ICTNET at TREC 2017 Real-Time Summarization Track

Xiao Wang[123], PengCheng Fan[123], Liang Cheng[123], Guoliang Xing[12] ,Zhihua Yu[12], Xueqi Cheng[4]
1)Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190
2)Key Laboratory of Web Data Science and Technology,CAS
3) University of Chinese Academy of Sciences, Beijing, 100190
4) Institute of Network Technology，ICT（YANTAI），CAS
{wangxiao,fanpengcheng,chengliang}@software.ict.ac.cn; {xingguopliang,yzh,cxq}@ict.ac.cn

1. INTRADUCTION

Nowadays Twitter represents a successful social application and own billions of users, and there is a growing trend for recommending high quality message to users due to the business need. The Real-Time Summarization (RTS) track explores techniques for constructing real-time update summaries from social media streams in response to users' information needs.

Our task then can be reduced to a recommendation system, actually it is a recommendation system based on data stream which has continuous and enormous message data of various types. As indicated by the name of the track, the main goal of this track is to solve the problem of making the recommendation real-time, relative and novel, which is exactly the demands of scenario A. For scenario B, posting the mail digest is like a compromise to scenario A, in this scenario, we only need to post a batch of tweets to users through email at the end of the day, which means there is no real-time limitation of scenario A, and this problem then can be reduced to traditional ad-hoc search problem. In this paper we mainly focus on scenario A and then conduct the solution of scenario B based on scenario A.

Substantially, Scenario A can be interpreted as following problem: given user profiles (also known as topics) which is the abstraction of a certain crowd of people’s interests, by monitoring the twitter steam data, we need to make real-time (as soon as the tweet is posted), relative and novel tweet recommendation to the corresponding profile once detected. The problem mainly contains the following aspects: text processing (i.e. natural language processing), vectorization (feature selecting and extraction, vector similarity) and data storage (database management), the key part is the vectorization and similarity calculation. We need to combine these parts together to build an effective system.

Our approach mainly includes two different models, the word2vec model and TF-IDF model. In word2vec model, we simply train a word2vec language model based on wiki corpus, and then appliy the model to tweet and profile to gain vector, based on the similarity between tweet vector and profile vector we adopt corresponding pushing strategy. The TF-IDF is different in vectorization, it cached tweets a few days ahead, and apply TF-IDF model on the cached tweet corpus as well as profiles to gain the tweet vector and profile vector, details will be explored in the rest of the paper. According to the results of evaluation, our TF-IDF model achieved better performance than the naïve word2vec model, the best run was around 15% above the medium score of all automatic runs this year in Scenario A.

2. SYSTEM DESIGN

This part elaborates the specific system design and details. As mention before, our system mainly focuses on scenario A (pushing notification) part, thus the following part is actually organized by the skeleton of scenario A. Scenario B is based on the design of scenario A, it will also be fully presented.

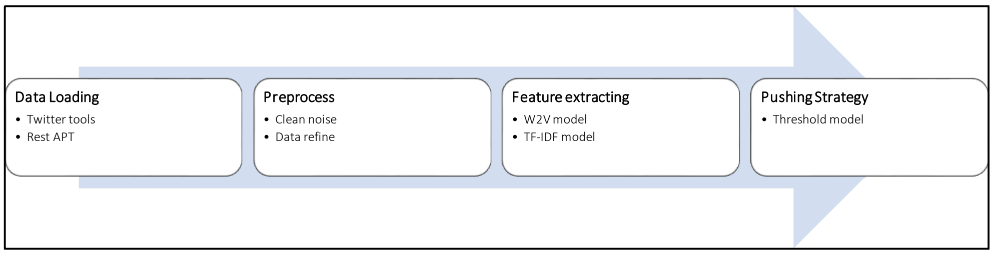

Figure 2-1

Our system is a linear system, as showed in Figure 2-1, it is constructed of data loading module, preprocess module, feature extraction module and pushing strategy module in sequential order. The logic is clear to follow, first we load tweet data and profile data, then implement the preprocessing of the tweet text and profile text, next we present the tweet text and profile text in vector, calculate the similarity between them, and pushing the tweet according to the pushing strategy. The details of each module will described below.

2.1. Data loading

In pushing notification part (Scenario A), since we need to push tweets to users in real-time, we cannot afford too much time in the whole process. Therefore, the system must be able to achieve the tweet pushing fast enough. Intuitively, we load the tweet one by one. We first sample the twitter stream(1%) by twitter tools[1] and cache the sample data in database. Here we cache the data because of the need of constructing the TF-IDF model, besides, the cached data is necessary for email digest (Scenario B).

Once the tweet data is cached, we load the data from database and move into next system module, here we need to pay attention to control the speed of loading data, ensuring that the loading speed is slower then sampling speed otherwise we would encounter database error.

For user profiles, we can obtain the profiles through REST API officially provided by NIST. These profiles are JSON format file, they can be loaded directly.

2.2. Preprocess

For twitter data, the preprocess module is necessary for handling the text data since normally the original data always includes many noises, we take different strategies to clean the noises and keep the main information. Our principle is keep the text information to maximum because we believe the most valuable information is always presented by the text.

2.3.1. Non-ASCII Filtering: Some tweets has a lot of emoticons and signals in text content. To keep the system more effective, we regard them as noises and filter them out.

2.3.2. Non-English Filtering: We just keep the English tweets in our system as required by the track, so we filter all non-English tweets from the raw data.

2.3.3. Too-Short Filtering: We believe that those too short tweets are too hard to get useful information. If we kept them, the vectorization result would not be good. Thus we filter these too short tweets. Basically, if a tweet is less than three words, it will be regarded as noises and will be cleaned.

2.3.4. URL Filtering: If we want to calculate the similarity between a tweet and profile, we should pay more attention on text content. We filter the URLs in tweet text content by adding a URL filter.

2.3.5. Hashtags-Retweet Filtering: We remove all hashtags and retweet information in the raw tweet data. Keeping hashtags will result in bad performance in our system so we remove them, specifically, for retweet we only retain the original tweet text of all retweets.

For user profiles, the JSON file already has decent format so we just skip the preprocess procedure.

[1] https://github.com/lintool/twitter-tools

However, we still need to do some expansion for these profiles, and it turns out that the "title" expansion has the best performance in our system, this will be discussed our feature extraction module next.

2.3. Feature extraction

This module is the key module of our system since it defines the way of text vectorization, which is crucial for similarity calculation and message pushing. In our system, the vectorization for tweet and profile is the same, we mainly adopt two models, the word2vec model and TF-IDF model.

2.3.1. Text refining

First we have to refine our preprocessed tweets and profiles. We mainly have three operations. 1) Stopwords filtering. We keep a stopword table in our system and then filter all the stopwords in the text (tweets and profiles). 2) Lemmatization. This is the common procedure in natural language processing, the purpose is to get the original form of the word, by means of this we can reduce the unnecessary dimension in vectorization. 3) Stemming. We only keep the necessary meaning part for words, this is the old trick like lemmatization which will help enhance the performance in vectorization. Then the refined text will be updated from the database for the next procedure of vectorization by the model.

2.3.2. Modeling

As mentioned above, we mainly take two different models to build our system, and they are implemented in our three submitted runs, we will compare and analyze the performance between them.

word2vec model: This model is naïve word2vec[2] model. We use Wiki corpus we crawled to train the word2vec model, and then obtain the word vector of text by the trained model. For tweet, we simply take the average after sum the word vector to get tweet vector. For profiles, we only take "title" part of a profile to get the corresponding vector like tweet. In this model, the main work load is the training process of the model as well as the corpus crawling. Our first run (ICTNET-Run1) adopted this model.

TF-IDF model: Strictly, TF-IDF is not a model, actually it is only a technique in VSM (vector space model), here we use TF-IDF is for the convenience to elaborate. In this model, first we used the cached tweet data and profiles from database as corpus to train the TF-IDF model, here we use third-part library to train it.[3] Particularly, for profile, we only use the "title" part, but we also need to do some expansion. Here we mainly take two ways to expansion. One way is to use the "description" part of the original profile to expand it because the "description" part is the detail information of "title" part which may complement the information for the single "title" part. The other way is to use google search API the expand the "title". We observe that some "title" is abbreviation of some entry like people's name or some specific place, and we cannot have more information in even "description" part of the profile, thus we decide to use the google search API to gain the most relative text information to expand the "title" part. In our run, we take the first 3 result by using the "title" as query.

Next, after we trained the TF-IDF model, we can get the tweet vector and profile vector of the same dimension, simply we take the average of the sum of the words in text like before. The rest part is the similarity calculation between tweet vector and profile vector. Our second and third run (ICTNET-Run2/ICTNET-Run3) used this model, with different parameters.

2.3.3. Similarity

We calculate the similarity to decide whether they are relative, the higher similarity, the higher relativeness, this is the same between tweets. We use the standard cosine similarity to calculate the similarity score.

[2] https://github.com/RaRe-Technologies/gensim
[3] https://github.com/scikit-learn/scikit-learn

$$similarity = \cos(\theta) = \frac{A \cdot B}{|A| \cdot |B|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Formula 2-1

Where A_i and B_i are the components of A and B respectively.

2.4. Pushing strategy

For scenario A, our pushing strategy takes the philosophy of simple, basically it is a threshold model. First we set a threshold of similarity score to judge whether a tweet is relative to a user profile, then after we calculated the similarity score, if the similarity score has exceeded the threshold, then we come into the novelty check. For novelty, we set another threshold, since we can push at most 10 tweets to one profile per day, so for each profile, we maintain a pushing queue. Once the tweet is qualified for the relativeness, we calculate the similarity between this tweet and each tweet in the pushing queue, if all the similarity score is under the novelty threshold (the higher similarity between tweets, the less novel of the new tweet), we push the tweet to the corresponding profile through the REST API.

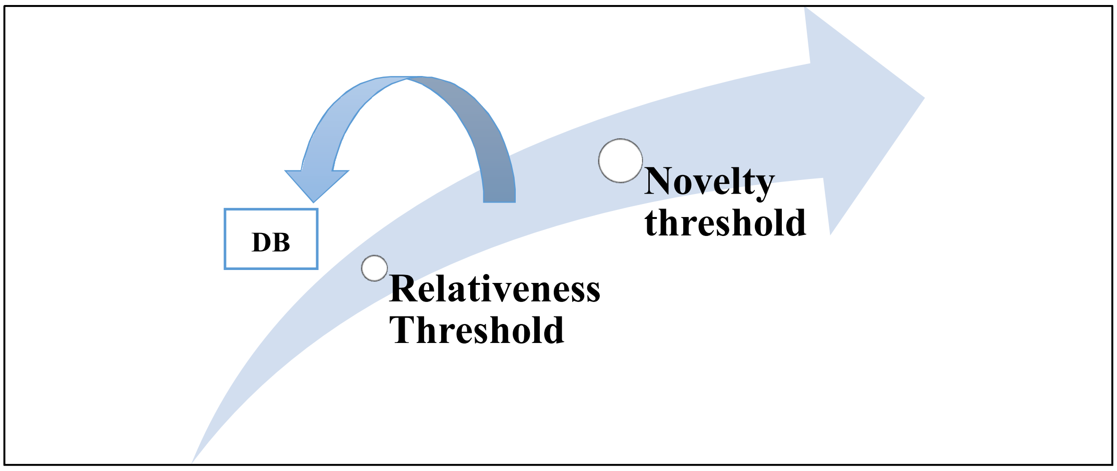

Figure 2-2

For scenario B, one thing to notice is that the email digest does not require novelty, as shown in Figure 2-2, our policy is to store all the similar tweet that pass the relativeness threshold in scenario A into database, at the end of the day, we sort all these tweets by the similarity score and choose the first 100 tweets as the email digest.

Here we list our parameter table.

	Relativeness threshold	Novelty threshold
ICTNET-Run1	0.80	0.60
ICTNET-Run2	0.53	0.80
ICTNET-Run3	0.65	0.95

Table 2-1

3. EVALUATION RESULTS

The evaluation period for TREC 2017 RTS is finally from July 25 00:00:00 UTC to August 3 23:59:59 UTC. In this period, all participated systems will perform their runs and submit the results after evaluation include ours.

For scenario A, the evaluation metrics include Expected Gain (EG) and Normalized Cumulative Gain (nCG), according to the final official email feedback, we list our overall performance in Table 3-1 below.

	EG-p	EG-1	nCG-p	nCG-1
Medium score	0.2194	0.1951	0.2095	0.1826
ICTNET-Run1	0.1959	0.0667	0.1751	0.0458
ICTNET-Run2	**0.2444**	0.1976	**0.2455**	**0.1987**
ICTNET-Run3	0.2338	**0.2005**	0.2227	0.1893

Table 3-1

As we can see from Table 3-1, the medium score of all participated runs is in the first row, overall our best run (ICTNET-Run2) outperform the medium run at each metrics. Especially in EG-p and nCG-p, our second run have nearly 15% and 20% enhancement separately.

For scenario B, basic metric this year is Normalized Discounted Cumulative Gain (nCG), the results of our runs are in Table 3-2.

	nDCG@10-p	nDCG@10-1
Medium score	0.2194	0.1865
ICTNET-Run1	0.1208	0.1143
ICTNET-Run2	0.2047	0.1381
ICTNET-Run3	0.2185	0.1527

Table 3-2

From Table 3-2 we can see that our system has ordinary performance in scenario B, the best run (ICTNET-Run3) is only close to the medium score in nDCG@10-p. The reason is not hard to find because our system is mainly focused on scenario A, for scenario B, we just sort the similar tweet passed the relative threshold. Substantially, scenario A only consider the current similarity between tweet and profile due to the requirement of real-time, however in scenario B we have to consider the overall similarity between tweet and profile in a whole day, this is different with real-time scenario, so the simply sorting would not have good performance.

According to the two tables, we can tell that our TF-IDF model (used in ICTNET-Run2 and ICTNET-Run3) outperforms our word2vec model, we have concluded the reason mainly come from the fact that the word2vec model has not utilized the information of twitter data and profiles, only the Wiki corpus is not enough to present the real text vector.

We also list our “strict” and “lenient” precision of Scenario A below and compare them with TREC 2016 top 10 runs, our best run (ICTNET-Run2) has decent performance according to the rank.(last row)

Rank	Team Best Run	Precision(strict)	Precision(lenient)
1	CLIP-A-1-08	0.5028	0.5083
2	UmdHcilBaseline-49	0.4762	0.4762
3	PRNATaskA3-36	0.3883	0.4369
4	iritRunBiAm-21	0.3792	0.3906

5	run2-32 (PKUICST)	0.3769	0.4015
6	QUExpP-38	0.3689	0.3770
7	WaterlooBaseline-50	0.3318	0.3587
8	MyBaseline-24(ISIKol)	0.3189	0.3501
9	WaterlooBaseline-51	0.3180	0.3355
10	nudt sna-29	0.3112	0.3515
*	***ICTNET(2017)***	***0.3403***	***0.4174***

Table 3-3

4. CONCLUSIONS& Acknowledgements

In this paper, we have discussed system design of our RTS track in TREC 2017 from data loading to pushing strategy, the main work of our system is to use the pre-cached Twitter data along with the Google query expansion to train the TF-IDF model, our pushing strategy takes use of threshold model out of simplicity. According to the evaluation results, our system has a decent performance in Scenario A pushing notification whereas trivial in Scenario B of email digest. For future work, we will try to explore the enhancement of model accuracy as well as pushing strategy, the dynamic threshold model is worthy exploring, beside, the profile expansion is also a potential part to enhance.

We would like to thank all organizers and assessors of TREC and NIST. This work is sponsored by the National Basic Research Program of China (the 973 program) under grant numbers 2014CB340401 and 2014CB340406; Taishan Scholars Program of Shandong Province，China（No.ts201511082）

5. REFERENCES

[1]. *J. Lin. TREC 2017 track guidelines. http://trecrts.github.io/TREC2017-RTS-guidelines.html.*

[2]. *Lin, J.; Efron, M.; Wang, Y.; and Sherman, G. 2015. Overview of the trec-2014 microblog track. Technical report, DTIC Document.*

[3]. *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.*

[4]. *Luchen Tan, Adam Roegiest, and Charles LA Clarke.2015. University of waterloo at TREC 2015 microblog track. In Proceedings of the Twenty-Fourth Text REtrieval Conference*

[5]. *Kathy Lee, Ashequl Qadir, Vivek Datla, Sadid A. Hasan, Joey Liu, Aaditya Prakashy, Oladimeji Farri. Assorted Textual Features and Dynamic Push Strategies for Real-time Tweet Notification. In Proceedings of the 25th Text REtrieval Conference, TREC '16, 2016.*

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