PolyU at TREC 2016 Real-Time Summarization

Haihui Tan **Dajun Luo** **Wenjie Li**

tanhaihui92@gmail.com
luodajun100@gmail.com
cswjli@comp.polyu.edu.hk

Abstract. This paper presents the participation of The Hong Kong Polytechnic University (PolyU) to the TREC 2016 Real-Time Summarization track. The two tasks related to Scenario A and Scenario B both focuses on information real-time processing. During the evaluation period, the system monitors the Twitter sample stream with respect to a number of "interest profiles". We submitted three runs for both scenarios. We describe the system overview and the implementation details in this paper.

1. INTRODUCTION

The TREC Real-Time Summarization (RTS) track explores techniques for constructing real-time update summaries from social media streams in response to users' information needs. The TREC 2016 Real-Time Summarization evaluation took place from August 2, 2016 00:00:00 UTC to August 11, 2016 23:59:59 UTC. During the evaluation period, participating systems monitor the Twitter sample stream with respect to a number of "interest profiles" that represent users' information needs. The Twitter streaming API offers an approximately 1% sample of all tweets (sometimes called the "spritzer"). For this year, 203 interest profiles (queries) are given.

In Scenario A, the content that is identified as relevant by a system based on the user's interest profile will be pushed in real-time. At a high level, push notifications should be relevant (i.e., on topic), timely (i.e., to provide updates as soon after the actual event occurrence as possible), and novel (i.e., users should not be pushed multiple notifications that are about the same thing).

In Scenario B, a system can identify a batch of up to 100 ranked tweets per day per interest profile. At a high level, these results should be relevant and novel. Timeliness is not important as long as the tweets were all posted on the previous day.

Comparing these two scenarios, Scenario A is a "really" real-time task while Scenario B is not, which means we may utilize some more sophisticated real-time techniques and pay more attention on the efficiency problem during implementation.

To fulfil the requirements mentioned above, we develop a system, which consists of pre-processing, relevance measurement, redundancy detection and push strategy. We also take efficiency, robustness and reliability into consideration.

2. SYSTEM OVERVIEW

The whole system is based on the "bag-of-words" model. As shown in Figure 1, the system consists of two parts: Offline Part and Online Part.

Offline Part: 1. In order to find the synonyms and related keywords to better measure the relevance, we expand the original query (interest profiles) utilizing the Bing News Search API and Reuters Corpus. 2. In order to measure the relevance of each coming tweet w.r.t the query, we train a Relevance Measurement Model based on the labelled data of Microblog Track from past years, 3. Utilizing the clustered data from past years, we train a clustering model to do redundancy detection and discard the redundant ones.

Online Part: 1. Use official API to listen the Tweet Stream. 2. Pre-process a vast amount of tweets to filter out most of the "trash" or irrelevant tweets and transform each tweet into a standard and clean format. 3. Extract the feature vector from each tweet based on tweet text, URL external text and metadata. 4. Estimate Relevance score. 5. Check the redundancy. 6. Do a Scenario A Push and Scenario B Ranking using certain rules (such as daily quota, etc.).

Since the core tasks in Scenario A and Scenario B are the same, i.e., to estimate the tweet relevance to interest profiles, the workflows for these two scenarios are merged into one single system, except the last module (Push for Scenario A and Rank for Scenario B).

The difference among PolyURunA1, PolyURunA2, PolyURunA3 (PolyURunB1, PolyURunB2, PolyURunB3) is in the Relevance Measurement Model. There's a slightly difference in Relevance Measurement strategy among 3 runs. We will describe in more detail in the next section.

3. COMPONENTS

Offline Part:

3.1 Query Expansion

In order to find the synonyms and related keywords to measure the relevance better, inspired by the approaches proposed in [3][4], we do the query expansion. For every topic, we feed the topic ***title*** into Bing News Search API and get top 50 search results' snippets. Then we calculate the TF-IDF score based on Reuters Corpus for every term in all snippets, and select top-20-score terms as "Expansion Terms". We select top 10 terms from ***narrative*** and ***description*** the same way as above, as "Narr-Desc Terms". All terms except stopwords in ***title*** as "Title Terms".

3.2 Relevance Model Training

For 3 different runs, we implement 3 different Relevance Measurement strategies. And we use the same relevance measurement on both scenarios.

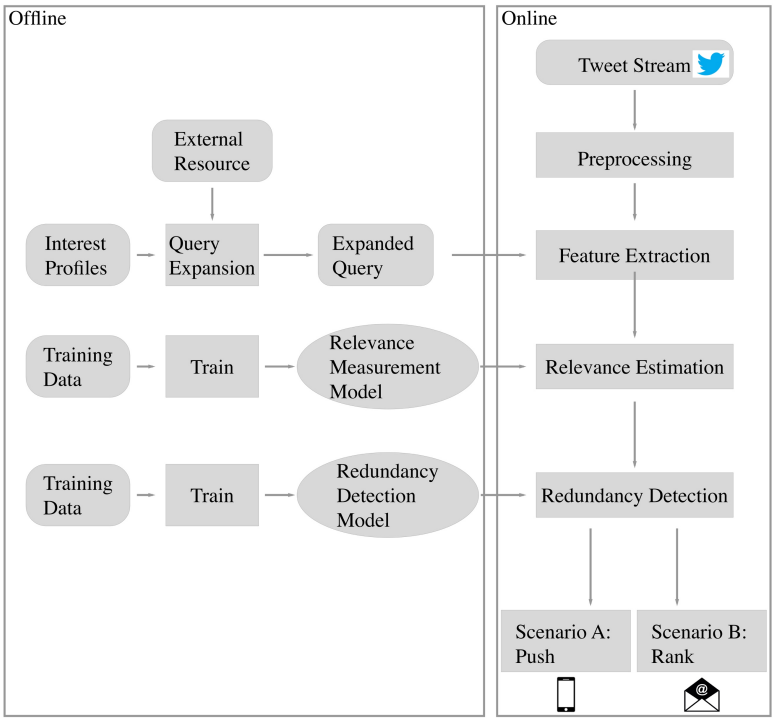

Figure 1: An Overview of the system

Run 1: 12-features model (add word2vec-info feature)

Almost the same as Run 2, except that we add an additional feature: word2vec-info. To briefly explain the intuitive idea behind this feature, let's look at an example. For topic MB236, "California drought agricultural effects", after query expansion, we can get these terms as expanded query: "water" "groundwater" "california" "drought" "impact" and etc.

Now, let's imagine there are two different tweets comming:

tweet A: "water drought groundwater"

tweet B: "water California impact"

Both can be hit 3 times by terms in query. However tweet B is more relevant, apparently, because tweet B gives more information. In other words, "water" "California" "impact" these three words are semantically "distant" from each other so they bring more relevant information. On the other hand, "water" "groundwater" "drought" are semantically close to each other (actually all about water), although they hit 3 times, they bring less information.

Under this intuition, we utilize the word2vec model [5] to calculate the similarity between words, and then we can calculate how much relevant information a tweet brings to us. We give the equation:

$$info = \sum_{i=1}^{N} (1 - similarity(t_i, t_{close}))$$

Here, t_i denotes the term which hits the text, N denotes the number of terms which hit the text, t_{close} denotes the most similar term to t_i which hits the text. The range of $similarity(t_a, t_b)$ function is (0, 1] where 1 means t_a and t_b are the same word. The word2vec model is trained on GoogleNews Corpus.

Run 2: 11-features model (without word2vec-info feature)

In a word, the core task is to determine whether a tweet is relevant to an interest profile (profile) or not. Intuitively, we consider the relevance estimation as a classification problem in which we should classify each coming tweet into three classes: 2 (highly-relevant), 1 (relevant) or 0 (not-relevant).

Inspired by [2][4][2], we utilize a 11-features vector space to represent every tweet to a corresponding interest profile (query), and we list all of them as following:

count_ti: the number of times "Title Terms" appear in tweet text

count_te: the number of times "Title Terms" appear in external URL text

count_di: the number of times "Narr-Desc Terms" appear in tweet text

count_de: the number of times "Narr-Desc Terms" appear in external URL text

count_ei: the number of times "Expansion Terms" appear in tweet text

count_ee: the number of times "Expansion Terms" appear in external URL text

is_link: a tweet consists of URL for 1; otherwise for 0

log_followers_count: the log of the number of followers of the author who publishes the tweet.

log_statuses_count: the log of the number of tweets the author publishes.

world_count: the number of words in this tweet.

hashtag_count: the number of hashtag in this tweet

We train a SVM model based on labelled Microblog Track past data. For each coming tweet, we predict it into 2 (highly-relevant), 1 (relevant) or 0 (not-relevant) with a probability based on well-trained SVM classifier. Since the metrics this year lays emphasis on precision rather than recall, so only if a tweet is predicted into highly-relevant class, then we let the corresponding probability value be its Relevant Score and do the next processing; otherwise we discard this tweet.

Run 3: Naive Strategy

This Run implements a naïve but efficient strategy to estimate the relevance between an interest profile and a tweet. Intuitively, more "profile terms" appears in a tweet more relevant they are. We regard this naïve strategy as a reference compared to other two runs with more complicated relevance-estimation strategy. So, we just add up 6 kinds of number of appearance as following:

ti: the number of times "Title Terms" appear in tweet text

te: the number of times "Title Terms" appear in external URL text

di: the number of times "Narr-Desc Terms" appear in tweet text

de: the number of times "Narr-Desc Terms" appear in external URL text

ei: the number of times "Expansion Terms" appear in tweet text

ee: the number of times "Expansion Terms" appear in external URL text

$$Relevance\ Score = ti + te + di + de + ei + ee$$

Online Part:

3.3 Pre-processing

(a) Language detection: Based on RTS official guideline, Non-English tweets should be judged as not relevant. To filter out non-English tweets, we read the "*lang*" field of Tweet Stream API attribute and retain only the "en" ones. If length (ASCII characters)/length (the whole text) < 0.8, we discard this tweet, because that means this tweet consists of too many non-ASCII characters. Then, we use langdetect [1] package to detect the language and discard non-English tweets. The main reason why we use these three-level detector to detect non-English tweets is that efficiency is crucial in this real-time track. We will discuss the efficiency issue in the Section 4.

(b) Trash discard: Learned from [2], we decide to discard "trash tweet" before further processing. If a tweet meets one or more these conditions below, we regard it as "Trash" tweet and filter out it (bracketed texts are intuitive reasons):

1. the length of text is less than 20; (too few words to provide enough information)

2. the number of hashtag is more than 5; (typical trash tweet)

3. All characters are capital. (typical trash tweet)

(c) Keyword filtering: In consideration of efficiency, we implement a keyword filtering to filter out the vast majority of apparently irrelevant tweets. We select keywords from every topic's *title* based on the IDF score on Reuters Corpus and check them manually. If a coming tweet does not match any one of these keywords, we discard it.

(d) Crawl URL: crawl external URL webpage text to get more text about this tweet for further process.

(e) Clean original tweet text: remove hashtag, remove RT@, remove all URLs.

3.4 Redundancy Detection

For each tweet considered as relevant to an interested topic, an efficient and naive similarity measurement was adopted to detect redundancy.

We observe that most of redundant tweets are not rephrased but simply copies of the original tweet. Hence our method assumes that the similarity between two tweets is merely determined by occurrences of their common vocabulary.

The similarity for redundancy detection is defined as the union score divided by the intersection score of two tweets.

$$\text{Similarity}(t_1, t_2) = \text{union\_score}(t\_1, t\_2) \,/\, \text{intersection\_score}(t\_1, t\_2)$$

The union score of two tweets is defined as the sum of word length of the union of two tweets. And the intersection score is defined as the sum of word length of the common word of the two tweets.

For example, tweet A is ***"I have a cat"***, tweet B is ***"My cat has claws I do not have"***. The union of these tweets is ***"I have a cat my cat has claws I do not have"***. "I" length is 1, "have" length is 4, and etc.. So the union score is 1+4+1+3+2+3+3+5+1+2+3+4 = 32. While the intersection is ***"I have cat cat I have"***. Hence the intersection score is 1+4+3+3+1+4 = 16. By our definition, the similarity of sample tweets is 16 / 32 = 0.5.

Using the training data from previous years, we find an optimal threshold of 0.6 for redundancy detection. For each topic we maintain a list of pushed tweets. Before pushing a new tweet, it's similarity with every tweet which is pushed previously in the list corresponding to the topic is computed to determine redundancy. Then discard the redundant ones.

3.5 Push Strategy and Ranking Strategy

As for Scenario A, it seems like a variant of The Secretary Problem. However, a crucial difference is that The Secretary Problem aims to get more sum of gain or sum of score, regardless of average gain or "precision". In Scenario A, according to metrics of guideline, recall is much less important than precision. So, a "cautious" push strategy is reasonable. We set thresholds based our observation on the Tweet Stream for days before evaluation period. As for Run1, we only push the tweets which were classified as "highly-relevant" with the probability of 0.7 or higher. For Run2, we only push the tweets which were classified as "highly-relevant" with the probability of 0.6 or higher. For Run3, we only push the tweets with the relevance score of 7 or higher.

As for Scenario B, we use the same relevance measurement with Scenario A. The only difference is that we collect and store all the tweets relevant to each topic during one day.

Then select top ten tweets to push, ordered by the probability to be classified as "highly-relevant".

4. ABOUT EFFICIENCY

The efficiency of the whole system is significant in this year's Track, since the listening and pushing part are both truly real-time. The Twitter streaming API offers an approximately 1% sample of all tweets. According to docs [6], Twitter streaming volume is not constant. Throughout the course of a 24 hour period, there is a natural ebb and flow to the number of Tweets delivered per second. According to our test, the coming data rate of Twitter Public Stream is about 50 tweets/s ~ 80 tweets/s. So that is to say, if the rate of a system's pipeline to process all coming tweets is less than this rate, there might be some problem.

Our approach to tackle this problem is to set 5 levels of filters as mentioned above. The idea is that put the compute-fast filtering module front in pipeline, so only a few of tweets need to pass compute-intensive or time-consuming module, such as external URL webpage crawling or Relevance Estimation.

5. RESULTS

Team	Run ID	P (strict)	P (lenient)
COMP2016	run1-11	0.5143	0.5238
COMP2016	run2-12	0.5465	0.5581
COMP2016	run3-13	**0.5710**	**0.5828**

Table 1: Performance of submitted runs for Scenario A (evaluated by the mobile assessors)

Team	Run ID	EG-1	nCG-1	GMP(.33)
COMP2016	run1-11	0.2565	0.2515	-0.0804
COMP2016	run2-12	0.2559	0.2483	**-0.0585**
COMP2016	run3-13	**0.2698**	**0.2909**	-0.3262

Table 2: Performance of submitted runs for Scenario A (evaluated by NIST assessors)

Team	Run ID	nDCG-1	nDCG-0
COMP2016	PolyURunB1	0.2536	0.0215
COMP2016	PolyURunB2	0.2523	0.0184
COMP2016	PolyURunB3	**0.2898**	**0.0684**

Table 3: Performance of submitted runs for Scenario B (evaluated by NIST assessors)

For Scenario A, Table 1 and Table 2 report the performance of our three runs. As we can see, Run3 outperforms both other runs, indicating that a naïve strategy is very useful in such kind of task. Note that Run3 gets a worst GMP (.33) in Table 2. It means although the naïve strategy gains much, it suffers from pushing too many non-relevant tweets. Run2 performances better under GMP (.33) metric.

For Scenario B, Table 3 reports the performance of our three runs. Apparently, Run3 significantly outperforms other two runs. That means a naïve strategy is effective for this scenario.

6. ACKNOWLEDGMENTS

The work described in this paper was supported by Research Grants Council of Hong Kong (PolyU 152094/14E) and National Natural Science Foundation of China (61272291).

7. REFERENCES

[1] Shuyo Nakatani. 2010. Language detection library (slides). URL: http://www.slideshare.net/shuyo/language-detection-library-for-java.

[2] Luchen Tan, Adam Roegiest and Charles L.A. Clarke. University of Waterloo at TREC 2015 Microblog Track. In *The Twenty-Fourth Text REtrieval Conference (TREC 2015) Proceedings*. NIST, 2015.

[3] Feifan Fan, Yue Fei, Chao Lv, Lili Yao, Jianwu Yang and Dongyan Zhao. PKUICST at TREC 2015 Microblog Track: Query-biased Adaptive Filtering in Real-time Microblog Stream. In *The Twenty-Fourth Text REtrieval Conference (TREC 2015) Proceedings*. NIST, 2015.

[4] Tianyi Luo, Dehong Gao and Wenjie Li. POLYUCOMP at TREC 2014 Microblog Track.

[5] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013. arXiv:1301.3781v3 [cs.CL] 7 Sep 2013.

[6] Twitter Documentation of Streaming APIs. URL: https://dev.twitter.com/streaming/overview/processing