TREC 2020 NEWS Track Background Linking Task

Rahul Gautam, Mandar Mitra, Dwaipayan Roy
Computer Vision and Pattern Recognition Unit
Indian Statistical Institute
203, B.T.Road, Kolkata, India - 700108.
{arr.gee23,mandar.mitra,dwaipayan.roy,}@gmail.com

Abstract

The Background Linking task is a problem that focuses on providing users with suggestions for articles to read next, when the user is reading a news article. The suggested articles should provide adequate context and background information for the article that the user is currently reading. In this paper, we describe several methods that we explored for this task, and report their results.

1 Introduction

This is the third iteration of the News Track at TREC. The News Track consists of two tasks, viz., Background Linking and Entity Ranking. We explore the Background Linking (BL) task. The goal of the BL task is: given a news article, suggest similar articles that provide context and background for the given article. In order to solve the problem, we need to address the following issues:

- *how to extract the most useful keywords from a reasonably long document (i.e., a news report)*
- *how to compute appropriate weights for the extracted terms*
- *how to use the extracted keywords to search accurately*

2 Our method

The dataset provided in this task is the TREC Washington Post Collection (version 3). It consists of 671,947 Washington Post articles from the years 2012-2019. Each article consists of the following fields: ID, TITLE, CONTENT, AUTHOR, ARTICLE TYPE, DATE, and URL. Given an article as input, our method has to retrieve a list of articles, ranked according to their estimated usefulness in providing context and background for the given article. A ranked list is evaluated on the basis of a known list of useful articles (as judged by experts). Methods are compared on the basis of their NDCG@5 scores. All our methods use Lucene 8.3.4 (`http://lucene.apache.org`) to index and query the dataset. For all submissions, we tune our parameters using data from the TREC News 2018 and 2019 tasks. Below, we describe the methods corresponding to our submissions for the TREC 2020 Task.

2.1 IRISINews1: Boosted terms from content

This method consists of extracting K key terms from the query article, and issuing a weighted lucene query against the index to get the result list. We use a simple tf-idf formula to compute weights for terms from the document. For a term t in a document field d, the weight of term

is given by $tf(t,d) \times idf(t)$, where $tf(t,d)$ is the number of occurrences of t in d, and $idf(t)$ is given by $1 + \log((1+N)/(1+df))$. We pick the top words from the *content* field, as it proved to be more useful than the title field in our experiments. By varying the number of words (K) extracted from the content field, we find that $K = 80$ yields the best performance for us.

Using the above query, we compute the score for a document using the BM25[2] formula . Additionally, for each query term, we assign a numerical weight as specified previously. The BM25 score for each matching query term is multiplied by the weight of that term. These scores are added up to get the total similarity score for the document.

2.2 IRISINews2: Document embedding with ranked list merging

In this method, we represent each news article as a vector in an embedding space. For every query article, we prepare a ranked list of articles based on the cosine similarity between the query article and all other articles in the index. We merge the ranked list generated by the method described here with the list produced by IRISINews1 to obtain the final ranked list.

We train a word2vec [3] model on the dataset. Each word in the corpus is represented as a 300 dimensional vector. An article is represented as the sum of the vectors corresponding to the top K words contained in the article. For extracting the top terms, we resort to the tf-idf scoring described for the previous method. We find that extracting $K = 25$ words and summing their vector representations yields reasonably good performance. For scoring, we rank the documents by their cosine similarity with the query embedding. We merge the ranked list from IRISINews1 with the list produced by the cosine similarity ranking using the method described below.

We retrieve 100 documents with the best performing parameters of each method. Let l_1, l_2 denote, respectively, the list of documents retrieved by IRISINews1, and by the embedding-based method. We sum normalize the scores in both the lists. Let the i-th entry in the list l be called $l[i]$, and let the score for it be denoted as $l[i].score$, Next we do the following.

$$\begin{aligned} l_1[i].score &= \lambda \times l_1[i].score \\ l_2[i].score &= (1-\lambda) \times l_2[i].score \\ &\text{where}(0 \leq \lambda \leq 1) \end{aligned} \tag{1}$$

After that we merge the two lists by simply taking their union; if a document appears in both the lists, the score for the document is the sum of the scores from the individual lists. By varying λ, we find that $\lambda = 0.6$ yields the best performance on the 2018 dataset.

2.3 IRISINews3: Use of named entities

In the previous approaches, we used only words picked from the content part of the article. To use the title part, we applied the Stanford corenelp [1] Named Entity Recognition (NER) module to pick core terms from the title.

We generate two queries, one generated by method 1 (IRISINews1) with best performing parameters. Let us call this query q_1. We generate another query by weighting the named entities by their tf-idf score. We call this query q_2. We merge them in the following way, Let $q[i].weight$ be the weight of the ith term of query q, we change the weight of the terms as

$$\begin{aligned} q_1[i].weight &= \lambda * q_1[i].weight \\ q_2[i].weight &= (1-\lambda) * q_2[i].weight \\ &where(0 \leq \lambda \leq 1) \end{aligned}$$

We join the two queries to form the new query with $\lambda = 0.8$ and issue it against the index to get the final ranked list.

2.4 IRISINews4: Merging 3 ranked lists

Here we merge 3 ranked lists, l_1, l_2 and l_3. The list l_1 was generated by issuing a weighted query with top 20 terms extracted from the title. l_2 Was generated by a similar method, here top 50 terms from the content was used to construct the query. The third list was generated by ranking the documents with respect to their cosine similarity with the query document as done in IRISINews2.

We merge l_1 and l_2 using equation (1) with $\lambda = 0.1$. After getting the new list form merging l_1 and l_2, we merged the new list with l3 with $\lambda = 0.6$ to get the final ranked list.

3 Experimental results

In below tables

Submission Name	Description	2018 Score	2019 Score
IRISINews1	Boosted query	0.3964	**0.5263***
IRISINews2	Document vector approach	**0.4041***	0.5131
IRISINews3	Using named entities	0.3993	0.5253
IRISINews4	Merging 3 ranked lists	0.3997	0.5160
htwsaar4	verbose query comprising of title and content	**0.4619****	-
UDInfolab_all	All words and Entities used as query	0.438	**0.606****

Table 1: Comparison between all methods
* Best performance among our methods
** Best performance among all TREC submissions

4 Conclusion

In this paper, we have explained our approach for the TREC 2020 News Track for background linking tasks. The results show that our system performs reasonably well using simple methods. There are further scope of improvement in our methods to obtain even better performance.

References

[1] Stamford CoreNLP library - *https://stanfordnlp.github.io/CoreNLP/*

[2] Robertson, Stephen and Zaragoza, Hugo [*The Probabilistic Relevance Framework: BM25 and Beyond*]

[3] Mikolov, Sutskever, Chen, Corrado, Dean [*Distributed Representations of Words and Phrases and their Compositionality*]