CAB431 Project

Section A Report

Student Name: Jason Zhong

Student Number: 9674985

GitHub Link: https://github.com/zzs493403580/project

Task 1

In the indexing procedure, the “index.bat” file is the script used to run commands to index the documents. The .bat file should be run in cmd in the Terrier’s bin directory which is outside the project folder, otherwise there would be errors, or it can be used after the directory value in the file is fixed.

In terms of pre-processing, the documents were setup firstly to prepare for indexing. Next, stopwords were removed by the “Stopwords” of “Dtermpipelines” option as provided by TREC when indexing. Stemming was not adopted, because it does not have significant influence on the results produced by some methods such as NTLM (for word embeddings later) and instead, it might cause mismatch between stemmed and non-stemmed collections.

The index statistics are shown below:

- Size of vocabulary: 290828
- Number of tokens: 46849318
- Number of indexed documents: 164597

Task 2

The “queryPrep.py” file is to prepare the queries in a format that TREC can use. The “evaluate.bat” script file is to evaluate both TF-IDF and BM25 models, using the query file prepared by executing that python file and 0.4 as the c value in Terrier. The result file names (the counters) may be different from those in the script file and need to be changed due to the difference of computing environments.

After executing the two files in sequence, the average results were obtained as the following table:

Table 1: Evaluation Scores for TF-IDF and BM25

	TF-IDF	**BM25**
MAP	0.1395	0.1393
GMAP	0.0162	0.0157
P@10	0.2554	0.2530
nDCG@10	0.0483	0.0476

As can be seen from Table 1, the overall results from TF-IDF are better than that from BM25. For details:

- The smallest gap between these two approaches is in MAP, which means they have very close ability of ranking relevant documents
- The largest different is in P@10, where TF-IDF retrieved obviously more relevant documents for the first 10 positions.

Therefore, TF-IDF is expected to perform better on finding relevant documents, particularly on the number of relevant documents.

Using the query-by-query MAP data from the evaluation, the “adreData.py” produced the number of queries for different intervals of Gain/Loss growths of MAP for BM25 over TF-IDF (provided that two files for the two models with MAP scores for every query are given), and the chart is presented below:

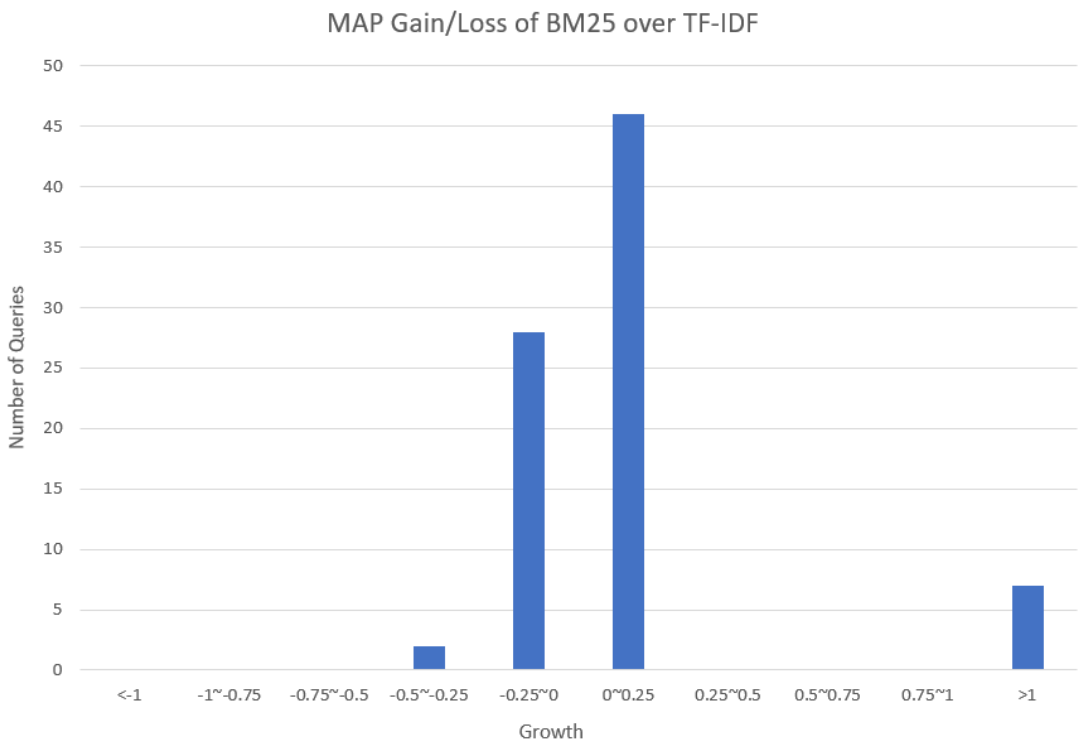

Figure 1: MAP Gain/Loss growths for BM25 over TF-IDF

According to the chart in Figure 1:

- The majority of the queries show only very minor differences of MAP scores between TF-IDF and BM25
- Most queries had less than 25% increases or decreases
- To compare them more precisely, the number of queries with growths in MAP from TF-IDF to BM25 were actually slightly more than those with declines

Although the difference was very small, there were still 7 queries whose MAP rose more than 100%. Thus, it could be concluded that BM25 model performs better in MAP and is more likely to put the most relevant documents at the top positions.

Using the query-by-query MAP data from the evaluation for both models, and sorted by Excel, the gain/loss values of MAP for every query is shown as the following figure:

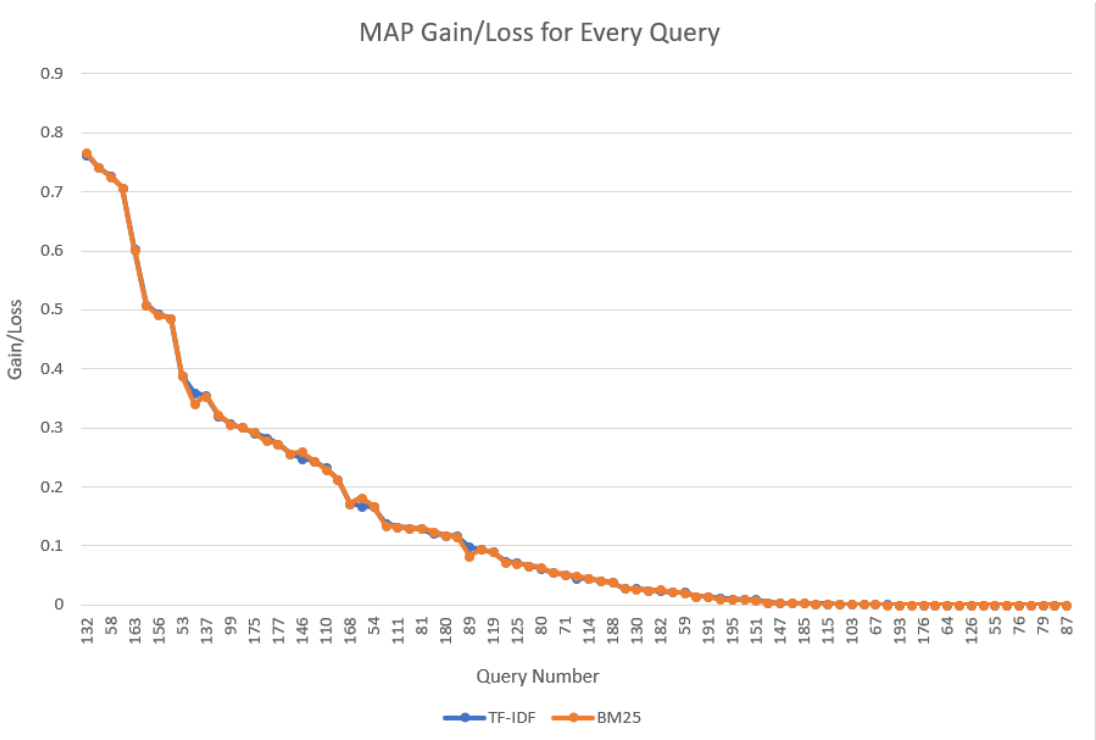

Figure 2: MAP gain/Loss for Every Query Retrieved by TF-IDF and BM25

To compare the difference of MAP gain/loss of the two models for every query, it can be seen in Figure 2 that:

- The gain/loss values of the two models are very close and most of them are almost the same
- Even the largest gap, for query 53, the difference is less than 0.02

Looking at the overall result, there is equivalent occurrences that TF-IDF outnumbered BM25 and BM25 outnumbered TF-IDF. Therefore, it is difficult to point out which of them is better only from this figure, and it is obvious that they have almost the same performance. In other words, they have very close abilities to position relevant documents, which is compatible to the comparisons of previous table and figure.

Task 3

The documents in the folder “AP8889” were preprocessed by “corpusPrep.py” to extract the text, combine them together and fix the format so that Glove can use. The “vectorize.sh” script file (must be executed in Glove's directory and after the corpusPrep.py has been executed to get the corpus file) was used to make vectors files with hidden layers 100, 300, and 700 and window size 10 in conjunction with Glove’s functions. They can be downloaded from:

100-hidden-layer:

https://drive.google.com/open?id=1zAxp367aPJdRDApsV6S-NUfeIMwGx8ux

300-hidden-layer:

https://drive.google.com/open?id=1Jwr0CpBflvwEOjYAc_oNxyoy4kkv5IRB

700-hidden-layer:

https://drive.google.com/open?id=1ekn8VKr1_2rldAH_DCX_WL8rP6_9x7si

Task 4

Adopted Approach:

The retrieval with embeddings was implemented by integrating and evaluating neural word embeddings in information retrieval. As a set of files where each word is with a vector acquired from neural translation language models (NLTM) were given, including CBOW, SKIPGRAM, every word from the corpus were estimated by the translational probabilities to find out the best one to pair with a query word:

$$p_{cos}(u|w) = \frac{cos(u,w)}{\sum_{u' \in V} cos(u',w)}$$

where the *cos(u, w)* is to find the cosine similarity using the vectors of the two paired words. The term with the highest translational probability will be embedded to pair with a word in query.

Code Explanation:

The word embeddings were implemented in the "wordEmbedding.py" file. This file implements a set of methods to prepare word vectors from different vector files, find the best embedded words, expand it to the queries and write the expanded queries to new files for evaluations later.

Stop words were applied so that too common words will not be checked with vector and expanded. Each term in a query other than stop words will be paired with a word which has the highest translational probability to expand the query.

In order to improve the efficiency, for each query word the corpus terms' cosine similarities as well as their sums were pre-calculated only once and stored in a map so that they can be used by directly calling from the map without calculating again and again (more technical details can be seen in the code file with comments).

Use of the Code:

It must be executed in the directory of the "queryPrep.py" file (because it imports that file's methods) and those vector files, to generate new query files. The execution may

last more than an hour (approximately 3 seconds per query) and each query's number will be printed out on the console to show the progress. After it is finished, there will be nine files of new queries (for Cbow, Skipgram and Glove with 10 window size and 100, 300 and 700 hidden layers) in a new folder named "expanded_topics".

Task 5

The "evaluateEmbeddings.bat" script file is used to evaluate the results after the "wordEmbedding.py" file is executed to produce new query files. The result file names may be different from those in the script file and need to be changed due to the difference of computing environments. The evaluation scores are presented in the table below, using 0.4 as the c value in Terrier which is the same as that of the baseline.

Table 2: Evaluation Scores for Three Embedding Methods with Window Size 10 and Different Numbers of Hidden Layers

Embedding Methods	**Cbow**			**Skipgram**			**Glove**		
Number of Neurons	100	300	700	100	300	700	100	300	700
MAP	0.1236	0.1292	0.1311	0.1297	0.1394	0.1506	0.1084	0.1115	0.1082
GMAP	0.0137	0.0155	0.0135	0.0199	0.0207	0.0223	0.0080	0.0090	0.0086
P@10	0.2262	0.2274	0.2162	0.2679	0.2821	0.2964	0.1506	0.1635	0.1565
nDCG@10	0.0415	0.0442	0.0432	0.0481	0.0502	0.0564	0.0259	0.0298	0.0307

As is shown in Table 2, for the three embedding methods:

- Skipgram constantly had the largest evaluation scores with 100, 300 and 700 hidden layers
- Glove shown the smallest values

To focus on the different dimensions:

- With 300 hidden layers, both Cbow and Glove had their best performance
- Skipgram is different and reached the best scores when it was with 700 hidden layers
- However, except for Skipgram, the differences of dimension have little impact on the effectiveness of the other methods (slight impact on Skipgram)

Compared with the baseline in Table 1:

- Only Skipgram from the three methods outperformed the baseline (this is more obvious for 300 and 700 hidden layers)

- All the four scores for that with 700 hidden layers were more than those of baseline

To conclude, among the three embedding methods, Skipgram was the best method that moderately increases the evaluation scores, especially with many (more than 300) hidden layers. Cbow had similar performances to Skipgram but reached its peak at 300 hidden layers, and thereby it was not that sensitive to the number of hidden layer as Skipgram. Glove was the least effective whose evaluation scores were much lower than those of Skipgram and Cbow. But Overall, with 10 window size, dimensions have almost no influence on Cbow and Glove, and only small influence on Skipgram. When it comes to the comparison of Skipgram and baseline, Skipgram with 700 hidden layers had outnumbered the baseline for the four evaluation scores, particularly better at MAP and P@10. Therefore, queries embedded by Skipgram with 700 hidden layers are expected to retrieve more relevant documents and rank them to the top positions.

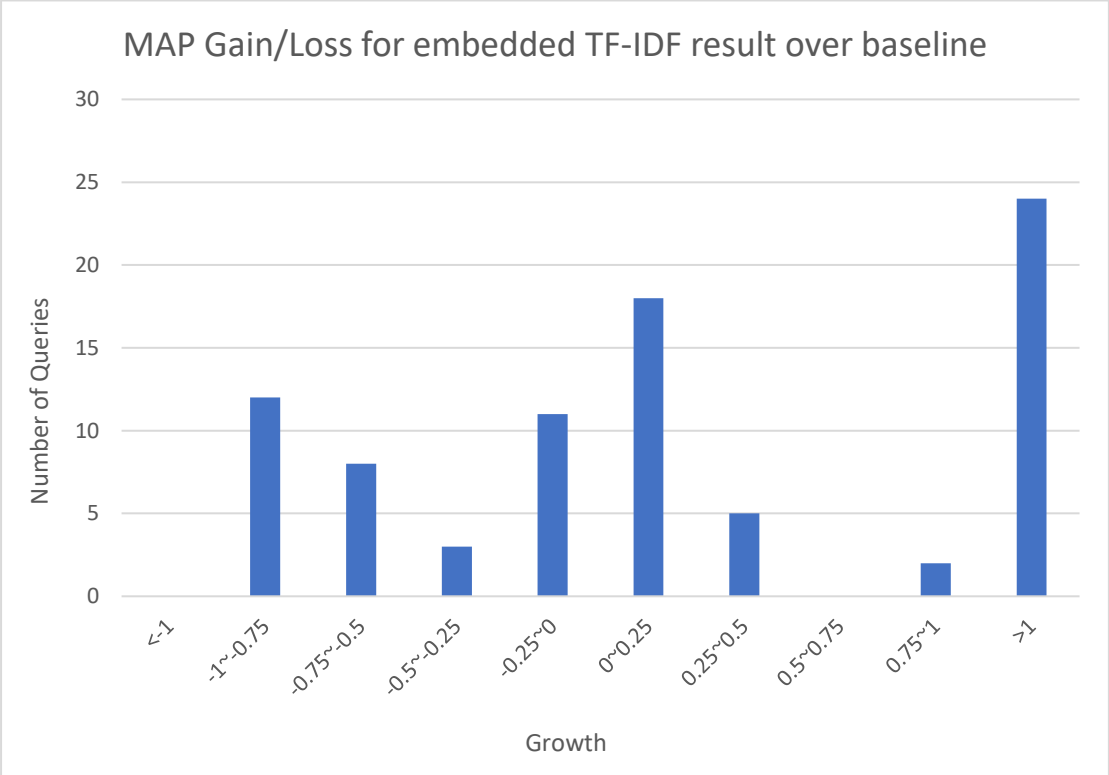

Figure 3: MAP Gain/Loss for TF-IDF after Embedded by Skipgram with 700 hidden layers over TF-IDF baseline

As can be seen in Figure 3:

- The overall number of queries that saw growths of MAP score after applied word embeddings by Skipgram with 700 hidden layers was more than that of baseline.

It is also noticeable that:

- Many queries (almost 25) had increases of more than a double.

Therefore, proper word embeddings did improve the retrieval results of TF-IDF to rank relevant documents to top positions.

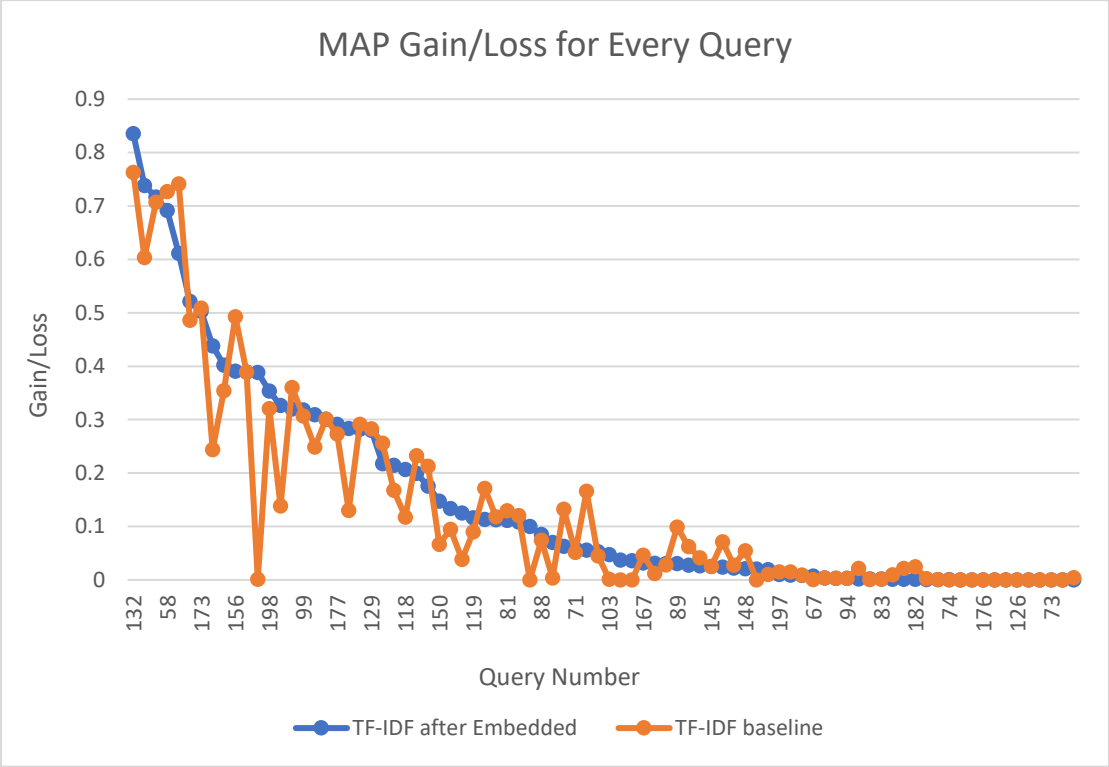

Figure 4: MAP Gain/Loss for every Query After Embedded by Skipgram with 700 hidden layers Compared to Baseline

Figure 4 clearly indicates that:

- Most queries after applied word embeddings performed better than those without word embeddings
- The most dramatic improvement was from nearly zero to almost 0.4.

Therefore, it can be summarized that applying word embeddings (Skipgram with 700 hidden layers) did improve the retrieval results of TF-IDF to rank relevant documents to top positions.

Nevertheless:

- The gap between the two results for every query was also remarkable, either in increase or decrease.

There is very likely to be some query drifts. That is probably because, although sometimes word embeddings can provide very useful expansions for queries, some embedded words can be incorrect but were still compulsively expanded as other words had even lower confidences.