Wonyoung Lee

Group Assignment 2 Report: Literature Organization
(Search Engine for COVID-19 Open Research Dataset (CORD-19))

The data we were given was a collection of 33,000 json documents that each contained a title, abstract, body text and more. Our task was to build a natural language processing and information retrieval program that would take in this collection and input the query and output a list of the 100 most relevant documents from the corpus. This was done in three big parts: parsing, organizing, and retrieving. The following paragraphs explain our model in more detail.

For the parsing of documents, first, we tokenized the sentence and removed punctuations. We took this step because we believed that punctuations do not carry important information in finding query relevant documents. We also took this step first because our program calculates the relevance of a document to the query on a word by word basis. Therefore, we thought that tokenizing first would make it easier to process each word of the query when we get the query. After tokenizing and removing punctuations, we removed stopwords, which are words like “to” “and” “the” that carry more of a grammatical function than conceptual meaning. We took this step because all documents have these words to some extent and measuring based on these stopwords can be misleading let alone inefficient. After stopword removal, we stemmed words, meaning we got rid of letters that are not core to the meaning of the word but only serve a grammatical purpose. For instance, the letters “s” or “es” at the end of a plural word was removed. After all these steps were taken, we had a collection of parsed documents. Then, we extracted the titles and abstracts and stored them in separate csv and pickle files for future use. We extracted the titles and abstracts because they are the shortest yet most informative parts of an article. We attempted to handle the whole body text but this took an unbearably long time and we decided that for time efficiency, we should parse and extract title and abstract information and that this would be enough.

After the preprocessing step, we organized the parsed information. We changed our organization method as we moved forward. We initially made an inverted index that kept the TF-IDF score of a document to a word and used the TfidfVectorizer() function and applied this to all existing words in the corpus. We chose to calculate the term frequency and inverse document frequency because we believed that measuring these scores would be more time and space efficient than unsupervised learning methods such as word embeddings. Here, we also additionally incorporated a weight difference to the title TF-IDF scores and abstract TF-IDF scores. After completing the program and trying out different ratios, we decided that the title to abstract weight ratio should be 5 to 1. However, after designing retrieve, we decided that instead of using the TF and IDF scores, we should use the BM25 method to also account for document length and text saturation and help retrieve a more accurate result. Therefore with this organized data, when we took in queries, we were able to apply the BM25 on each word of the query to the doc.

In the retrieval, we designed both the Vector Space Model, which calculates the distance between docs and query using our initial TF-IDF score inverted index, and the BM25 model which calculated the TF-IDF scores taking into account the two additional parameters mentioned above. However, after trials, we found the BM25 model to be more successful and accurate (by ~15%). We believed that this was due to the fact that the Vector Space Model does not account for document length and text saturation. In conclusion, we discarded our vector space model and chose the BM25 model for our retrieval process. More specifically, in our retrieval function, we take in the query and parse them (so that their form matches the parsed words in the documents) and input them into the bm25() functions.

We evaluated our program by matching up the titles of the 100 documents our program produced to the first 100 documents we got after searching the given keywords like "coronavirus immunity" into the CORD-19 website. Our program had up to 48 matches out of 100 when compared to the website.

There were some design choices we had to make because we were faced with challenges. First, we found out that different languages also existed in the database. And without any grammatical knowledge to parse these languages, we decided to ignore the documents that were written in different languages. This may have caused us to lose information and we believe that we can improve on this in the future by using translation methods or absolutely unsupervised models. Second, we found out that some of the titles and abstract information were either empty or made of special characters without meaning. Therefore, we made sure to include if and else if cases that skipped over these when organizing the data. Third, time complexity was very big when we first built the program. However, by using pickles and csv files in the extraction and organization, we were able to save preprocessing time. Even with this change, however, the runtime was too large to include the body text. Therefore, we believe that a good point of improvement could be to use less for loops by using different data structures or preexisting functions that run on constant time in the program and enhance the runtime so that we can also examine the body text. We were able to also think about additional enhancements such as making the query input be easier for users and making the result be easier for users to understand and access the documents such as including hyperlinks to the document.

In conclusion, our program used on BM25 model the weighted TF-IDF scores of the titles and abstracts of the documents to find the 100 most relevant documents to the inputted queries.

We have submitted our project to the kaggle challenge.
The link to our project is here: https://www.kaggle.com/joshreitz/biol1595-assign2