ICTNET at TREC2017 OpenSearch Track

Peng Xu[1,2,3], Long Bai[1,2,3], Suiyuan Zhang[1,2,3], Fang Yang[1,2,3], Zhibin Zhang[1,2], Xiaoming Yu[1,2], Xiaolong Jin[1,2], Xueqi Cheng[1,2]
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
{xupeng,bailong,zhangsuiyuan,yangfang}@software.ict.ac.cn; {zhangzhibin,yuxiaoming,jinxiaolong,cxq}@ict.ac.cn

1 Introduction

The OpenSearch track explores The "Living Labs" evaluation paradigm for IR that involves real users of operational search engines. The task in the track will continue/expand upon the ad hoc Academic Search task of TREC 2016. It is difficult to define who is better in the ranking experimentation, because the real users in the natural environments search a key word with their own purpose. The best way to evaluate two ranks is let the real users make use of them. So TREC 2017 Open Search Track provides this platform which is a new form to assess the ranks good or bad.

The Open Search provide the training queries, testing queries and candidates documents, but it did not tell us which document is more relevant to a specific query which is necessary to train our model. So first we need to crawl the rank of all the documents on each query from an existing web search engine. Then we try a serial of features in order to find the relevance between the queries and the documents. And we also designed scoring rules to give each document a score. Finally, we used XGBoost to train models for each training query and then found a way to predict testing data based on the models.

Feedback data is the key to this track. We find a simple way to integrate the feedback into our model. Unfortunately, there is so little feedback that can hardly improve the result.

2 Approach

2.1 Crawl Training Data

Since we only know the candidate documents of each query, we should use external information to rank candidate documents. We crawl the top 100 documents for each query from real search engine SSOAR. If the number of returned documents is less than 100, we just retain all the documents.

We discover that the training data and SSOAR returned documents contain various languages, so we uniformly translate them into English by Google Translate. Then we try to match candidate documents with SSOAR returned documents in order to get a ranked document list.

We use the function below to measure the similarity of a candidate document and a returned document.

$$sim_{title}(d_1, d_2) = \frac{\#words\ occur\ in\ both\ document\ titles}{\#words\ occur\ in\ candidate\ document\ title}$$

Practically, we set a threshold at 0.6 to judge if two documents match, which we find all candidate documents match with a distinct returned document.

2.2 train the model

We use 3 types of features to train XGBoost models. topic model similarity, sentence embedding similarity and document meta-data features are used. In order to learn scoring functions, each ranked document is given a score.

2.2.1 Topic Model

Topic Model is an effective way to gain semantic similarities of documents[2]. We considered several models and finally decided to use Latent Dirichlet Allocation (LDA)[3]. We use all the training documents to learn the model and infer the topic distributions of each document and query.

Then we can get the similarity just by calculating the cosine similarity of two topic distributions.

$$sim_{topic}(d, q) = \frac{LDA(d) \cdot LDA(q)}{||LDA(d)|| \cdot ||LDA(q)||}$$

Where $LDA(d)$ refers to the topic distribution of document d.

2.2.2 Document Embedding

We use pre-trained 300-dimensions vectors generated by word2vec[4] as word embedding. Then we use bag-of-words model to represent a document, which embedding is the average of all word embeddings occur in it.

$$\boldsymbol{e_d} = \frac{1}{n}\sum_{w_i \in s} \boldsymbol{e_{w_i}}$$

where w_i represents the ith word in document d which consists n words. $\boldsymbol{e_d}$ and $\boldsymbol{e_{w_i}}$ represents embeddings of the document and the word respectively.

Again, we just use cosine similarity to measure the similarity of document-query pair.

$$sim_{embed}(d, q) = \frac{\boldsymbol{e_d} \cdot \boldsymbol{e_q}}{||\boldsymbol{e_d}|| \cdot ||\boldsymbol{e_q}||}$$

2.2.3 Meta data

We use author, publish time and type and other five meta data as discrete features.

2.2.4 Scoring Rules

We give each ranked document a score in order to learn a scoring function. The score of a candidate document is decided by its position in the returned document list given by SSOAR. This way is like last year's paper[5]. The scoring rules are as follows:

- $\text{score} = 5.0 - 0.05 \times \text{position}$: ranked position between 1 to 5
- $\text{score} = 5.0 - 0.10 \times \text{position}$: ranked position between 6 to 10
- $\text{score} = 4.5 - 0.10 \times \text{position}$: ranked position between 11 to 20
- $\text{score} = 2.5 - (1.0/30) \times \text{position}$: ranked position between 21 to 50
- $\text{score} = 1.0 - 0.01 \times \text{position}$: ranked position between 50 to 100
- $\text{score} = 0$: document isn't in the returned document list.

2.3 predict testing queries

Since we just train models for each training query, how to predict document scores on a testing query becomes a problem. We find the top 5 training queries that is most similar to the testing query, and use their models to approximate the testing query model. The final score of a document on a testing query is the average of its scores given by the five training query models. Cosine similarity between query embeddings is used to determine the relevance of a testing query and a training query.

If a training query is less similar to the testing query, it is also less credible, so we set a threshold at 0.5 to select credible training query models for prediction. A training query model should be abandoned if its relevance to the testing query is less than the threshold, though it may already be the top 5 most similar to the testing query.

2.4 exploit feedback data

Generally speaking, feedback is essential to our system. If a document gets "win", we should increase its score. We will decrease its score if it gets "loss". The score should be maintained unchanged if it gets "tied". Then we will retrain our models to get a better performance. Unfortunately, we get too many "tied" that could hardly improve our system.

3 Experimental Results

Our results are shown in tables below.

Training Data Results for SSOAR

OUTCOME	WINS	LOSSES	TIES	IMPRESSIONS
0.36	3	4	22	29

Testing Data Results for SSOAR

	OUTCOME	WINS	LOSSES	TIES	IMPRESSIONS
Round 1	0.43	5	9	2588	2602

As shown in the figure above, training data results are too small, so the way to exploit feedback is hard to have an impact in train models. Maybe the data that we crawled is from SSAOR, so there are many ties in testing data results. If that is the case, we should use another authoritative web search engine ranks as training data.

4 Conclusion& Acknowledgements

We use external information on the Internet to train models on training data, and use them to predict ranks on testing data. We also considered feedback to improve our system. We also find some shortcomings of our system. First, more external information could be used such as Google Scholar. Second, we train a model for each training query, which may not make full use of training data. If we separate queries into serval types, and learn a model for each type of queries, our system may be more robust. Finally, we did not make good use of feedback because of the lack of feedback.

In a word, we think we have done our best so far, and we expect to get a better result next time.

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, This work is also supported by National Key Research and Development Program of China under grant 2016YFB1000902, and NSF Foundation of China under grants 61572473 and 61772501.

5 References

[1] Tianqi Chen, Carlos Guestrin, XGBoost: A Scalable Tree Boosting System[C]. KDD '16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Pages 785-794.

[2] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation[J]. Journal of Machine Learning Research, 2003, 3:993-1022.

[3] Hoffman M D, Blei D M, Bach F. Online learning for Latent Dirichlet Allocation[C] International Conference on Neural Information Processing Systems. Curran Associates Inc. 2010:856-864.

[4] Yao Yao, Xiaoping Liu, Liu Penghua, Sensing spatial distribution of urban land use by integrating points-of-interest and Google Word2Vec model[C]. International Journal of Geographical Information Science Volume 31, 2017 - Issue 4

[5] Cheng Li, Zhen Yang, David Lillis, BJUT at TREC 2016 OpenSearch Track: Search Ranking Based on Clickthrough Data. The Twenty-Fifth Text REtrieval Conference (TREC 2016) Proceedings