Light-weight, Conservative, yet Effective: Scalable Real-time Tweet Summarization

Reem Suwaileh, Maram Hasanain, Tamer Elsayed

Computer Science and Engineering Department
Qatar University
Doha, Qatar
{reem.suwaileh,maram.hasanain,telsayed}@qu.edu.qa

ABSTRACT

Microblogging platforms and Twitter specifically have become a major resource for exploring diverse topics of interest that vary from the world's breaking news to other topics such as sports, science, religion and even personal daily updates. Nevertheless, one by herself cannot easily follow her topics of interest while tackling the challenges that stem from the Twitter timeline nature. Among those challenges is the huge amount of posted tweets that are either not interesting, noisy, or redundant. Additionally, one cannot survive with manual techniques to summarize tweets related to topics that are discussed on the stream and are developed rapidly. In this paper, we tackle the problem of summarizing a stream of tweets given a pre-defined set of topics in the context of Qatar University's participation in TREC-2016 Real-Time Summarization (RTS) track. We participated in both push notification and e-mail digest scenarios. Given a set of users' interest profiles, our RTS system for push notifications scenario adopts a light-weight and conservative filtering strategy that monitors the continuous stream of tweets over a pipeline of multiple stages, while maintaining a scalable processing of a large number of interest profiles. For the e-mail digest scenario, we adopted a similar but even simpler approach. At the end of each day, a list of potentially relevant tweets is retrieved using a query of topic title terms that is issued against an index of all streamed tweets of that day. Our push-notification runs exhibited the best performance among all submitted automatic runs in the push notification task this year. Moreover, our best-performing email-digest run was the second-best among all submitted automatic runs in the email-digest task this year. However, the evaluation results show that the performance is still away from being adopted in practice.

1. INTRODUCTION

Twitter is one of the leading social media networks through which users post information as personal as their daily habits to updates and opinions on the world's breaking news. This rich and diverse stream of posts attracted users to turn to Twitter as a major source of information on ongoing topics and events. However, due to the continuous and enormous volume of the Twitter stream, manually looking for updates on topics of interests is a very daunting task for users. This necessitates the need for a real-time summarization system that automatically tracks events or topics of interest to users (perhaps millions of topics for millions of users) in parallel and generates a summary of representative on-topic tweets in real-time for each.

TREC '16 Gaithersburg, Maryland USA

Given the continuous tweet stream, a real-time summarization (RTS) system aims to monitor the online stream and identify the tweets that are relevant to a set of predefined interest profiles (representing the users' topics of interest) while taking their novelty and freshness into account. For instance, if a user is interested in following the updates on the "Brazilian Soccer League", the system should efficiently monitor the stream and capture the on-topic tweets including all aspects of the topic (e.g., results and standings) which might change over time. Accordingly, real-time summarization approaches should use simple and efficient approaches that can scale to follow multiple interest profiles in parallel. Most importantly, the RTS systems are expected to overcome many challenges that stem from the nature of tweets, such as sparsity and topic drift. The former challenge originates from the very short length of tweets. One way to tackle such a challenge is by enriching the tweet text by contextual terms. The latter challenge requires the system to cope with the changes of the topic over time. One possible solution to this challenge is to update the topic representation by terms from the topic's new aspects that are discovered over the live stream.

In this paper, we present our real-time summarization system as a participant in TREC-2016 Real-time Summarization Track. Given a live stream of tweets and a set of interest profiles of users that represent their topics of interest, the RTS track has two main scenarios: (1) Scenario A (push notification), in which the system is expected to push few (relevant and novel) tweets per day as notifications on the user's mobile phone for each topic, and (2) Scenario B (email digest), in which the system suggests a list of tweets that summarizes the topic over a period of time and sends it to the user as a periodic email digest for each topic. This year, the track organizers provided a broker through which participants can fetch the interest profiles and push the filtered tweet immediately when they system decides to elect one [2]. The interest profile is composed of three main fields: title (short query), description (1-2 sentences describing the information need), and narrative (a paragraph describing the information need).

For the push notifications scenario, we adopted a light-weight and conservative filtering strategy that listens to the

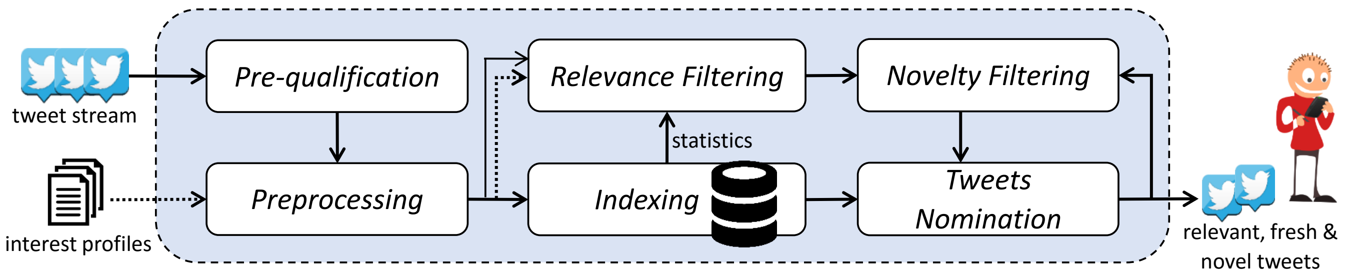

Figure 1: A high-level overview of the core real-time summarization system

English tweets over the 1% sample of the Twitter stream, and processes the incoming tweets successively. Our system processes the incoming tweets in a pipeline that involves preprocessing, pre-qualification, relevance filtering, novelty filtering, and tweet nomination. Our system is considered a pure "streaming" system as it adopts "one-tweet at-a-time" processing model to ensure the least possible latency in making pushing decisions. To alleviate the sparsity and topic drift problems, we followed the typical solution, that is topic expansion; we attempt two expansion methods: (1) over the local stream and (2) over Twitter search service. In the former, we extract the expansion terms from the potentially-relevant documents that are identified by the system. In the latter, we search Twitter online and extract the expansion terms from the top returned tweets. Surprisingly, according to our results, both expansion methods affect the system performance negatively; our best run is the one that does not perform any expansion.

For the email digest scenario, we adopted a similar but even simpler approach. At the end of each day, a list of potentially-relevant tweets was retrieved by searching the index of all streamed tweets of that day. We experimented with different ranking models with a static relevance threshold. The tweets go through similar novelty check to the one adopted in the push notifications scenario, and the surviving tweets are ranked based on their relevance scores before being added to the submitted digest. Our results show that a ranking algorithm that uses language modeling with Jelinek Mercer smoothing outperforms other models.

The remainder of the paper is organized as follows. We present the design of our systems in detail in Section 2. We then discuss our official TREC results in Section 3 followed by the drawn conclusions and some guidelines for the future work in Section 4.

2. APPROACH

Our system this year extends upon the system that we participated with in TREC-2015 Microblog Filtering track [5]. The system adopts a light-weight and conservative filtering strategy; it monitors and processes the stream of tweets following a pipeline of multiple stages, while tracking a large number of topics in a scalable manner. Each topic is represented by an *interest profile* consisting of three fields: a short *title* that describes information need, a *description* of one or two sentences and a *narrative* paragraph that articulates the information need in detail. This year, we only utilize the title to represent the topic in our system since our experiments over TREC-2015 test collection showed that adding the other two fields to the topic representation negatively affects the system performance [5].

In this section, we first present the core architecture of our RTS system that was leveraged in both scenarios, then we discuss each component in the system in detail for both scenarios.

2.1 System Design

Figure 1 depicts the high-level architecture of our system. As we indicated earlier, our system is a pure "streaming" system, in contrast to "micro-batching" systems. It adopts one-tweet-at-a-time processing model to ensure the shortest possible latency in making pushing decisions.

2.1.1 *Pre-qualification*

As the system gathers tweets from the Twitter stream, it ignores non-English tweets detected using the tweet language attribute provided by the Twitter API, and replaces all retweets by their original tweets. The incoming tweet is not eligible to pass to the subsequent pipeline components if it is a low-quality tweet. We define low-quality tweets as tweets with very few terms, too many hashtags or too many URLs. Based on this criteria, we filter out any tweet that has less than 5 terms or more than one URL or more than 3 hashtags. Furthermore, we drop any tweet that does not match at least one title term of any interest profile from being further processed.

2.1.2 *Preprocessing*

A tweet that survives the pre-qualification filter will be passed to the preprocessing stage where a sequence of steps are included: special characters (e.g., emoticon and symbolic characters) removal, stop-words removal, URL removal, and stemming. We also expand the tweet with the hashtags but as terms (i.e., without the '#' prefix), before it is examined against all interest profiles for relevancy.

2.1.3 *Indexing*

Since we acquire term statistics in other components of the system, we initialized the system with an index of a 5-day stream of tweets preceding the beginning of the evaluation period. The system also incrementally indexes all incoming English tweets during the evaluation period.

2.1.4 *Relevance Filtering*

We use the vector space model to represent each interest profile (title specifically) and each incoming tweet as a vector using *idf*-based term weighting scheme. We compute the term weights using the following equation:

$$w(t) = idf(t) = \log \frac{N - df(t) + 0.75}{df(t) + 0.75} \quad (1)$$

where N is the number of tweets indexed at the time of constructing the vector, and $df(t)$ is the document frequency of the term. We chose this term weighting function due to being light-weight (which is necessary for real-time and scalable systems) and also very similar to the standard *tf-idf* weighting function noticing that terms rarely appear more than once in a tweet due to the limited length (140 characters).

Once the incoming tweet is represented in the vector space, the RTS system computes its relevance score against each interest profile using the standard Cosine Similarity function. To efficiently compute relevance scores, an *in-memory* index of profile vectors is maintained to match an incoming tweet with interest profiles. The relevance model makes a threshold-based decision in which it considers a tweet with a similarity score above a relevance threshold τ_r as a *potentially-relevant* tweet to the corresponding profile.

2.1.5 Novelty Filtering

The RTS system should only pushes tweets that are *relevant* and *novel* to the user. Therefore, a novelty model has to be used to estimate the novelty for each potentially-relevant tweet. In our system, any potentially-relevant tweet of a profile has to be examined against all previously-pushed tweets for the corresponding profile to estimate its novelty before deciding to push it to the corresponding user. To estimate the novelty of a potentially-relevant tweet, we leverage a *lexical* similarity measure, that is a variant of Jaccard similarity, which computes the lexical overlap between the tweet and each pushed tweet for the corresponding profile as follows:

$$J'(Q,T) = \frac{|Q| \cap |T|}{max(|Q|, |T|)} \quad (2)$$

Where Q and T are the profile and the tweet term sets, and $|Q|$ and $|T|$ are their lengths (in terms) respectively. A tweet is considered novel if its similarity score with every pushed tweet is below a novelty threshold τ_n (i.e., it is different enough from any of the pushed tweets), otherwise, the system does not consider pushing it to the user.

2.2 Push Notifications Scenario

The push notifications scenario simulates a recommender system that sends pop-up messages to users on their mobile phones after capturing tweets that match their interests. The task design restricts the number of pushed tweets per profile to a maximum of 10 tweets per day to avoid overwhelming the users. Having such constraint on the number of tweets to push, the system should wisely select the best candidate tweets to elect to the user in a timely fashion. We explain next how we used tweet freshness to nominate tweets to be pushed for an interest profile.

2.2.1 Tweets Nomination

While tracking all interest profiles simultaneously and monitoring the tweets stream, the system maintains, for each of the interest profiles, a list of *candidate tweets* that contains the tweets that were found relevant and novel so far. The RTS system periodically nominates a tweet to push to the broker [4] for a topic if the system overtakes a silence period δ or it has already found l candidate (i.e., potentially relevant and novel) tweets for that topic. Before actually pushing a candidate tweet to the user of a specific profile through the broker, the system *re-ranks* tweets in the candidates list of that profile based on relevance and freshness using equation 3 below. The top tweet is then pushed to the user through the broker.

$$S(t) = S_r(t) * \frac{100 - (CurTime - time(t))}{100} \quad (3)$$

$S_r(t)$ is the relevance score of tweet t (computed using Cosine similarity as we discussed earlier), cur_time is the current system time (in minutes), and $time(t)$ is the tweet creation time (in minutes). The final nomination score $S(t)$ adopts a *linear decay* factor that linearly penalizes the tweets based on their posting time, hence favoring fresh tweets.

2.2.2 Profile Expansion

To cope with topic development over time, the system periodically enriches the topic representation for an interest profile using Rocchio's pseudo relevance feedback. The main idea is to periodically select k "relevant" terms from p tweets that are potentially-relevant to the topic, and add those terms to the topic representation.

We utilized two different sources of the expansion. The first is the list of potentially-relevant tweets as detected by the relevance filtering, and the second is the top resulting tweets from searching Twitter using the online search service[1]. For the latter, we used the topic title as a query and restricted the search date to the current date to get as fresh tweets as possible. The system also applies the same qualification and preprocessing rules to the tweets and deduplicates the result list. For both sources, terms of all candidate tweets are scored by adding up their *idf-based* scores as follows:

$$w_e(t) = n_R(t) * idf(t) \quad (4)$$

Herein, $w_e(t)$ is the score of the term t in the pseudo-relevant tweet set R, $n_R(t)$ indicates the number of tweets in R that contains t, and $idf(t)$ is the *idf-based* weight of t as computed earlier.

After scoring all terms, the top k terms, denoted as expansion terms, are added to the topic vector. To avoid topic drift, the topic vector is reset to the title terms (i.e., original vector) before each expansion, as shown below.

$$\vec{q}' = \vec{q} + \beta * \vec{e} \quad (5)$$

where $\vec{e}$ is the normalized vector of the k expansion terms, and β is a parameter used to restrict the influence of expansion terms on the new topic vector.

2.3 Periodic E-mail Digest Scenario

In this scenario, the RTS system is required to compile a daily list of a maximum of m tweets per interest profile and send it as an email digest to the user. For that, we adopted a similar but even simpler approach than the approach for push notification scenario. At the end of each day of the evaluation period, the system issues the title of the interest profile against the local tweet index that is incrementally updated over time. We experimented with three retrieval

[1]https://dev.twitter.com/rest/public/search

models: query-likelihood model with Dirichlet smoothing, query-likelihood model with Jelinek-Mercer smoothing, and a combination of both [1]. The system retrieves a list of $2m$ tweets and filters out the tweets that have a relevance score below a static relevance threshold τ_r. The survived tweets are then passed to the novelty filtering that is exactly similar to the push notification scenario. The tweets that pass the novelty checking are then ranked based on their relevance scores before sending the top m as a digest email to the user.

3. EXPERIMENTAL EVALUATION

In this section, we present the evaluation measures used to evaluate our RTS system in both scenarios, and the results of our official TREC-2016 runs. The RTS system for the push notifications scenario is allowed to push a maximum of $n = 10$ tweets per topic in each day of the evaluation period. If a system pushes more than 10 tweets, the extra tweets will be just ignored. For the other scenario, the system is expected to send a daily tweet list of a maximum of $m = 100$ tweets per topic.

3.1 Evaluation Measures

Three basic evaluation measures were used in push notifications scenario: the expected gain (EG) (which is the official measure), the normalized cumulative gain (nCG), and the Gain minus Pain (GMP) measures. All of them were computed per day per topic and then averaged per topic.

- **Expected Gain (EG):**

$$\mathrm{EG} = \frac{1}{N} \sum_{t \in P} G(t) \tag{6}$$

  P is the set of tweets that are pushed by the system and N is the number of those tweets (i.e., N must be ≤ 10 tweets). $G(t)$ is 0 if the tweet is judged as non-relevant, 0.5 if it is judged as relevant, and 1 if it is judged as highly-relevant. The measure also penalizes redundancy in pushed tweets using the semantic clusters (each contains a set of relevant tweets that are semantically similar to each other) that are released as part of the relevance judgments; once a tweet from a cluster is pushed, all upcoming pushed tweets from the same cluster are considered non-relevant.

- **Normalized Cumulative Gain (nCG):**

$$\mathrm{nCG} = \frac{1}{Z} \sum_{t \in P} G(t) \tag{7}$$

  Z is the maximum possible gain for that topic in that specific day based on all judged pushed tweets.

- **Gain minus Pain (GMP):**

$$\mathrm{GMP} = \alpha \sum_{t \in P} G(t) - (1 - \alpha) P \tag{8}$$

  P is the number of non-relevant tweets pushed by the system and α is a parameter to balance between gain and pain.

Unlike in TREC-2015 filtering track, this year, the freshness is not considered in evaluating the system output, hence the gain of a run is not subject to temporal penalty. However, the latency was reported separately for tweets that contribute to the gain using mean (MLT) and median ($MedLT$) latency measures. Those measures compute the difference between the pushing time of a tweet and the first tweet of the cluster which the pushed tweet belongs to (i.e., reference tweet).

As for the e-mail digest scenario, the daily e-mail digest is evaluated as a ranked list of tweets using **normalized discounted cumulative gain (nDCG)** computed at cut-off *10* of the list. The measure also penalizes redundancy in the same manner followed in the push notifications scenario.

For all evaluation measures in both scenarios, the score of a submitted run is the average of scores over all topics. Moreover, for all measures, there are two variations based on how the silent days (i.e., days when there were not any relevant tweets) are treated. The measures that have "1" as a suffix reward a system by a score of 1 if it kept quiet on silent days, while measures that have "0" as a suffix treat all systems the same way by assigning a score of 0 on silent days regardless of how they behaved.

3.2 Official Runs

In this section, we discuss our submitted runs for both scenarios in detail including the configuration and results. We used the test collection from the microblog track of TREC-2015 [3] to tune parameters of all our runs.

3.2.1 Push Notifications Scenario

We submitted three runs in this scenario. In all of them, we set both of the relevance and novelty thresholds τ_r and τ_n to 0.6 according to the experiments we conducted on TREC-2015 test collection.

- **QUBaseline** is the baseline run that does no expansion to the topic profile.

- **QUExpP** has a similar configuration to QUBaseline except that it uses pseudo relevance feedback to expand the profile *hourly*. It extracts the top $k = 2$ terms from the top $p = 20$ potentially-relevant tweets detected by the relevance filtering. We set the expansion term weight factor to be $\beta = 0.2$ to avoid the topic drift.

- **QUExpT** also performs *hourly* expansion, but using Twitter search API. It adds the top $k = 1$ term extracted from the top $p = 20$ pseudo-relevant tweets returned from Twitter live search. Similar to the previous run, we set the expansion factor β to 0.2.

Table 1 shows the results for our official runs for the push notifications scenario in comparison to the baseline run (i.e., YoGosling) [6] provided by the track organizers. The table also shows the performance of two other hypothetical runs denoted as *Median* and *Best*, which indicate the average of the best and median scores respectively per topic among all participated runs. These scores were provided by the track organizer for EG-1, nCG-1 and GMP.5 measures. Note that the unit of both the mean latency (MLT) and median latency (MedLT) is seconds.

The table clearly shows that our baseline run (QUBaseline) outperforms the other two runs and also the track baseline in terms of the official measure EG-1. It is interesting to

Table 1: Official TREC 2016 results of QU runs for the push notifications scenario. Best value per column is boldfaced.

Run	EG-1	EG-0	nCG-1	nCG-0	GMP.33	GMP.5	GMP.66	MLT	MedLT
QUBaseline	**0.2643**	**0.0321**	**0.2479**	0.0157	-0.1357	-0.0888	-0.0447	173843	62478
QUExpP	0.2519	0.0233	0.2413	0.0127	-0.1641	-0.1134	-0.0657	161403	56863
QUExpT	0.2552	0.0230	0.2455	0.0133	**-0.0986**	**-0.0647**	**-0.0329**	141163	46025
YoGosling	0.2289	0.0253	0.2330	**0.0295**	-0.6000	-0.4317	-0.2733	**120909**	**8718**
Median	0.2335	-	0.2303	-	-	-0.1049	-	-	-
Best	0.3816	-	0.4576	-	-	0.0388	-	-	-

note that topic expansion had a negative effect on the system performance in both expansion runs. This could have happened due to several reasons. One possible reason is that we did not exhaustively tune the expansion parameters. Perhaps performing expansion at a lower rate or using a different expansion factor β could have helped improve the overall system performance. Moreover, as topic expansion might cause drift in some topics, a further failure analysis of the topics that got drifted is needed to explore better methods of expansion term selection.

Compared to the other submitted runs, we notice that all our runs outperform the median for the measures EG-1 and nCG-1. Indeed, our best submitted run scored above the median over EG-1 measure in 19 topics out of the 56 evaluated topics. While there is a significant difference with the "oracle" best run, our best submitted run exhibited the best score over EG-1 measure for 15 topics. It is worth mentioning that our runs exhibited the best performance (according to the official evaluation measure, EG-1) among all submitted automatic runs (42 runs submitted by 19 participating teams) in this scenario this year.

3.2.2 E-mail Digest Scenario

We have also submitted three runs in this scenario. In all of them, we set the novelty threshold τ_n to 0.6.

- **QUDR8** retrieves tweets using a language model with Dirichlet smoothing while setting the smoothing factor μ to 2000. The relevance threshold τ_r is set to 8.
- **QUJM16** retrieves tweets using a language model with Jelinek-Mercer smoothing while setting the interpolation factor λ to 0.7. The relevance threshold τ_r is set to 16.
- **QUDRJM24:** ranks tweets using a scoring function that combines evidence from two retrieval models [1]: language model with Dirichlet smoothing ($\mu = 2000$) and Jelinek-Mercer smoothing ($\lambda = 0.7$). The relevance threshold τ_r is set to 24.

Table 2: Official TREC 2016 results of QU runs for the e-mail digest scenario. Best value per column is boldfaced.

Run	nDCG1	nDCG0
QUDR8	0.2344	0.0094
QUJM16	**0.2621**	**0.0300**
QUDRJM24	0.2558	0.0237
YoGoslingBSL	0.2352	0.0299
Median	0.1931	0.0325
Best	0.4427	0.2481

Table 2 shows our results for this scenario compared to the YoGosling baseline run provided by the track organizers [6]. The table shows that QUJM16 run (that retrieves tweets using a language model with Jelinek-Mercer smoothing) is the best performing run compared to all other runs. Overall, that run was ranked second among all submitted automatic runs in that scenario this year. We plan to work on improving this simple approach by experimenting with more effective retrieval models. Additionally, a more thorough parameter tuning is needed to set λ, and the novelty and relevance thresholds.

4. CONCLUSION AND FUTURE WORK

In this work, we presented a light-weight scalable real-time tweet summarization system as our participation in the real-time summarization track in TREC-2016. For both of the scenarios of the track (i.e., push notifications and e-mail digest scenarios) we followed a simple pipeline that includes multiple stages: pre-qualification, preprocessing, relevance filtering, novelty filtering, and tweets nomination. As for the push notification scenario, we experimented with different interest profile expansion approaches to test how we can enrich the topic representation with terms of interest to cope with topic development over time. Surprisingly, our results showed that the baseline run that does not perform any expansion outperformed the ones used expansion. As for the e-mail digest scenario, we used simple daily ad-hoc search over the index of tweets collected during a day to retrieve the set of potentially-relevant tweets that went through the same pipeline. Our results showed that a simple language modeling retrieval model with Jelinek-Mercer smoothing achieved the best performance compared to other runs. Overall, our push notification runs exhibited the best performance among all submitted automatic runs this year.

We plan to further perform failure analysis on all components of the system. We specifically plan to investigate the poor performance of runs that perform expansion by conducting experiments with different expansion techniques to update the topic profile over time. We also plan to experiment the effect of using dynamic thresholds on the system performance. Additionally, we are planning to study the effect of using different similarity features in both relevance and novelty filters such as the semantic and social features.

5. ACKNOWLEDGMENTS

This work was made possible by NPRP grant# NPRP 6-1377-1-257 and NPRP grant# NPRP 7-1313-1-245 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

6. REFERENCES

[1] E. A. Fox and J. A. Shaw. Combination of Multiple Searches. In *Proceedings of the Second Text REtrieval Conference*, TREC 2, pages 243–252, 1993.

[2] J. Lin. TREC 2016 track guidelines. http://trecrts.github.io/TREC2016-RTS-guidelines.html.

[3] J. Lin, M. Efron, Y. Wang, G. Sherman, and E. Voorhees. Overview of the TREC-2015 Microblog Track. In *Proceedings of the 24th Text REtrieval Conference*, TREC '15, 2015.

[4] J. Lin, A. Roegiest, L. Tan, R. McCreadie, E. Voorhees, and F. Diaz. Overview of the TREC-2016 Real-Time Summarization Track. In *Proceedings of the 25th Text REtrieval Conference*, TREC '16, 2016.

[5] R. Suwaileh, M. Hasanain, M. Torki, and T. Elsayed. QU at TREC-2015: Building Real-Time Systems for Tweet Filtering and Question Answering. In *Proceedings of the 24th Text REtrieval Conference*, TREC '15, 2015.

[6] L. Tan, A. Roegiest, C. L. Clarke, and J. Lin. Simple Dynamic Emission Strategies for Microblog Filtering. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, SIGIR '16, pages 1009–1012, 2016.