Exploiting Live Feedback for Tweet Real-time Push Notifications

Reem Suwaileh and Tamer Elsayed
Computer Science and Engineering Department
Qatar University, Doha, Qatar
{reem.suwaileh,telsayed}@qu.edu.qa

ABSTRACT

Twitter has been developed as an immense information creation and sharing network through which users post diverse information. Although a user would regularly check her Twitter timeline to stay up-to-date on her topics of interest, it is impossible to survive with manual topic tracking techniques while tackling the challenges that emerge from the Twitter timeline nature. Among these challenges are the big volume of posted tweets, noise (e.g., spam), redundant information (e.g., retweets), and the rapid development of topics over time. This necessitates the development of real-time summarization (RTS) systems that automatically track predefined topics of interest and summarize the stream while considering the relevance, novelty, and freshness of the selected tweets.

We tackle this problem as part of Qatar University's participation in TREC-2017 Real-Time Summarization (RTS) track. Our RTS system adopts a light-weight and conservative filtering strategy that monitors the continuous stream of tweets over a pipeline of multiple phases including pre-qualification, preprocessing, indexing, relevance filtering, novelty filtering, and tweets nomination. The system also exploits life (explicit) feedback to update profiles and pushing criteria (e.g., relevance threshold). The experimental results show that the runs that exploit the live explicit feedback exhibited a better performance in comparison to the baseline run that has been the best (among our runs) for the last two years. Additionally, all submitted runs have scored above the median provided by the track organizers in the batch evaluation.

1 INTRODUCTION

The rich and diverse stream of Twitter posts empowered users to utilize it as a main source of information on ongoing topics and events. However, due to the nature of the Twitter stream, manually tracking topics of interests (e.g., events) is a tough task for users, not to say impossible. This necessitates the need for a real-time summarization system that automatically tracks order of millions of topics of interest to users in parallel and produces a summary of on-topic tweets in real-time for each topic. Given the tweet stream, a real-time summarization (RTS) system aims to track predefined interest profiles over the online stream and pick relevant, novel, and fresh tweets out to suggest to the user. For instance, if a user is interested in following the updates on the “Opinions on Al Jazeera media network", the system should efficiently monitor the stream and capture the on-topic tweets including all aspects of the topic 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-2017 Real-time Summarization Track. Given a live stream of tweets and a set of interest profiles that represent users' topics of interest, the system simulates the scenario of push notification in which it pushes few tweets per day as notifications on the user's mobile phone for each topic. The track organizers hosted a broker through which participants can fetch the interest profiles and push the filtered tweet immediately when the 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). Most importantly, on the other side of the broker, active online users receive the push notifications on their mobile phones (for the topics of interest they subscribed to) and judge them tweets for relevance and redundancy[5]. The users' feedback is then sent back to participants' systems. This live assessment is a substantial learning resource for the system in which they can adjust the preferences on the users' interest for better summarization.

Our RTS system adopts a light-weight and conservative filtering strategy that listens to the English tweets over the 1% sample of the Twitter stream and processes the incoming tweets successively. It is 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 exploit the live feedback in two ways: (1) profile expansion and (2) dynamic thresholding. In the former, the system utilizes the positive feedback (truly relevant tweets) to update the profiles representation. In the latter, the system updates the relevance threshold periodically.

We organize the remainder of the paper as follows. We lay out the system approach in Section 2 and discuss the evaluation setting and our results in Section 3. Finally, we conclude and present possible future work in Section 4.

2 APPROACH

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. We extended our RTS system that we participate with in the real-time tweets summarization tracks in the past two years [6, 7]. In this section, we describe

TREC'17, November 2017, Gaithersburg, Maryland, USA
.

the system architecture for scenario A (i.e., Push Notifications scenario) in detail. We also focus on how the system utilizes the explicit feedback to improve performance.

2.1 Baseline System

Our system is conservative in a sense that it extensively filters out the noise and narrows down the set of candidate tweets for efficient scoring. Specifically, given a list of interest profiles (i.e., Topic in traditional ad-hoc), the system tracks these profiles over Twitter stream in a scalable manner and processes only the matching tweets (i.e., tweets that match at least one term of the profiles' titles) in a pipeline of multiple components: pre-qualification, preprocessing, indexing, relevance filtering, novelty filtering, and tweets nomination.

2.1.1 Pre-qualification. While the system monitors the stream using Twitter Streaming API[1], it filters out non-English and low-quality tweets. The criteria by which we determine the quality of a tweet is based on its length and number of hashtags and URLs it contains. Specifically, the system ignores any tweet that has less than five terms or more than one URL or more than three hashtags. Unlike previous years, the baseline system does not filter out retweets as the original (underlying) tweets might not be gathered among the 1% sample of Twitter stream.

2.1.2 Preprocessing. Once a tweet is qualified, the system preprocesses it in a series of steps that aim at cleaning its text before scoring it for relevance and novelty. These steps include expanding the tweet with hashtags' terms (i.e., after removing the '#' prefix), removing special characters (e.g., emoticon and symbolic characters), stopwords and URLs, and stemming.

2.1.3 Indexing. As we acquire term statistics in the components of the system, we initialized the system with an index of a 10-days stream of tweets prior to the beginning of the evaluation period. The system also incrementally indexes all incoming English tweets during the evaluation period.

2.1.4 Relevance Filtering. The system represents both interest profiles (represented by profiles' titles) and incoming tweets as vectors in the vector space model. To construct the vectors, the system computes an *idf*-based term weighting as follows:

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

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).

An incoming tweet is scored against a subset of profiles (the profiles that it matches) for relevance independently using the standard Cosine Similarity function. To compute the relevance scores in real-time efficiently, the system constructs an *in-memory* index of profile vectors to match an incoming tweet with interest profiles. The relevance of a tweet is determined using a relevance threshold τ_r. If the relevance score of a tweet is greater than the predefined threshold, the systems add the tweet to the potentially-relevant tweets for the corresponding profile.

[1]https://developer.twitter.com/en/docs/api-reference-index

2.1.5 Novelty Filtering. The system then measures the novelty of the potentially-relevant tweet by computing the overlap between it and already-pushed tweets using a modified version of Jaccard similarity:

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

Where Q and T are the profile and the tweet term sets, and $|Q|$ and $|T|$ are their lengths (in terms) respectively. To consider a tweet in the tweets nomination step, it must not exceed a predefined degree of overlap, i.e., a novelty threshold τ_n, with already-pushed tweets. This way the system does not overwhelm the user with redundant notifications.

2.1.6 Tweets Nomination. Thus far, the system identified the relevant and novel tweets. To satisfy the users' need while avoiding inundating them, the system has to take into account pushing a maximum of 10 tweets per day per profile. Moreover, the system has to consider the freshness of candidate tweets. Thus, the system should intelligently select the optimal candidate tweets to nominate to the user.

While following all interest profiles in parallel over tweets stream, the system maintains a list of *candidate tweets* for each of the interest profiles. The candidate tweets are the potentially relevant and novel tweets that the system identifies so far for a specific profile. For each profile, the RTS system periodically selects the next tweet to elect to user from the candidate list through a broker [4]. This selection component is triggered when the systems exceed a silence period δ or it has already found l candidate tweets for that profile.

To wisely select the best tweet to send to the user, the system *re-ranks* tweets in the candidate lists while considering relevance and freshness using equation 3. This re-scoring linearly penalizes the tweets based on their posting time, hence favoring fresh tweets. The top tweet is then pushed to the user.

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

$S_r(t)$ 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).

2.2 Exploiting Live Feedback

Unlike previous years, the system receives live explicit feedback for pushed tweets from online assessors, simulating the real settings of RTS system, through the broker. Based on the responsiveness assessors, the system might obtain multiple assessments or nothing for every pushed tweet. Although the explicit feedback is a valuable resource that could improve the system performance, it introduces many challenges. Among these challenges are the latency (when users are not responsive enough) and aggregating multiple judgments (as each tweet might receive multiple judgments). To mitigate the effect of these two challenges and for simplicity, the system only considers the first judgment for each tweet and ignore the remaining.

Our RTS system exploits the live explicit relevance feedback (ERF) that it fetches periodically from the broker, in two methods: (1) Profile Expansion, and (2) Dynamic Thresholding. In this section, we explain each of these methods in detail.

2.2.1 Profile Expansion. Topics discussed on Twitter develop over time which causes the so-called drift challenge. To cope with the Topic change on the live stream, the system periodically updates the profile representation periodically using a modified version of Rocchio's relevance feedback as shown in equation4. We only consider the truly relevant tweets as our experiments show that penalizing nonrelevant tweets, badly effects the RTS performance.

$$\vec{q}' = \vec{q} + \frac{\beta}{|F_r|} \sum_{d \in F_r} \vec{d} \tag{4}$$

Herein, F_r is the set of truly relevant documents (judged by the live assessors), $|F_r|$ is the number of truly relevant tweets, and β is a parameter used to control the influence of relevant tweets' terms on the new profile vector.

2.2.2 Dynamic Thresholding. The default thresholding technique in RTS system is to use a static global relevance threshold τ_r across all topics. Although the static threshold can be tuned over past tweet collections, it is still difficult, if not possible, to find a static threshold that maintains its optimality across time especially when topics evolve over time in the live stream. In other words, while considering the topic development, the RTS system is expected to elect a satisfying number of relevant and novel tweets to the user for a long time (spans over days to years). Moreover, considering the difference in difficulty level between interest profiles, where difficult profiles typically have less number of relevant tweets, makes the global threshold non-promising method.

To mitigate the effect of using a static global threshold on the RTS system performance, we utilize the live assessments to update the relevance threshold over time. The system uses an initial static global relevance threshold τ_r for all profiles and periodically updates per-profile p_i relevance threshold. To update the threshold, the system uses the number of pushed tweets that received positive feedback in the past period. If a profile p_i got no positive feedback in the last period, the system slightly decreases the relevance threshold τ_r by 0.05 with a lower bound of 0.5. Otherwise, the threshold is increased using the following equation:

$$\tau_{r_i}' = \tau_{r_i} + min(\frac{R_{p_i}}{100}, 0.15) \tag{5}$$

Where τ_{r_i} is the current threshold of profile p_i, τ_{r_i}' is the updated threshold of that profile and R_{p_i} is the number of truly relevant tweets for profile p_i within a time period t_T. The threshold upper bound is set to 0.8. Controlling the upper and lower bounds of the relevance threshold prevents extreme updates that might harm the system performance; using a very low threshold allows the system to overwhelm the user with many uninteresting updates, otherwise, the system would be strict and pushes no updates.

3 EXPERIMENTAL EVALUATION

Thus far we discussed the major components of the RTS system. We now discuss the evaluation settings, configurations, and performance (in terms of effectiveness) of our RTS system.

3.1 Evaluation Measures

The participating systems were evaluated using batch evaluation measures and live evaluation. For the former, there are three main batch evaluation measures: the expected gain (EG), the normalized cumulative gain (nCG), and the Gain minus Pain (GMP) measures. For the latter, there are two main measures: precision (P) and utility (U). Each of these evaluation measures is evaluated for each topic per day and then averaged over evaluation days. The final score of a run is the average of scores over all topics.

Similar to last year, the evaluation measures do not penalize systems for temporal latency. The latency was reported separately using mean (*MLT*) and median (*MedLT*) of the difference between the pushing time of each pushed tweet and the first tweet of the cluster which the pushed tweet belongs to (i.e., reference tweet).

3.1.1 Batch Evaluation: Following the traditional TREC evaluation, all participating systems' output are used to construct a pool of potentially-relevant tweets and to be labeled by in-house assessors. Systems are evaluated using this resultant qrels as follows:

- **Expected Gain (EG):**
$$\text{EG} = \frac{1}{N} \sum_{t \in P} G(t) \tag{6}$$
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 $\leq$ 10 tweets).
- **Normalized Cumulative Gain (nCG):**
$$\text{nCG} = \frac{1}{Z} \sum_{t \in P} G(t) \tag{7}$$
Z is the maximum possible gain for that topic in that specific day based on all judged pushed tweets.
- **Gain minus Pain (GMP):**
$$\text{GMP} = \alpha \sum_{t \in P} G(t) - (1 - \alpha)P \tag{8}$$
P is the number of non-relevant tweets pushed by the system and α is a parameter to balance between gain and pain.

For all above measures, $G(t)$ 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 RTS system in the push notifications mode is allowed to push a maximum of $n = 10$ tweets daily per profile during the evaluation period. All extra tweets will be discarded. All measures also penalize 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.

Each of the aforementioned measures has two variants there are two variants of treating the silent days. Silent days (in contrast to eventual days) are the days when there are no relevant tweets (for a specific profile). 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

Table 1: Official TREC 2017 results of QU runs for the push notifications scenario (batch evaluation). Best value per column is boldfaced.

Run	EG-p	EG-1	nCG-p	nCG-1	GMP.33	GMP.5	GMP.66	MLT	MedLT
QUBaseline	0.2422	0.2146	0.2260	0.1984	-0.2326	-0.1459	-0.0644	64812.8	**1**
QUExp	0.2356	**0.2185**	0.2159	0.1987	**-0.1498**	**-0.0909**	**-0.0354**	**63943.6**	**1**
QUExpDyn	**0.2547**	0.2068	**0.2475**	**0.1996**	-0.5182	-0.3457	-0.1833	77033.4	19
Median	0.2194	0.1951	0.2095	0.1826	-	-0.1707	-	-	-

Table 2: Official TREC 2017 results of QU runs for the push notifications scenario (live evaluation). Best value per column is boldfaced.

Run	# Rel.	# Redun.	# Nonrel.	# Unjudged	P_s	P_l	U_s	U_l	MLT	MedLT
QUBaseline	875	139	1298	111	0.3785	0.4386	-562	-284	**212.7**	1
QUExp	664	100	940	79	**0.3897**	**0.4484**	**-376**	**-176**	232.2	1
QUExpDyn	1399	274	2589	175	0.3282	0.3925	-1464	-916	310.2	1
Median	-	-	-	-	0.3403	0.4174	-805	-456	-	-

that have "p" as a suffix penalize systems by 0.1 multiplied by the number of pushed tweets by the system (fraction of the ten-tweet daily limit).

3.1.2 Online Evaluation: Using the explicit relevance feedback gathered from the the live assessors, systems are evaluated using two online precision measures:

- **Strict Precision** (P_s) = $|\frac{R}{R+RR+NR}|$
- **Strict Utility** (U_s) = $|R - RR + NR|$

Where R, RR, and NR are the set of relevant, redundant and nonrelevant tweets, respectively. There are two variants of these measures: *strict* measures, that have "s" suffix, penalize systems on redundant tweets and *lenient* measures, that have "l" suffix, are more flexible in which they do not evaluate on novelty.

- **Lenient Precision** (P_l) = $|\frac{R+RR}{R+RR+NR}|$
- **Lenient Utility** (U_l) = $|(R + RR) - NR|$

3.2 Official Runs

In this section, we discuss our submitted runs for both scenarios in detail including the configuration and results. We tuned all system parameters over two test collections of microblog tracks: TREC-2015 Real-time Filtering track [3] and TREC-2016 Real-time Tweet Summarization track [3].

We submitted three automatic runs that we describe below:

- **QUBaseline** is the baseline run that activates the core system components without exploiting the live relevance feedback.
- **QUExp** has a similar configuration to QUBaseline run, except that it uses utilizes the live relevance feedback to *hourly* update the textual representation of profiles. It expands the profile representation by the positive relevance feedback after weighting them by a factor $\beta = 0.2$ to avoid the topic drift.
- **QUExpDyn** in addition to performing *hourly* expansion, this run adopts a simple adaptive thresholding in which it dynamically updates the relevance threshold per profile.

In all runs, we set both relevance τ_r and novelty τ_n thresholds to 0.6 as it was the best value in the experiments that we conducted.

Table 1 shows the performance of our official runs in comparison to median scores provided by the track organizers for EG, nCG, and GMP.5 measures. Note that both the mean latency (MLT) and median latency (MedLT) are reported in seconds. Surprisingly, *QUExpDyn* and *QUExp* runs outperform the baseline *QUBaseline*. This never happens in our participation in the last two years. *QUExpDyn* is the best run in EG_p and *nCG* measures and *QUExp* performs well in EG_1 all variants of *GMP* measure compared to other runs. These results show how exploiting explicit live feedback improves the system effectiveness in contrast to our attempts when using pseudo-relevant feedback[6, 7]. Additionally, when comparing to *Median* scores, all QU runs outperform them in all measures, except for *QUExpDyn* in *GMP*.5 measure.

In table 2 we present the system results using the online evaluation. It can be clearly seen that *QUExp* run outperform all other runs in all measures. Additionally, we notice that QUBaseline and *QUExp* runs outperform the median in all measure, however, *QUExpDyn* fails to beat the *Median* in the online evaluation.

4 CONCLUSION AND FUTURE WORK

We presents our RTS system participated in the real-time summarization track in TREC-2017. The baseline system filters tweets in a simple pipeline that includes multiple stages: pre-qualification, preprocessing, relevance filtering, novelty filtering, and tweets nomination. The system also exploits live explicit feedback to update profiles in two ways: (1) profile expansion and (2) per-profile dynamic thresholding. Generally, the results show that, runs that use the explicit relevance feedback outperform the baseline in batch evaluation. Surprisingly, the results show that the run that performs dynamic thresholding is the best in batch evaluation, however, it poorly perform in online evaluation (scoring below the median score). We plan to further perform failure analysis on all components of the system to better understand the results. Additionally, we are planning to experiment with different relevance models (e.g., the so-called Relevance Model [1]) when utilizing the live explicit feedback.

5 ACKNOWLEDGMENTS

This work was made possible by 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.

REFERENCES

[1] Victor Lavrenko and W Bruce Croft. 2001. Relevance based language models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 120–127.

[2] Jimmy Lin. 2017. TREC 2017 Track Guidelines. http://trecrts.github.io/TREC2017-RTS-guidelines.html. (2017).

[3] Jimmy Lin, Miles Efron, Yulu Wang, Garrick Sherman, and Ellen Voorhees. 2015. Overview of the TREC-2015 Microblog Track. In *Proceedings of the 24th Text REtrieval Conference (TREC '15)*.

[4] Jimmy Lin, Adam Roegiest, Luchen Tan, Richard McCreadie, Ellen Voorhees, and Fernando Diaz. 2016. Overview of the TREC-2016 Real-Time Summarization Track. In *Proceedings of the 25th Text REtrieval Conference (TREC '16)*.

[5] Adam Roegiest, Luchen Tan, and Jimmy Lin. 2017. Online In-Situ Interleaved Evaluation of Real-Time Push Notification Systems. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 415–424.

[6] Reem Suwaileh, Maram Hasanain, and Tamer Elsayed. 2016. Light-weight, Conservative, yet Effective: Scalable Real-time Tweet Summarization.. In *Proceedings of the 25th Text REtrieval Conference (TREC '16)*.

[7] Reem Suwaileh, Maram Hasanain, Marwan Torki, and Tamer Elsayed. 2015. QU at TREC-2015: Building Real-Time Systems for Tweet Filtering and Question Answering. In *Proceedings of the 24th Text REtrieval Conference (TREC '15)*.