented with 14 different retrieval models (used to retrieve the initial ranked list of tweets) and 3 different TTG models (used to generate the final timeline). Our results demonstrate the effectiveness of the proposed approach; it managed to improve TTG performance over a strong baseline in 76% of the cases, out of which 31% were statistically significant, with no single significant degradation observed.

Keywords: Tweet summarization · Microblogs · Dynamic retrieval cut-off · Query difficulty · Query performance prediction · Regression.

1 Introduction

Coping with a flood of user-generated content about ongoing events through the online social media is getting more challenging over time. With several trending topics of interest that are active simultaneously, losing track of some of them is sometimes inevitable due to the large amount of posts compared to the limited time. One potential solution is to have the ability to get a retrospective timeline of posts that cover trending topics or events; Tweet Timeline Generation (TTG) systems aim at addressing this problem [14].

TTG task is typically query-oriented. The user provides a query representing the topic of interest and requires the TTG system to provide a list of tweets that were posted prior to query time and that are both *relevant* to the topic and *non-redundant*. This construction of the problem suggests a natural design of a TTG system that consists of two consecutive steps. First, it *retrieves* a ranked list of tweets that are potentially-relevant to the topic (called the *retrieval* step) and then *generates a timeline* of non-redundant tweets out of that list (called the *timeline generation* (TG) step). This design, in turn, imposes a natural dependency of the quality of the generated timeline on the quality of the retrieved list of tweets. Additionally, an important decision that a TTG system usually makes is how many retrieved tweets (or in other words, which depth of the retrieved ranked list) to start the timeline generation step with. Out of **13** teams participated in the first offering of the TTG task at TREC-2014, at least **10** teams[3] have used this design and **7** of them have used a static (i.e., fixed) depth (or rank cutoff) of the retrieved tweets over all queries [14].

Figures 1 and 2 show how the performance of an example clustering-based TTG system is sensitive to the depth of the retrieved list of tweets. Performance is measured using weighted F_1 (denoted by wF_1) used as the official evalaution measure of the TTG task at TREC-2014 [14].

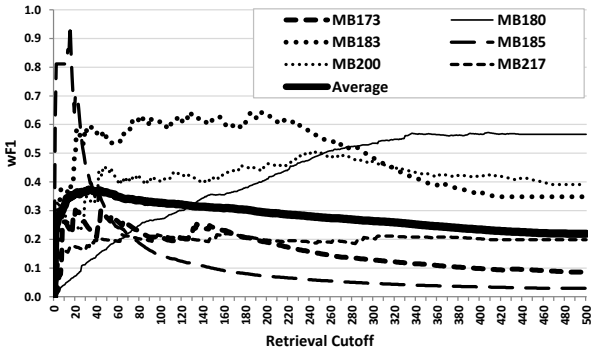

Fig. 1. TTG performance using different cutoffs over 6 TREC-2014 queries.

Figure 1 demonstrates this for 6 different TREC-2014 queries and for the average performance over all queries as well. It shows that different queries behave differently in terms of the effect of changing retrieval depth on TTG perfromance.

Given 55 TREC-2014 queries, Figure 2 shows how an optimal per-query rank cutoff can improve the performance over a static one by comparing the performance of two oracle TTG systems: the first used the best global static cutoff (i.e., a fixed cutoff over all queries that maximizes the average performance, which happened to be at depth 33), while the other used an optimal cutoff per query

[3] No published work on the system design of the remaining 3 teams.

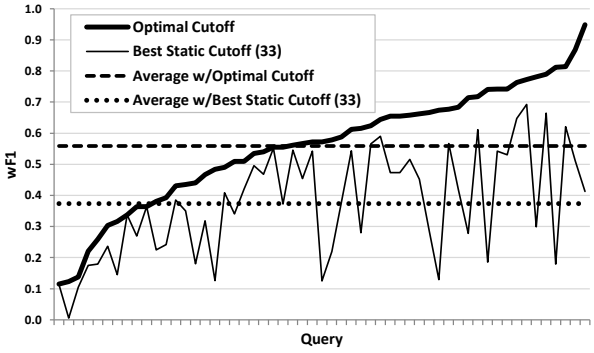

Fig. 2. Effect of using Static vs. dynamic cutoffs on a TTG system performance.

(i.e., a different cutoff per query that maximizes the performance of each query separately). The figure indicates that it is possible to achieve large improvements, reaching 50%, by just dynamically selecting the right retrieval cutoff per query, without any changes to neither the retrieval nor the TG components of the system.

Motivated by the above observations, we address the problem of improving the performance of a *given TTG system* by controlling the depth of the retrieved list of tweets. We propose to tackle the problem by learning a regression model that predicts optimal list depth needed to optimize the TTG performance. The model is learned over features that estimate the retrieval quality at different depths of the list. The problem of estimating the performance of a retrieval system given a query, called query performance prediction (QPP), has been studied extensively [3] and had recently showed promising results in microblog search [10, 18]. In this work, we leverage QPP techniques to predict a suitable retrieval cutoff for a TTG query. To our knowledge, this is the first study that leverages QPP for *improving* TTG or (more generally) tweet summarization systems.

Our contribution is two-fold. First, we showed that the performance of TTG systems is highly sensitive to the depth (and thus the quality) of the retrieved list of tweets over which the timeline is generated. Second, we proposed a learning framework that leverages QPP to improve the overall performance of any typical TTG system that starts with the retrieval step.

The rest of the paper is organized as follows. In section 2, we summarize the related work. We define the problem in section 3. The proposed approach is introduced in section 4. The experimental results are presented and discussed in section 5 before we conclude and give some directions of future work in section 6.

2 Related Work

Several research studies have targeted the TTG problem and the more general tweet summarization problem; many were part of the TTG task in TREC 2014

microblog track. There were also studies that investigated the performance prediction of summarization systems. We touch upon those related directions in this section.

2.1 Tweet Timeline Generation

Lv et al. [16] designed a TTG system that first applies hierarchical clustering on the top k tweets from a ranked list retrieved for a query. The timeline is then composed of the highest-scoring tweet from each cluster. Value of k was determined differently per query using a *static* retrieval score threshold. Our method allows both different depth and score cutoffs across queries.

Xu et al. [22] applied set-based novelty detection over the top k tweets retrieved. A tweet is added to the novel set (and timeline) if its similarity with any of the tweets in a sequentially-updated novel tweet set is below a threshold. Magdy et al. [17] used 1NN clustering of the top k tweets to generate a timeline. Their results show that the choice of the value of k can affect the TTG performance. Similarly, Xu et al. [22] tuned the parameter k for their TTG system, indicating that it had an effect on the performance.

Xiaohui et al. [4] also used the idea of clustering in TTG, but looking at tweets in a different way. They compute what they call a sequential pattern over each tweet in an initially retrieved list of tweet. The pattern captures term co-occurrence and semantics in the tweet. Once patterns are computed, the system clusters tweets with similar patterns together and select the tweet with highest retrieval score from each cluster to be added to the timeline.

2.2 Tweet Summarization

Shou et al. [19] employed an online incremental clustering algorithm with data structures designed to maintain important cluster information as the tweet stream evolves. Their system allowed for creating summarizes in two modes: online where summaries are created based on current clusters in memory, and historical which creates summaries based on history of clusters maintained in a data structure called Pyramidal Time Frame. In both modes, the system uses an algorithm that constructs a cosine similarity graph between tweets in all clusters then applies the LexRank method [8] to select most novel and central tweets to add to the summary.

In a more recent work, Chen et al. [5] worked with tweets retrieved by a search model which is similar to TTG but for the problem of tweet summarization. In their system, they classify tweets in the full list into genres and proceed to summarize tweets in each genre. They re-enforce the per-genre list of tweets by retrieving documents from the Web using a search query composed of terms selected from that list of tweets. Each terms in the tweet list is weighted using a measure that focuses on the authority of authors of tweets in which this term appeared. Once those Web documents are retrieved, they are split into sentences and added to the list of tweets (i.e., creating artificial tweets). A graph-based

summarizer is then applied and top-scoring sentences/tweets are selected to create the summary.

2.3 Predicting Summarization Performance

In another direction, Louis and Nenkova investigated the prediction of summarization performance in the context of classical summarization tasks. They attempted to use predictors computed on input documents to predict the performance of summarization systems [15]. Some of these predictors are usually used in predicting query performance in adhoc search tasks. They ran experiments using single- and multi-document summarization and evaluated their approach over a large set of training/testing datasets. Their results showed promising correlation between predicted and actual summarization system performance. This encouraged us to consider query performance prediction in the context of TTG, not to predict TTG performance but to improve it by predicting the optimal cutoff of a retrieved ranked list of tweets to start with.

Another line of studies investigated finding the optimal cutoff for the ranked list of results, but for the purpose of effective re-ranking techniques [12, 2]. However, that work focused on finding the optimal *global* cutoff over all queries. In our case, we predict the optimal cutoff *per* query.

3 Problem Definition

Given a query q posted at time t_q and a tweet collection C, TTG aims at generating a timeline T of *non-redundant* tweets that are *relevant* to q and posted prior to t_q. A TTG framework is usually composed of a retrieval component (represented by a *retrieval model*) that provides a ranked list R of tweets that are potentially-relevant to q, and a timeline generation (TG) component (represented by a *TTG model*) that accepts the list R and generates the tweet timeline T extracted from the top k tweets in R. We address the problem of improving the quality of T generated by *a given TTG system* by optimizing the cutoff value k.

4 Approach

We formulate the problem as a learning problem. Given a query q and, a ranked list R of tweets retrieved by a retrieval model, and a TTG model, we aim to learn a *regression* model that estimates (or *predicts*) a cutoff value k applied to R that is needed to optimize the TTG performance for q. k determines depth of R to be used in generating the timeline.

4.1 Features

To train the regression model, we propose to leverage the idea of query performance prediction (QPP). We compute a predictor for the query at m different

cutoff (i.e., depth) values applied to R, resulting in m predicted values that together constitute the feature set. Each predicted value (i.e., feature) is an estimation of the retrieval quality for q at the corresponding cutoff. Similarly, a feature vector can be generated for each query in a query set Q, yielding a set of Q feature vectors generated using the same retrieval model. Moreover, since different retrieval models can theoretically retrieve different ranked lists of tweets given the same query, the regression model can be trained using feature vectors generated using different retrieval models, which results in a larger set of feature vectors.

Query performance predictors are usually computed using a list of documents retrieved in response to the query using a retrieval model [3]. Several predictors were proposed in microblog search context [18] in addition to those proposed in other domains like news and Web search [6, 7, 20].

We experimented with **10 predictors** including the most effective ones in microblog search, in addition to effective predictors typically used with non-microblog collections. We selected microblog-specific predictors computed based on the following two measures of the topical-focus of R_k, the list composed of the top k tweets in R. Each measure is computed *per tweet* in R_k [18]:

1. Query Terms Coverage (QTC): QTC computes the *coverage* of query terms in the tweet, i.e., the number of query terms appeared in the tweet, and normalize it by the length of the query.
2. Top Terms Coverage (TTC): Given the n most frequent terms in R_k, TTC measures the coverage of these terms in the tweet normalized by n.

Once a measure is computed over each tweet in R_k, we compute mean, median, lower percentile, and upper percentile of QTC/TTC values over all tweets in R_k. Each one of these statistics represent a predictor. We also experimented with variants to these predictors using inverse document frequencies (IDF) of terms when computing the coverage.

As for typical predictors, we used the normalized query commitment (NQC) due to its reported effectiveness over different test collections [20].

4.2 Retrieval Models

We used retrieval approaches covering a large spectrum of effective techniques usually used in microblog search. We group these approaches into the following main groups:

- Standard query-likelihood (QL).
- Query Expansion (QE) models based on Pseudo Relevance Feedback (PRF).
- QE that benefits from web resources to select the expansion terms (QEW).
- Learning-to-rank (L2R)-based models. L2R models can also be combined with other models such as QE.
- Temporal (TEMP) models that emphasize temporality of the data (tweets) and the adhoc search task when performing retrieval.

In total, we used **14** retrieval models. We acquired ranked results (i.e., tweets) retrieved by these models using the 55 queries and dataset provided by the TREC-2014 TTG task (further details in section 5) from 3 participated teams [22, 17, 9]. We evaluated the performance of the models using mean average precision (MAP), which is the commonly-used measure to evaluate microblog adhoc search [14]. MAP was computed over the top 500 tweets per query. We summarize the models used in Table 1.

Table 1. Summary of retrieval models used

Group	ID	MAP	Group	ID	MAP
QL	QL1 [22]	0.385	QE+L2R	QEL [17]	0.470
	QL2 [9]	0.398		HQEL [17]	0.482
QE	QE1 [17]	0.464		WQEL [22]	0.497
	QE2 [9]	0.466		WQETL [22]	0.571
	QE3 [9]	0.456	TEMP	TDC [9, 13]	0.406
	QE4 [9]	0.490		TRM [9, 11]	0.445
QEW	HQE [17]	0.477			
	WQE [22]	0.485			

4.3 TTG Models

We worked with TTG models selected from existing literature based on their reported effectiveness, while attempting to diversify the types of considered models. We considered models based on two main concepts:

- Topical Clustering. TTG and tweets summarization systems based on topical clustering were effective in related studies [9, 17, 19]. These models create topical clusters for the input tweets assuming that each cluster reflects a sub-topic of the main topic. Some tweets from created clusters are selected to form the timeline based on several factors including: a) which clusters to represent in the timeline, b) number of tweets to select from each cluster and c) a selection criterion that minimizes redundancy in the timeline.
- Temporal clustering [1]. It groups tweets of the input set (viewed as a stream) considering temporal signals, usually extracted based on posting time of tweets. The timeline is generated based on selecting tweets from these clusters considering the same factors as in topical clustering.

We implemented and experimented with the following effective, existing TTG models. Specifically, we implemented *1NN* and *Centroid* which were the top **second** and **fourth** TTG systems (respectively) in TREC-2014 TTG task of the microblog track. Additionally, we implement a temporal TTG system.

- *Centroid* [9]: This model incrementally clusters the input tweet list by computing similarity between the tweet and centroids of clusters updated with

each new tweet. The tweet with the maximum retrieval score is used as the centroid of a cluster and it is included in the final timeline.
- *1NN* [17]: Similar to *Centroid*, tweets are incrementally clustered, but model only adds a tweet to a cluster if its similarity to any tweet in a cluster exceeds a threshold, earliest tweet in each cluster is added to the timeline.
- *Z-Score* [1]: This is a model that considers temporal signals in tweets. The model creates fixed-length time buckets of tweets given the ranked list sorted chronologically. Each term in each bucket is scored using the Z-Score–a measure designed to help detect spiking terms in a bucket. A tweet in a bucket is scored by summing the Z-Scores of all of its terms and the tweet with the maximum score is included in the timeline.

4.4 Regression Models

We combine predictors computed at different cutoffs using Weka's[4] implementation of two regression models: typical linear regression and the M5P algorithm that is based on generating model trees [21] . For the M5P model, we experimented with both pruned and un-pruned trees.

5 Experimental Evaluation

5.1 Experimental Setup

Dataset In our experiments, we used TREC-2014 microblog track test collection, which includes access to Tweets2013 of 243 Million tweets and 55 queries [14]. For simplicity, we assume that we experiment with H adhoc retrieval models, T TTG systems, and Q queries. Though the dataset used is relatively small, we increase the size of training examples in our ground truth by using a large set of adhoc models as discussed next.

Generating Ground Truth To generate our ground truth, we pre-identified the optimal retrieval cutoff for each query for each retrieval model by changing the cutoff from 1 to 500 (with step 1 with cutoffs 1-100 and step 10 with 110-500[5]), and identifying the one maximizing TTG performance. We repeat that for each TTG system. The optimal cutoff values represent the target function that the regression model is learning. Overall, the ground truth includes $T * H * Q$ samples (i.e., feature vectors).

We adopted weighted F_1 (denoted by wF_1) performance evaluation measure that was the official measure in TREC-2014 [14]. wF_1 combines precision and recall of the TTG system over a set of semantic clusters of relevant tweets to query q. The retrieved clusters are weighted by the number of tweets in each

[4] http://www.cs.waikato.ac.nz/ml/weka/

[5] We observed that TTG performance is less sensitive to change in list depth with large cutoffs

cluster and the tweets themselves are weighted by their relevance grade where "relevant" tweets get a weight of one and "highly-relevant" tweets get a weight of two. Additionally, we report overall system performance by averaging per topic wF_1 over all queries to compute average wF_1, which is a more accurate way than the one used to compute reported TTG results in [14].

Training and Testing Data For training and testing our regression model, we adopted *leave-one-query-out* cross validation. For each TTG system, we train our model on $Q-1$ queries and test it on the remaining unseen one query. Since we have H retrieval models, for each query, we train a regression model using $(Q-1)*H$ training samples and test it over $1*H$ testing samples. The trained model is used to *predict* the cutoff value for each unseen sample, which is eventually used by the TTG system to generate a corresponding timeline. We repeat that process Q times.

Baseline We compare the performance of our proposed approach with a baseline that applies one optimal *static* cutoff to all queries. This baseline follows a similar approach used by the best team at TREC TTG task to select the list depth by learning a static *score cutoff* over training queries [16]. Our baseline system learns the optimal static cutoff using a leave-one-query-out approach that is similar to the one described above for the regression model. It is trained over the same training set (i.e., $Q-1$ queries) used to train the regression model by changing the cutoff in the same way as above and picking the cutoff that maximizes the average TTG performance while using it for all queries in the training set. We then apply the learned cutoff to the remaining testing query. This process is followed independently on each retrieval model and on each TTG system. We believe this is a strong baseline as it benefits from the training data to learn an optimal, but static, retrieval cutoff.

Statistical-significance testing in our experiments was performed using two-tailed paired t-test, with $\alpha = 0.05$.

5.2 Results and Discussion

We studied 10 sets of features (one for each predictor) and two different regression models. Additionally, using both regression models, we also attempted to combine sets of features in an attempt to combine predictors used, but that resulted in poorer performance with some TTG systems compared to using single predictors. We suspect this is because of the small training/testing dataset we have, impeding learning a regression model over such large set of features.

Due to space limitation, we only report the results of the best performing setup using pruned **M5P** model trees and the feature set based on the IDF-variant of lower percentile of TTC values described in section 4.

Averaging percent-improvement that our method achieved over the baseline on all queries and retrieval models, our proposed approach improved for all of the 3 TTG models. The overall percent-improvement ranges from 3.2% with the

Z-Score model to 6.8% with *1NN* model; surprisingly, those models were the worst and best performing respectively, according to the baseline results, among the models we used.

Table 2 shows improved wF_1 (denoted by wF_1^*) and the percent-improvement over baseline for each retrieval model used with each of the TTG models. In 32 out of 42 cases, our proposed approach improved over the baseline, reaching up to 17% increase in wF_1; 10 of those cases were statistically significant, while none of the cases where the performance dropped was statistically significant. Furthermore, the results demonstrate the strength of our method as it managed to improve both systems with low and high TTG performance.

Table 2. Baseline wF_1 for each TTG model with each retrieval model vs. improved wF_1 (wF_1^*) along with the percent-improvement over baseline. Bold improvements are statistically significant.

Retrieval	Centroid		1NN		Z-score	
Model	wF_1	wF_1^*(%)	wF_1	wF_1^*(%)	wF_1	wF_1^*(%)
QL1	0.3372	0.3783**(+12.2)**	0.3701	0.3697(−0.1)	0.3450	0.3204(−7.1)
QL2	0.3757	0.3809(+1.4)	0.3854	0.4102(+6.4)	0.3002	0.3373**(+12.4)**
QE1	0.3844	0.3898(+1.4)	0.3847	0.4117(+7.0)	0.3540	0.3754(+6.0)
QE2	0.3374	0.3779**(+12.0)**	0.3464	0.3814(+10.1)	0.2973	0.3405**(+14.5)**
QE3	0.3486	0.3659(+5.0)	0.3634	0.3673(+1.1)	0.3450	0.3204(−3.2)
QE4	0.3684	0.3774(+2.4)	0.3644	0.4057(+11.3)	0.3137	0.3587**(+14.3)**
HQE	0.3730	0.4069**(+9.1)**	0.3917	0.4282**(+9.3)**	0.3315	0.3752**(+13.2)**
WQE	0.3764	0.3748(−0.4)	0.3417	0.3774(+10.5)	0.3646	0.3621(−0.7)
QEL	0.3927	0.3928(+0.0)	0.3979	0.4105(+3.2)	0.3240	0.3801**(+17.3)**
HQEL	0.3962	0.4193(+5.8)	0.4089	0.4368(+6.8)	0.3709	0.3825(+3.1)
WQEL	0.3711	0.3672(−1.1)	0.3776	0.3954(+4.7)	0.3458	0.3602(+4.2)
WQETL	0.4068	0.3979(−2.2)	0.4011	0.4149(+3.4)	0.3965	0.3529(−11.0)
TDC	0.3777	0.3789(+0.3)	0.3845	0.4160(+8.2)	0.3561	0.3311(−7.0)
TRM	0.3326	0.3685(+10.8)	0.3219	0.3710**(+15.2)**	0.3129	0.2999(−4.2)

Finally, we went a step further and studied the relationship between the difference in performance (between the proposed approach and the baseline) and the optimal cutoff value per query. We compared the average optimal cutoff over H retrieval runs and the average percent-improvement over the baseline over the same H runs for different queries and per TTG system. The results showed that almost all queries of which performance was degraded have an average optimal cutoff value that is ≤ 100. Moreover, at least 9 of the top 10 degraded queries (i.e, the ones with largest degradation) have an average optimal cutoff value ≤ 50 in all TTG systems. This is logical as the error in low cutoffs has a larger effect on performance than in large cutoffs, because tweets at higher ranks are potentially more relevent and therefore missing them becomes more costly. Another possible reason is the general sensitivity of query performance predictors to the depth of the retrieved list. Since the predictors (used as features) were not tuned per

retrieval model, it might produce poor results at shallow lists in some cases. This indicates that more attention should be given to features at the first 100 cutoffs and possibly to a regression model that penalizes errors in queries of low optimal cutoffs than in those of higher cutoffs. We also notice that almost all queries of high optimal cutoffs (≥ 150) were improved in all TTG systems.

6 Conclusion and Future Work

In this work, we used query performance predictors to predict the optimal depth of retrieval ranked list to use with a TTG system. Our results showed the effectiveness of this method in improving performance of 3 sample different TTG systems across 14 different retrieval approaches. Out of 42 different cases, 32 were improved with 10 of them had significant improvement, while in only 10 cases TTG effectiveness was degraded but insignificantly.

For future work, more analysis of failure instances is needed especially for instances where optimal cut-offs are low. Other performance predictors can also be tried and we plan to experiment with more regression models. Another evident direction is to study how good the predictors are in predicting actual retrieval performance and how is that related to their performance in predicting optimal cutoff for TTG. With larger test collction (more importantly larger set of queries), extensive experiments can be conducted for more concrete results.

Acknowledgments This work was made possible by NPRP grant# NPRP 6-1377-1-257 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

1. Alonso, O., Shiells, K.: Timelines as summaries of popular scheduled events. In: Proceedings of the 22Nd International Conference on World Wide Web Companion. pp. 1037–1044. WWW '13 Companion (2013)
2. Arampatzis, A., Kamps, J., Robertson, S.: Where to stop reading a ranked list?: Threshold optimization using truncated score distributions. In: Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 524–531. SIGIR '09 (2009)
3. Carmel, D., Yom-Tov, E.: Estimating the Query Difficulty for Information Retrieval. Synthesis Lectures on Information Concepts, Retrieval, and Services 2(1), 1–89 (2010)
4. Chen, X., Tang, B., Chen, G.: BUPT_pris at TREC 2014 microblog track. TREC '14 (2014)
5. Chen, Y., Zhang, X., Li, Z., Ng, J.P.: Search engine reinforced semi-supervised classification and graph-based summarization of microblogs. Neurocomputing 152, 274–286 (2015)
6. Cronen-Townsend, S., Zhou, Y., Croft, W.B.: Predicting query performance. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 299–306. SIGIR '02 (2002)

7. Cummins, R.: Predicting query performance directly from score distributions. In: Salem, M.V.M., Shaalan, K., Oroumchian, F., Shakery, A., Khelalfa, H. (eds.) Information Retrieval Technology, pp. 315–326. No. 7097 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (Jan 2011)
8. Erkan, G., Radev, D.R.: Lexrank: Graph-based lexical centrality as salience in text summarization. Journal of Artificial Intelligence Research 22(1), 457–479 (2004)
9. Hasanain, M., Elsayed, T.: QU at TREC-2014: Online clustering with temporal and topical expansion for tweet timeline generation. TREC '14 (2014)
10. Hasanain, M., Malhas, R., Elsayed, T.: Query performance prediction for microblog search: A preliminary study. In: Proceedings of the First International Workshop on Social Media Retrieval and Analysis. pp. 1–6. SoMeRA '14 (2014)
11. Keikha, M., Gerani, S., Crestani, F.: Time-based relevance models. In: Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1087–1088. SIGIR '11 (2011)
12. Lan, Y., Niu, S., Guo, J., Cheng, X.: Is top-k sufficient for ranking? In: Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management. pp. 1261–1270. CIKM '13 (2013)
13. Li, X., Croft, W.B.: Time-based language models. In: Proceedings of the Twelfth International Conference on Information and Knowledge Management. pp. 469–475. CIKM '03 (2003)
14. Lin, J., Efron, M., Wang, Y., Garrick, S.: Overview of the TREC-2014 Microblog Track (Notebook Draft). TREC '14 (2014)
15. Louis, A., Nenkova, A.: Performance confidence estimation for automatic summarization. In: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics. pp. 541–548. EACL '09 (2009)
16. Lv, C., Fan, F., Qiang, R., Fei, Y., Yang, J.: PKUICST at TREC 2014 microblog track: Feature extraction for effective microblog search and adaptative clustering algorithms for TTG. TREC '14 (2014)
17. Magdy, W., Gao, W., Elganainy, T., Zhongyu, W.: QCRI at TREC 2014:applying the KISS principle for the TTG task in the microblog track. TREC '14 (2014)
18. Rodriguez Perez, J.A., Jose, J.M.: Predicting query performance in microblog retrieval. In: Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval. pp. 1183–1186. SIGIR '14 (2014)
19. Shou, L., Wang, Z., Chen, K., Chen, G.: Sumblr: Continuous summarization of evolving tweet streams. In: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 533–542. SIGIR '13 (2013)
20. Shtok, A., Kurland, O., Carmel, D., Raiber, F., Markovits, G.: Predicting query performance by query-drift estimation. ACM Transactions on Information Systems (TOIS) 30(2), 11:1–11:35 (2012)
21. Wang, Y., Witten, I.H.: Induction of model trees for predicting continuous classes. In: Proceedings of the 9th European Conference on Machine Learning Poster Papers. ECML '97 (1997)
22. Xu, T., McNamee, P., Oard, D.W.: HLTCOE at TREC 2014: Microblog and clinical decision support. TREC '14 (2014)