University of Pittsburgh at TREC 2014 Microblog Track

Zheng Gao
School of Information Science
University of Pittsburgh
zhg15@pitt.edu

Rui Bi
School of Information Science
University of Pittsburgh
rub14@pitt.edu

Abstract:

An ad hoc retrieval task aims at return the most relevant documents based on a particular query. And high precision and recall always depends on clear query and elaborative documents. If the query is ambiguous while document is short and general, common retrieval models may not work well on the feedback. In this way, more expansive information will be needed to add in order to implement original queries and documents. That is the main purpose of microblog track of 2014 TREC Conference.

The paper describes the first participation of University of Pittsburgh in 2014 TREC microblog track. The data is based on tweet collection which gathered in 2013. We got two runs for the final results which are base on BM25 feedback algorithm. The details of our retrieval model include query expansion, document expansion and retrieval model for the final rank.

Key Words:

Information retrieval, query expansion, Google result, document expansion, BM25

Introduction:

The corpus of TREC 2014 microblog track is the same as the corpus used in 2013. It is much larger than the tweet collection used in 2011 and 2012. Approximately, the corpus includes almost 260 million tweets within a two month period (2/1/2013 - 3/31/2013 inclusive). And we can obtain the whole collection through official API however we can only get 10 thousand relevant tweets for a particular query. The tweets' information is encoded in json format which includes tweet id, screen name, text and retweeted information, etc. Under this condition, our goal seems simple : re-rank the 10 thousand tweets through our own retrieval method to modify original results.

	2011	2012	2013	2014
Number of topics	50	50	60	55
Total documents	16M	16M	260M	260M

We divided the whole task into three sub divisions: query expansion, document expansion and feedback retrieval model.

For the query expansion, there are many ways to implement query items, such as wordNet, Google search API, Bing search API and Yahoo Boss search API, etc. In the end, after several trials, we decided to use Google search results as our main method to expand query instead of other kinds of APIs.

As for document expansion. Because tweets are always short documents and can't more than 140 words, we chose two ways to get document

expansion: implement original tweets with its most relevant tweets and crawl its affiliated link as another implementation of tweet documents. Because affiliated links always related to the tweet content, it is always regarded as introduction and spread of the original tweet.

For the feedback retrieval model, there are many retrieval models to use, including boolean retrieval model, vector space model and language model. After compared the results with different kinds of model, we decided to use BM25 as the main retrieval method because it is more accurate to generate related results based on our whole retrieval system.

System Design:

1. Data Preparation

We download twitter-tools-core [1] project from github, and then when turn it into maven project. We find we can only use command to get the tweet result. Then we modify the java code so that we can use IDE to deal with it. We then decide use eclipse as our IDE and java as programming language to finish the task. After we import the project to eclipse and test the baseline result of former year, we start to use new query set to retrieve tweet ranking lists. For each query, we can get a ranking list with 10,000 tweets. The information of the tweet include 13 items. We extract id, text as most useful information for retrieval model.

2. Query Expansion

In this part, we have considered several ways to expand query such as WordNet or Google search API. However, after we do several trials, we find another way that is not only more accurate but also easier to handle for query expansion, which is key words extracted from Google results.

After we choose a particular query, we use it to get the first 10 Google result pages. Then we crawl these pages, extract their main contents as final documents. And using tf*idf method to rank the weight of each word in these 10 documents. After several experiments for considering the amount of words as query expansion, we find that 10 keywords are enough to support the query. So in the end, we choose the first 10 words ranking in tf*idf retrieval lists besides original words of query itself as the query expansion.

3. Document Expansion

As tweets are short documents(A recent study in Harvard shows that the average words per tweet is 15 words), if we use usual retrieval method, it will lose accuracy. For example, if the query is " Ron Weasley birthday" while a tweet's content is just " ron weasley! ron weasley! ron weasley! ron weasley!", the tweet will rank high under usual retrieval model, but this tweet doesn't match the query well because it doesn't contain the main word "birthday". To avoid this condition, we need to document expansion.

We divide the expansion into two parts. One part is 100 related tweets and the other part is affiliated link.

Due to the short length of a tweet, a way to increase its length is to add related tweets' content to it. Based on former research, we choose the top 100 related tweets for each tweets as tweet expansion.

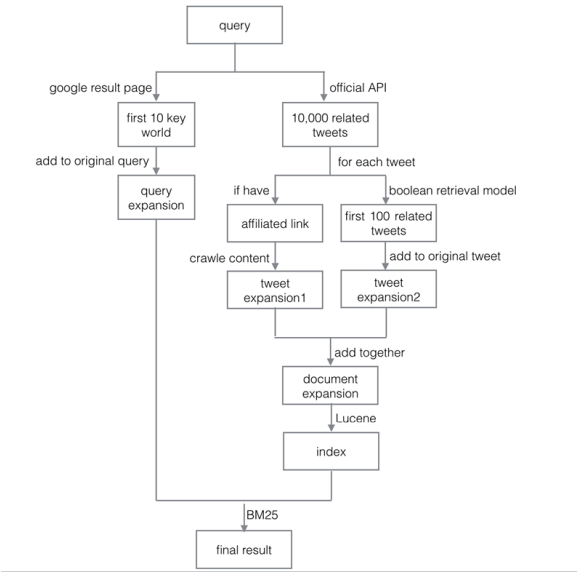
Based on former research, we find if a tweet contains a link at the end of itself, the link can shows more information related to query. In this way, the tweet should weight higher in the final ranking list. So if a tweet contains a link at the end, we crawl the content of the link and then regard it as link expansion.

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

We count these two expansion together with different weight, and re-rank the 10,000 expanded tweets by using BM25 method, then we get the final ranking list as results.

4. Design of Approach

The approach contains several steps. First of all, the query set offered by TREC Conference has 55 different queries. We divided these queries and get their related tweets separately. And then we deal with individual query through the following steps. The graph showed below is an overview of the whole procedure.



First thing is to deal with query. We crawl Google result page of a particular query and get the first 10 Google results. Then we crawl the then Google result pages and extract their main contents and filter stop words. After this step, each query get 10 Google result pages' content. We regard each page as a document and use tf-idf weight method to get the first 10 words which are not appeared in the query. We add the 10 words to the original query and then the new query forms expanded query. We have checked the results of our way and of Google API manually and we find by using our way can get more accurate and related words for the original query.

We use q to denote a query and use t to denote a term in a query. The frequency of term t in a document d is denoted by f(t,d). And tf(t,d) is regarded as the term frequency of t in d. D means the whole documents in the collection c. And N means the number of D. So the algorithm is defined as below:

$$tf(t,d) = \frac{f(t,d)}{\sum_{t \in q} f(t,d)}$$

$$idf(t,d) = log\frac{N - |(d \in D: t \in d)| + 0.5}{|(d \in D: t \in d)| + 0.5}$$

$$tf - idf(t,d) = tf(t,d) * idf(t,d)$$

$$score(q,d) = \sum_{t \in q} tf - idf(t,d)$$

As for the original 10,000 tweets, we need to do refinement on these original tweets. First of all, we filter all retweets and non-English tweets. And then, we use Lucene to build index with stemming and filtering stop words. Then we use the index and vector space model to generate each tweet's related tweet ranking list. We choose the first 100 tweets out of 9,999 tweets and regard them as related. We add the content of 100 tweets to the original tweet to form tweet expansion. We denote $p\ (d_i|d_j)$ as the current model. The algorithm of vector space model is defined below:

$$d_i = (d_{1,i}, d_{2,i}, \ldots . d_{t,i})$$

$$d_j = (d_{1,j}, d_{2,j}, \ldots . d_{t,j})$$

$$p(d_i|d_j) = \frac{d_i * d_i}{|d_i||d_i|}$$

For the link expansion part, we first process the original tweet. If a tweet contains a link at the end of itself, we extract the link, crawl its web page and get the main content as link expansion of a tweet.

In the end, we get two sub-divisions for tweet document expansion: link expansion and tweet expansion.

The next step is to build index for queries. We use each tweet document expansion result as a document. Both link expansion result and tweet expansion result are regarded as a field of a document. We store the two filed in a document and then build index. We use α and β to judge the weight of tweet expansion and link expansion in the final retrieval model. We denote le to link expansion and te to tweet expansion. And the model is defined below.

For the retrieval model , after we considered several classic models, we decided to use a vector space model BM25 as our final feedback algorithm because it has better recall and precision when compared to other method we considered. And it runs faster to get the result than other method in our project. We have 2 runs in the final project and both of them use the same retrieval model with different weight. For the first run Upitt, we set $\alpha = 0.8$ and $\beta = 0.2$, while the other run NewBee is $\alpha = 0.2$ and $\beta = 0.8$. We set the different weight because both of them give a good feedback while we don't how much affiliated link weighs for the final official evaluation.

$$score(q,d) = \sum_{t \in q} idf(t,d) * \frac{f(t,d) * (k_1 + 1)}{f(t,d) + k_1(1 - b + b * |d|/avgdl)}$$

$$p(q|d) = \alpha p(q|le) + \beta p(q|te)$$

Based on classic BM25 algorithm, we set $k_1 = 2$ and $b = 0.75$. And we denote $avgdl$ to average length of a document in the whole collection.

Finally, we use the expanded query to match the 10,000 expanded documents, and generate final ranking list as final result.

Results:

Based on our retrieval model, we have 2 runs in the final. The feedback results are listed below:

Run	R-Prec	MAP	P @ 30
UPitt	0.2020	0.1648	0.4164
NewBee	0.3176	0.2745	0.4485

The statistics in the Ad-hoc runs column are computed over all 77 ad-hoc runs that were submitted to the track. The statistics in Automatic ad-hoc runs column are computed over only the 73 automatic ad-hoc runs. In the task, we mainly use R-Precision, average precision and P@30 to evaluate the results. Judgment pools are created using depth 100 across all submitted ad-hoc runs, plus a random selection of 100 documents per topic from each TTG run. These pools are further reduced by removing retweets (declared irrelevant by track fiat) and then clustered so that textually similar tweets are close to one another (a step taken to enhance judgment consistency). Tweets are judged on a three-way scale of not relevant, relevant, and highly relevant (represented as 0, 1, and 2 respectively in the judgment file). The judgment file, qrels2014.txt, is posted on the tracks page in the active participants' section of the TREC web site.

Evaluation	Automatic ad-hoc runs(73runs)			Ad-hoc runs (77runs)		
	Best	Median	Worst	Best	Median	Worst
R-Prec	0.6659	0.4395	0.0043	0.6668	0.4437	0.0043
AvgPrec	0.6751	0.4155	0.0039	0.6772	0.4208	0.0039
P @ 30	0.8345	0.6261	0.0115	0.8370	0.6309	0.0115

Conclusion:

Our system uses query expansion and document expansion to enlarge the size of each document

in order to get a more clear sense of document. By using Google result to extract key words as expanded query as well as finding relevant tweets and crawling related links as expanded documents. We finally realize the goal of generating better feedback result for a specific query based on tweet collection. However, There are still some flaws in our retrieval system. First of all, it is not easy to consider some particular characters in queries. For example there is a query " Mad Men season 6 ". For the number"6", it is really hard to be considered in retrieval system. Moreover, the retrieval model is just simply used classic model BM25, we don't modify it for the particular task. In the future, we can improve the retrieval model in order to be more suitable for the particular task. And another flow is that when we extract main content from a web page, we create our own method to realize it. After finishing the task, we find by using Boilerpipe library we can get better result for extracting main content of a web page.

Reference:

1. Cheng Li, Yue Yang, Qiaozhu Mei. *A User-in-the-Loop Process for Investigational Search: Foreseer in TREC 2013 Microblog Track* . In TREC,2013

2. Jinhua Gao, Guoxin Cui, Shenghua Liu, Xueqi Cheng. *ICTCENT at Microblog Track in TREC 2013*. In TREC,2013

3. Taiki Miyanishi, Sayaka Kitaguchi, Kazuhiro Seki, Kuniaki Uehara. *TREC 2013 Microblog Track Experiments at Kobe University*. In TREC,2013

4. Zhen Yang, Guangyuan Zhang, Shuyong Si, Yingxu Lai, Kefeng Fan. *BJUT at TREC 2013 Microblog Track.* In TREC,2013

5. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hell-mann. *Dbpedia - a crystallization point for the web of data*. Web Semant., 7(3):154-165, 2009.

6. J. Teevan, D. Ramage, and M. R. Morris. *#twittersearch: a comparison of microblog search and web search*. In Proceedings of the fourth ACM international conference on Web search and data mining, WSDM '11, pages 35-44, New York, NY, USA, 2011. ACM.

7. A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller. *Twitinfo: aggregating and visualizing microblogs for event exploration*. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, pages 227-236, 2011.

8. K. Raman, P. N. Bennett, and K. Collins-Thompson. *Toward whole-session relevance: Exploring intrinsic diversity in web search*. In Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '13, pages 463-472. ACM, 2013.