Maithili Luktuke
CSE 272
5/7/23
HW 1 – Search Engine

Software:

The code was written in entirely in Python on a Google colab notebook. This code is written on a Macbook Pro with MacOS Big Sur installed on it. Firstly, several Python libraries were imported for this code. The first block of the code in the colab notebook shows all the imports. The most important imports are re, nltk, and pickle. The re library in python provides regular expression matching operations. The nltk library in python is used for symbolic and statistical natural language processing for English. The pickle library in python is used to serialize and deserialize a Python object structure.

Parsing the Documents:

Firstly, all the documents are read into the program. Then, the documents are separated by the document ID and then the rest of the document content. After that, the document is cleaned by removing any non-word characters such as any numbers or symbols that are not needed for the search engine to run properly. The text is also changed to all lowercase to make it easier to go through. Then, stopwords which come from the nltk library are removed from the document contents. The list of stopwords is extended to include the following: ['.U', '.S', '.M', '.T', '.P', '.W', '.M', '.I']. The cleaned document is then tokenized.

Index and Inverted Index:

The Python pickle library was used in order to create the index and the inverted index. Using the pickle library helped reduce processing time and made it so that the index wasn’t created every single time the colab notebook was run. Once the documents are cleaned the next step is to create the index and the inverted index. The index is created by mapping each document to its unique words and those words’ occurrences in the document. Then this index is used to create the inverted index. The inverted index maps each unique word to the document that it occurs in. When the program goes through the index to create the inverted index, the occurrences and the positions of the terms are recorded along with the document that that term is in. Each index was stored using an 8-digit ID that was taken from the document. It takes over 2 minutes for the index and inverted index to be made.

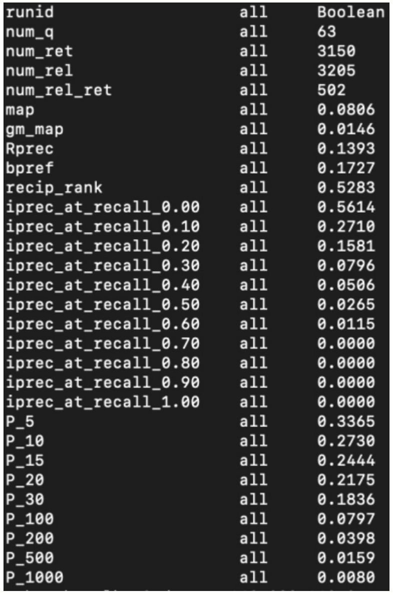
Parsing the Queries:

The queries which were given were parsed in basically the same way that the documents were parsed. As with the documents, the queries are read into the program. Then, since there are different headers in the queries such as <top>, <num>, etc. the queries are cleaned by taking those headers into account. The same cleaning as with the documents happens here as well; removing non-word characters and stopwords. Then the queries are stored along with their ID.

Ranking Algorithms:
Four different ranking algorithms are implemented here. They are as follows: Boolean, Tf, Tf_idf, and my own custom algorithm. For the custom algorithm, I used the Tf_idf scores and then multiplied each score by 2 and subtracted the the log of each score from that which would look like the following: (tf_idf score * 2) – log(tf_idf score).

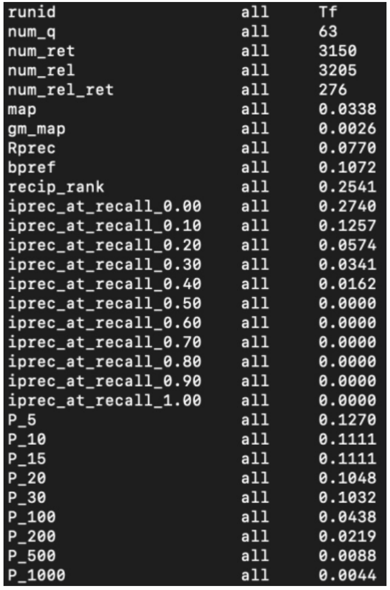
Experimental Results:
A log file was created for all of the ranking algorithms. The top 50 documents were extracted for each query in the log file. Then, these documents were evaluated using the TREC Github Repository code. The results are displayed below.

Boolean ranking:



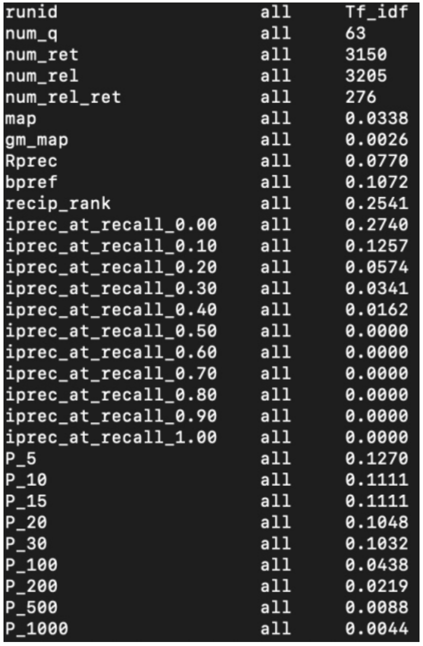
```
runid                   all     Boolean
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     502
map                     all     0.0806
gm_map                  all     0.0146
Rprec                   all     0.1393
bpref                   all     0.1727
recip_rank              all     0.5283
iprec_at_recall_0.00    all     0.5614
iprec_at_recall_0.10    all     0.2710
iprec_at_recall_0.20    all     0.1581
iprec_at_recall_0.30    all     0.0796
iprec_at_recall_0.40    all     0.0506
iprec_at_recall_0.50    all     0.0265
iprec_at_recall_0.60    all     0.0115
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.3365
P_10                    all     0.2730
P_15                    all     0.2444
P_20                    all     0.2175
P_30                    all     0.1836
P_100                   all     0.0797
P_200                   all     0.0398
P_500                   all     0.0159
P_1000                  all     0.0080
```

Tf Ranking:



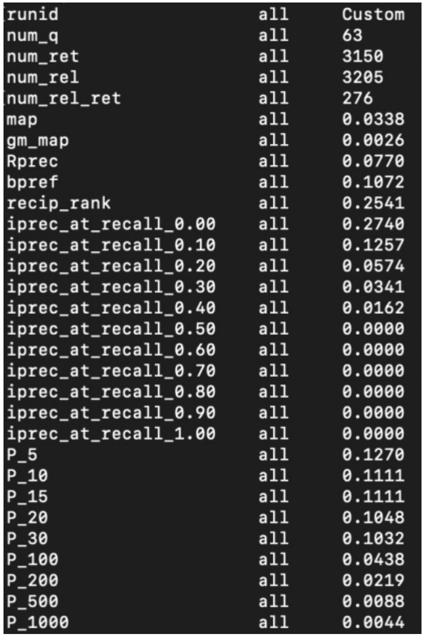
```
runid                   all     Tf
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     276
map                     all     0.0338
gm_map                  all     0.0026
Rprec                   all     0.0770
bpref                   all     0.1072
recip_rank              all     0.2541
iprec_at_recall_0.00    all     0.2740
iprec_at_recall_0.10    all     0.1257
iprec_at_recall_0.20    all     0.0574
iprec_at_recall_0.30    all     0.0341
iprec_at_recall_0.40    all     0.0162
iprec_at_recall_0.50    all     0.0000
iprec_at_recall_0.60    all     0.0000
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.1270
P_10                    all     0.1111
P_15                    all     0.1111
P_20                    all     0.1048
P_30                    all     0.1032
P_100                   all     0.0438
P_200                   all     0.0219
P_500                   all     0.0088
P_1000                  all     0.0044
```

Tf_idf Ranking:



```
runid                   all     Tf_idf
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     276
map                     all     0.0338
gm_map                  all     0.0026
Rprec                   all     0.0770
bpref                   all     0.1072
recip_rank              all     0.2541
iprec_at_recall_0.00    all     0.2740
iprec_at_recall_0.10    all     0.1257
iprec_at_recall_0.20    all     0.0574
iprec_at_recall_0.30    all     0.0341
iprec_at_recall_0.40    all     0.0162
iprec_at_recall_0.50    all     0.0000
iprec_at_recall_0.60    all     0.0000
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.1270
P_10                    all     0.1111
P_15                    all     0.1111
P_20                    all     0.1048
P_30                    all     0.1032
P_100                   all     0.0438
P_200                   all     0.0219
P_500                   all     0.0088
P_1000                  all     0.0044
```

Custom Ranking:



```
runid                   all     Custom
num_q                   all     63
num_ret                 all     3150
num_rel                 all     3205
num_rel_ret             all     276
map                     all     0.0338
gm_map                  all     0.0026
Rprec                   all     0.0770
bpref                   all     0.1072
recip_rank              all     0.2541
iprec_at_recall_0.00    all     0.2740
iprec_at_recall_0.10    all     0.1257
iprec_at_recall_0.20    all     0.0574
iprec_at_recall_0.30    all     0.0341
iprec_at_recall_0.40    all     0.0162
iprec_at_recall_0.50    all     0.0000
iprec_at_recall_0.60    all     0.0000
iprec_at_recall_0.70    all     0.0000
iprec_at_recall_0.80    all     0.0000
iprec_at_recall_0.90    all     0.0000
iprec_at_recall_1.00    all     0.0000
P_5                     all     0.1270
P_10                    all     0.1111
P_15                    all     0.1111
P_20                    all     0.1048
P_30                    all     0.1032
P_100                   all     0.0438
P_200                   all     0.0219
P_500                   all     0.0088
P_1000                  all     0.0044
```

As can be seen form the trec_eval results above, the Boolean ranking algorithm worked the best. The number of relevant documents returned is the most for the Boolean ranking algorithm. If this search engine was built with the Apache Lucene library, it would have worked better thus achieving better results. Under optimized implementations of these algorithms, the others probably would have performed better.

Learning:

From this assignment, I learned a great deal about how documents can be parsed and cleaned. I learned about several different Python libraries which I had not used before such as nltk, re, and pickle. Implemented the search engine was fun because I got to see a peak inside how a search engine is built and how much goes into making one work properly.

Github:

The following is the github link to the code: https://github.com/mluktuke/CSE-272_HW-1_Seach-Engine