Team Members: Enson Palacios

Git Link: https://github.com/epalacios1630/Final-Project.git

Final Project Report

For my project, I went with the jeopardy idea. I was given files of Wikipedia pages and Jeopardy questions and I built a model that when given a query returns the title of an article. I shortened all the contents of each article, lowercased words, removed punctuation, removed stop words, stemmed, and lemmatized. Afterward, I used ltn.lnc to score each document for each query. I then attempted to use an AI LLM to rerank all my results to obtain a better score.

The first part of the project was to preprocess everything. The goal was to shorten the token list per article to be as small as possible so that post-processing goes quicker. I had a collection of Wikipedia text files; the first part of preprocessing was creating a list of text for each document. By skimming the documents, I was able to see that each article started with "[[**article_name**]]\n", this was then followed by the contents of the article. I iterated through every wiki text file in my directory (from smallest ending num to largest) and collected a list of article titles and their respective contents. I created a unique ID for each document title and an ID to title dictionary that I used throughout the program. As I browsed through the files, I noticed lines that I didn't want to be included in the contents, they were formatted this way:

| title = The Fifteen Decisive Battles of the World: From Marathon to Waterloo

| url = http://ebookbrowse.com/sir-edward-creasy-the-fifteen-decisive.....

Many of the wiki documents contained sections of metadata formatted like this. Some of these were external links to sources, information about books (publication date, title, pmid), or seemingly nonsense information. While it is plausible that some of this information could be helpful, it seemed that the rest of the content had the bulk of identifying features. I made the executive decision not to keep in these lines, by doing this a lot of irrelevant information was removed and post-processing ran faster. There were also lines formatted like: "=**subsection**\=\n", which were subsections of the article. They were then followed by the contents of said subsections. Many of these subsections had umbrella terms that weren't specific ("Science", "Development", "History", "Culture"). In addition, some subsection names were irrelevant ("See also", "References", "External links"). That's not to say that there were subsections with important information ("World Crokinole Championship", "Relationship between luminous intensity and luminous flux" …). However, I found that most of the terms in the subsection were present in the content of that subsection anyway. In conclusion, I felt it was a net positive removing these lines, so I did. Now that I had a collection of article titles and their respective contents, I reduced the number of terms per article.

I converted the string list into a panda data frame and then applied several transformations to the frame. I first removed all the punctuation and lowered the text from each

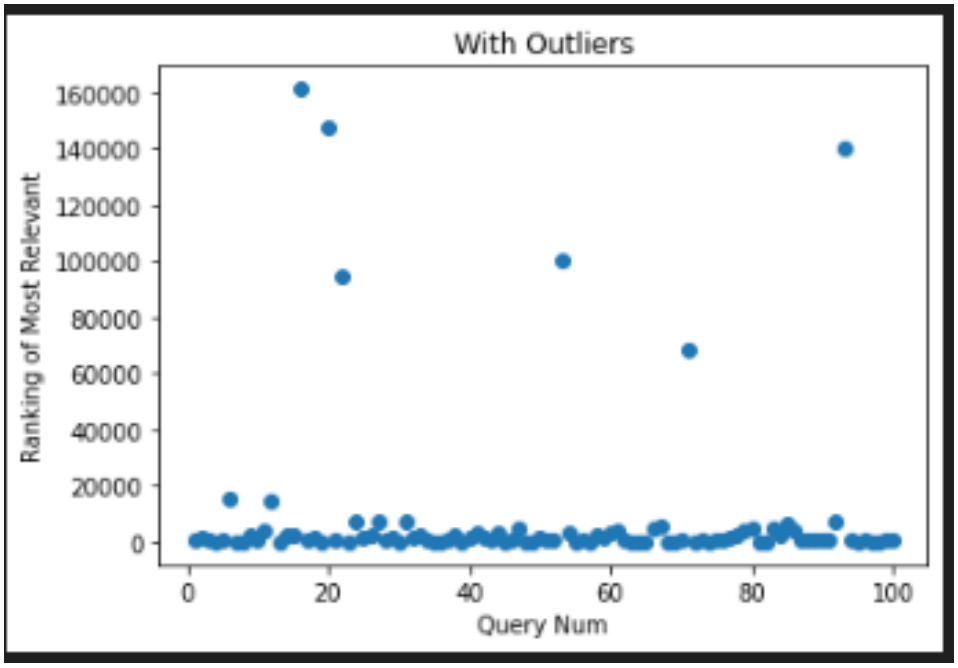
doc. A problem that I came across was finding a Python module that removed stop words from text, I ended up finding a JSON file that contained a list of these words and imported it into my notebook. Next, I browsed online and found a natural language processor called **nltk** that contains a stemmer and applied it to each doc. After that, I used the lemmatizer from **nltk** and applied it to each doc. This process took 80-100 minutes, but I stored the results in a file. Now that I had finished preprocessing all the wikidocs I took a look at the questions file. The file had blocks where the first two lines contained the questions, and the last one had the answers. I separated the answers from the questions (the answers will only be looked at to compare results, they're not used in the query itself) and remembered to apply all the same transformations to the query.

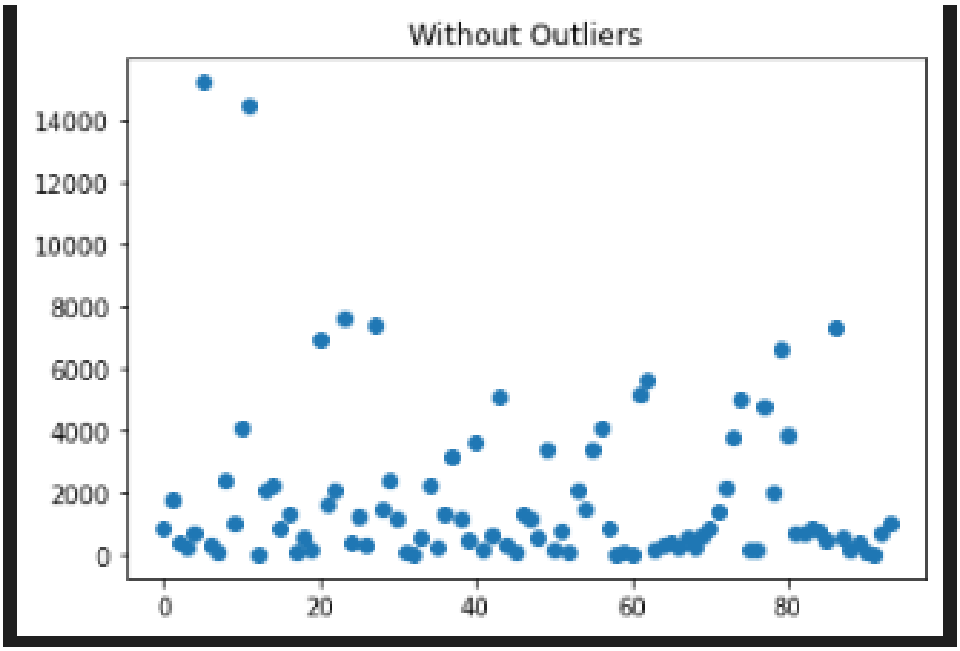
For the first stage of post-processing, I used tf-idf (specifically ltn.lnc) to generate a list per query that had all the scores, and all the doc ids associated with those rankings. Even though I had obtained a list of tokens per document and heavily shortened said lists, the tf-idf rankings still took a while. During pre-processing I found there were over 279,000 different Wikipedia articles and I had to make ltn calculations based on each term in each article. Then there were 100 queries, so I had to calculate ltn.lnc scores for each one. This whole process took about 40-60 minutes of runtime. Because I didn't want to take long to run my program each time, I created a file using the pickle module to store the finished results. Now that I calculated everything, I wanted to update this list so that it would return more precise results.

In my second stage of post-processing, I wanted to implement a reranker. I planned on implementing an LLM to generate a new list of ranks for the top ranks in our current rankings. To do this I downloaded the **langchain** module online and created an account with **OpenAI** so that I could access a key that was necessary for the text embedding process. Within this module, I used the class FlashrankRerank() as the compressor and OpenAIEmbeddings() to help create the retriever. I put everything together so that the compression retriever would return a new list of ranks using OpenAI, and the text and query data we already had. The problem with this is that there was too much information to process, and OpenAI has a limit to how much it can be used without paying. I tried reranking the top 10 docs per query, but I received a, "**You exceeded your current quota**" error. Next, I attempted to use other embedding models so I could implement the reranker, and I found other similar ones (like Google VertexAI, and Ollama) but from what I researched they all seem to have the same limit problem. It became immediately clear to me that I wouldn't be able to make any progress with this process.

I obtained an MRR score of 0.01056 and a precision@1 of 0. The mean of all relevant ranks was 8874.67, the standard deviation of the ranks was 28972.6, the count of relevant ranks under 1000 was 53, and those under 20,000 were 94. Looking at all the data there were a few big outliers when ranking which brought the average down. There were over 270,000 articles so clearly my preprocessing and scoring did something but not enough. I haven't talked to other teams so I'm not sure if the scoring I got based on the preprocessing is normal and the reranking via prompting is what boosts the results substantially. In conclusion, my results weren't as great

as I'd hoped they'd be, in the future, I intend on gaining a better understanding of AI prompting and finding a better open-source one.





Sources:

https://python.langchain.com/v0.1/docs/integrations/retrievers/flashrank-reranker/

https://python.langchain.com/docs/integrations/text_embedding/

https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

https://www.geeksforgeeks.org/python-stemming-words-with-nltk/