DIS-423 Project 3 · InfoExplorers

Minh Dinh Trong, Anton Balykov, Eric Saikali

14.12.2023

---

1 Introduction

The tasks of information retrieval and document ranking are two classical tasks in Information Systems, which are well studied and can be solved with a number of methods. An immediate approach is to create a tf-idf list for the whole corpus, and decide the similarity between the query and documents.
However, with a massive corpus and computational limitations, this approach is unfeasible. Therefore, in this project, to understand the concept of TF-IDF, we produce our own vectorization method based on the TF-IDF method, which not only accurately retrieves relevant documents but is also highly efficient. Moreover, in order to leverage the document semantics, we cooperated with a pre-trained deep learning model to refine the retrieval list. The result is an impressive recall of 75% just within 8 minutes for both tasks.

The implementation can be found at: Kaggle Link.

2 Description of Process

2.1 Dataset Preparation

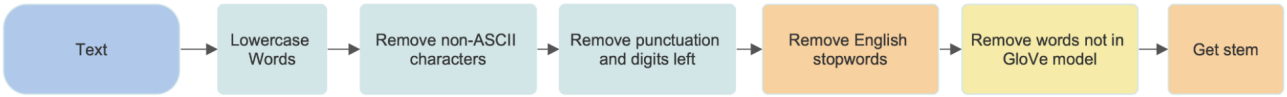
We imported the datasets and imported them to the framework with the pandas library.
For the json files, we dropped the metadata column, as it seems not to contain any data, and strip the text rows so that they do not have any unnecessary whitespaces.
For convenience, we merged the textual data with the queries in both train and test set by inner join.

2.2 Pre-processing

The stage of pre-processing aims to vectorize the given texts into tokenized feature vectors that convey the semantic meaning of the documents. To this end, we decided to extract TF-IDF features for each of these.

Provided to us was a large natural language dataset, including a massive corpus of more than 1.4 million documents and about 500,000 queries. However, when dealing with such an extensive dataset, certain challenges emerged. From a preliminary complexity test, we found that the size of the vocabulary scales linearly with the number of documents being analyzed. This means that the vocabulary size is bound to exceed one million words, posing a serious memory constraint. Additionally, as the size of the TF-IDF vector matrix for each document is directly dependent on the vocabulary size, with the large corpus we were handling, this led to a significant increase in computational cost and the risk of surpassing the 10-minute time limit for retrieval.

Therefore, we decided to reduce the dimensionality of the vocabulary size, following a strict word filter and using several external libraries, as follows:



For different steps in this process, we leveraged external libraries. The Python re (regular expression) library was used for step 3 to remove invalid characters. In steps 5 and 7, we accessed the NLTK library to obtain stop words and a stemming model. Step 6 involved fetching a language model using the Gensim API [1], pre-trained on the glove-wiki-gigaword-50 dataset [2].

2.3 Algorithm Functioning

With these preprocessing steps in place, we proceeded to calculate term frequencies for each document based on the reduced vocabulary. Term frequency represents how often a specific word appears in a given document. To maintain efficiency, we opted for scipy's lil_matrix, a sparse matrix data structure, for storing these term frequencies. This choice was essential for managing memory constraints efficiently.

The next step involved calculating the inverse document frequency (IDF). The IDF values were straightforward to obtain by counting the number of zero entries in each column (representing vocabulary terms) and calculating the logarithm of the inverse ratio to the total number of documents.

Subsequently, the TF-IDF matrix was constructed by multiplying the term frequency matrix with the IDF values, resulting in a sparse matrix representation of TF-IDF scores for the entire corpus.

As these features are completely reliant on the static corpus, we save a pickle of the TF-IDF matrix, along with the compact vocabulary and the IDF matrix for later use. Rather than saving to csv, we save it as a pickle file because it allows us to efficiently store and retrieve Python objects without the need to parse or convert data formats. Thanks to this efficient process, it took us around 20 minutes to pre-process all 1.4 million documents.

In order to process queries, we then convert them to vectors in the same fashion in order to sort out 5000 best documents per queries thanks to tf-idf this function therefore maps from 1.4 million to 5000 documents.

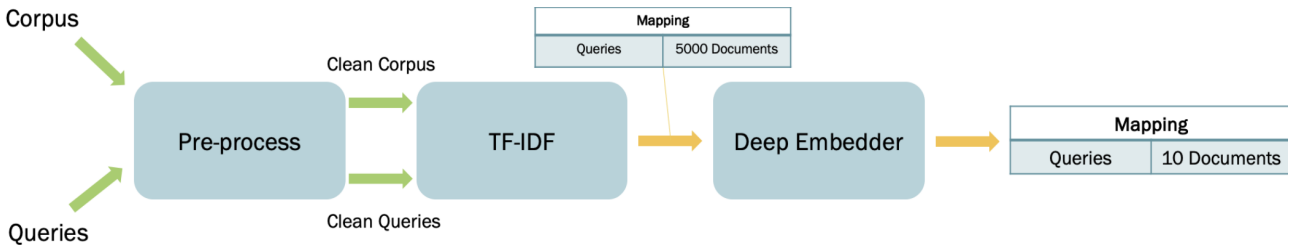
Since the list of candidates is relatively much smaller than the original corpus, we can leverage heavier but more powerful Deep Learning models for a good overall result.

That is why we decided to get text representation from the pre-trained model based on Microsoft's MiniLM model [3]. This model is finetuned on over 1 billion sentence pairs, which were obtained from Reddit Comments, Stack Exchange, Yahoo Answers,... [4]. From our research, this dataset is different from the provided corpus and not dependent for the task of information retrieval.

Using the reduced map of 5000 documents, the algorithm feeds it to the MiniLM model for semantic embeddings on latent space, and the saved embedding of the corpus to pin down for each queries, 10 best documents from the 5000 given.

On a side note, encoding the corpus using the deep embedder takes 25 minutes approximately.

Below is a graph showing the different steps to enable Task 1 total processing.



3 Results & Conclusion

3.1 Task 1

To realistically understand the performance of our retrieval system, in terms of recall and processing time, we took the first 7437 queries from the training set so that it has the same size as the test set. The following analysis is based on this smaller validation set.

Thanks to our rigorous filtering process, the resulting vocabulary has only 135,442 words. Table 1 shows some examples of the tokenized text. As can be seen from these examples, the tokenized words are key words in the queries, important to extract the meaning of these sentences.

query-id	text	processed
1185869	)what was the immediate impact of the success of t…	[immedi, impact, success, manhattan, project]
1185868	_____ justice is designed to repair the harm t…	[justic, design, repair, harm, victim, commun, off…
597651	what color is amber urine	[color, amber, urin]
403613	is autoimmune hepatitis a bile acid synthesis diso…	[autoimmun, hepat, bile, acid, synthesi, disord]
1183785	elegxo meaning	[mean]

Table 1: Examples of processed queries

However, we also see the limitation of our preprocessing technique. Since the text pre-process is made for general cases, the model has difficulty to find relevant documents corresponding to queries on numbers, for example, "$\frac{3}{5}$ of 60" (query-id:1288) or documents with unusual association of words & characters (corpus-id:4196638). The resulting query is just empty list, as in the following snippet:

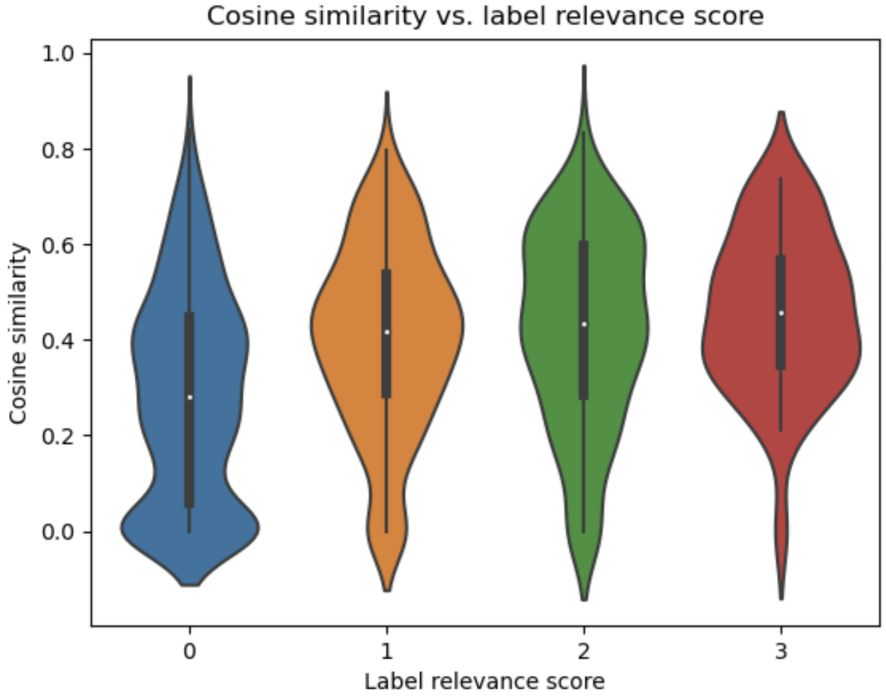
```
In 50   1  queries2[queries2['query-id'] == 1288].processed
           Executed at 2023.10.21 19:10:44 in 4ms

Out 50     Series([], Name: processed, dtype: object)
```

The graph below shows recall values at different choices of k of our first step using TF-IDF feature and cosine similarity to decide the relevance of the documents. As mentioned earlier, the higher k is, the better the recall. We chose k=1000 because this gives us a reasonable trade-off between accuracy and complexity for later steps. The simple yet fast tf-idf allows to discard completely unrelated

As mentioned earlier, the higher k is, the better the recall. We chose k=1000 because this gives us a reasonable trade-off between accuracy and complexity for later steps. The simple yet fast tf-idf allows to discard completely unrelated documents per queries.

3.2 Task 2

For task 2, we are visualizing the label relevance score and our predicted scores for each document. It can be seen that our scores reflect the labeled relevance of the documents, the higher cosine similarity, the more likely it has high relevance score. In the distribution of similarity values for score 0, it is clear that a remarked proportion of these values is 0. On the other hand, for the higher scores, although the discrepancies among these groups were not too clear, for higher scores, while the distinctions among these groups are not very pronounced, the majority of values fall within a higher range, between 0.3 to 0.5. Moreover, the average for each group increases substantially: 0.28, 0.40, 0.42, and 0.45 for 0, 1, 2, and 3, respectively.



References

[1] *How to download pre-trained models and corpora.* [Online]. Available: https://radimrehurek.com/gensim/auto_examples/howtos/run_downloader_api.html.

[2] *Glove: Global vectors for word representation.* [Online]. Available: https://nlp.stanford.edu/projects/glove/.

[3] *Minilm.* [Online]. Available: https://github.com/microsoft/unilm/tree/master/minilm.

[4] *All-minilm-l6-v2.* [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.