Homework 1
Search Engines

Anirudh Potlapally
1851180

Software:

The assignment is written in Python and executed within a Jupyter Notebook environment. The code is developed on a Windows 10 operating system. Several Python modules were utilized for different functionalities:

- re (Regex): Used for regular expression operations.
- pickle: Used for object serialization and deserialization.
- collections: Used for specialized container datatypes like defaultdict and Counter.
- pandas: Used for data manipulation and analysis.
- string: Used for string-related operations and constants.
- tqdm: Used for creating progress bars and monitoring iteration progress.
- nltk: Used for natural language processing tasks and functionalities.

Link to the code

Data Parsing:

The data parsing process begins by reading the documents stored in a text file. Each document consists of a document ID followed by the remaining text content. The document ID is separated from the text.

Next, stop words and specific markers such as ['.U', '.S', '.M', '.T', '.P', '.W', '.M', '.I'] are removed from the text. These stop words and markers typically carry little or no significant meaning in the context of the analysis. Afterward, the text is cleaned by removing punctuation marks, converting the text to lowercase, and eliminating extra spaces between words. This cleaning process helps standardize the text and remove irrelevant variations.

Finally, the cleaned text is transformed into a list of individual words. This step involves splitting the text string into separate word strings, allowing for further analysis and processing of the textual data.

The process of parsing query documents follows a similar approach. However, in this case, there is an additional task of identifying specific headers within the query documents.

The parsing starts by reading the query documents, typically stored in a text file. The content of each query document is examined to locate specific headers such as '<top>', '<num>', '<title>', and others. These headers provide essential information about the query, such as the query number, title, and additional details.

Like the data parsing process, the text within the query documents may also undergo cleaning steps, such as removing punctuation, converting to lowercase, and eliminating extra spaces. These steps help prepare the query text for analysis and retrieval tasks.

Index and Inverted Index:

After obtaining the clean documents, the next step involved indexing and creating an inverted index. Python dictionaries were predominantly used to implement this task.

The indexing process began by creating an index, which establishes a mapping between each document and its unique words and their respective occurrences. Subsequently, the index was utilized to construct the inverted index. The inverted index maps each unique term to the documents in which it appears. The creation of the inverted index involved iterating through the index and processing each document's terms. For every term encountered, its occurrence was recorded, along with the corresponding document. If the term had been encountered previously, the document and its associated information were appended to the existing entry in the inverted index. Otherwise, a new entry was created for the term in the inverted index.

By utilizing the dictionaries' key-value pairs, the indexing process facilitated efficient storage and retrieval of information, enabling quick access to the documents containing specific terms and their respective occurrences and positions.

Each index file was associated with an 8-digit ID derived from the document structure. This ID ensured a unique identifier for the corresponding index and inverted index files. The ID helped maintain consistency and facilitated easy retrieval of the correct index and inverted index for a given dataset. To optimize the processing time and avoid recreating the index every time the notebook is run, I employed the Python pickle library to store the index and inverted index in files.

By leveraging the pickle library and storing the index and inverted index in separate files, I could significantly reduce the processing time required to generate the index, enhancing the overall efficiency of the data parsing and analysis workflow.

Scoring and Ranking Algorithms:

During the implementation of the search engine, I incorporated four different ranking algorithms to retrieve documents:

- **Boolean Ranking**
- **TF Ranking**
- **TF-IDF Ranking**
- **Custom Relevance**
- **Custom Combination Ranking**

The **Custom Relevance Ranking** involves a modified scoring criterion that enhances the relevance of documents based on a previous search. The algorithm follows these steps:

1. Initially, the TF-IDF ranking algorithm is applied to the entire document collection, and the top k-most relevant (50) documents are identified based on their TF-IDF scores.

2. Next, a new scoring process uses only the previously identified relevant documents. This step focuses on refining the relevance scores based on the previous search results.

Each relevant document is assigned a relevance score using the following formula:

3. relevance_score = (tf-idf_score + discount_factor^relevance_exponent) - log(tf-idf_score)

    - The tf-idf_score represents the original TF-IDF score of the document.
    - The discount_factor is a constant value (0.75) raised to an exponent determined by the relevance of the document from the previous search.
    - The relevance_exponent amplifies the effect of the discount_factor based on the relevance of the document.
4. The log(tf-idf_score) term is subtracted from the overall score to reduce the impact of high TF-IDF scores and create a more balanced ranking.

The **Custom Combination Ranking** algorithm combines the strengths of the TF-IDF ranking and the boolean ranking methods to mitigate their respective limitations. By incorporating the relevant documents identified through TF-IDF ranking, the algorithm leverages the boolean ranking algorithm to determine the final ranking scores.

This approach capitalizes on the advantages of TF-IDF ranking, which considers the importance of terms in a document relative to the entire corpus, and the boolean ranking, which focuses on the presence or absence of terms in a document. By combining these two methodologies, we aim to enhance the overall effectiveness of the ranking process.

Through this custom combination approach, we strive to address the potential weaknesses of individual ranking algorithms and achieve more comprehensive and accurate document rankings. By leveraging the strengths of each method, we aim to improve the overall performance and relevance of the ranking results.

Experimental Results:

As per the assignment requirements, I performed the following steps:

1. Created a separate log file for each query, covering all documents.
2. Extracted the top 50 documents for each query.
3. Evaluated the extracted documents using the TREC GitHub Repository code.

The results can be seen in Figure 1.

runid	all	Boolean
num_q	all	63
num_ret	all	3150
num_rel	all	3205
num_rel_ret	all	491
map	all	0.0734
gm_map	all	0.0121
Rprec	all	0.1353
bpref	all	0.1652
recip_rank	all	0.5133
iprec_at_recall_0.00	all	0.5471
iprec_at_recall_0.10	all	0.2647
iprec_at_recall_0.20	all	0.1566
iprec_at_recall_0.30	all	0.0476
iprec_at_recall_0.40	all	0.0387
iprec_at_recall_0.50	all	0.0216
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	0.2825
P_10	all	0.2540
P_15	all	0.2370
P_20	all	0.2143
P_30	all	0.1847
P_100	all	0.0779
P_200	all	0.0390
P_500	all	0.0156
P_1000	all	0.0078

a) Boolean Ranking Results

runid	all	TF
num_q	all	63
num_ret	all	3150
num_rel	all	3205
num_rel_ret	all	193
map	all	0.0211
gm_map	all	0.0011
Rprec	all	0.0575
bpref	all	0.0791
recip_rank	all	0.2006
iprec_at_recall_0.00	all	0.2145
iprec_at_recall_0.10	all	0.0750
iprec_at_recall_0.20	all	0.0354
iprec_at_recall_0.30	all	0.0234
iprec_at_recall_0.40	all	0.0020
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	0.0889
P_10	all	0.0841
P_15	all	0.0794
P_20	all	0.0714
P_30	all	0.0677
P_100	all	0.0306
P_200	all	0.0153
P_500	all	0.0061
P_1000	all	0.0031

b) TF Ranking Results

runid	all	TF-IDF
num_q	all	63
num_ret	all	3150
num_rel	all	3205
num_rel_ret	all	193
map	all	0.0211
gm_map	all	0.0011
Rprec	all	0.0575
bpref	all	0.0791
recip_rank	all	0.2006
iprec_at_recall_0.00	all	0.2145
iprec_at_recall_0.10	all	0.0750
iprec_at_recall_0.20	all	0.0354
iprec_at_recall_0.30	all	0.0234
iprec_at_recall_0.40	all	0.0020
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	0.0889
P_10	all	0.0841
P_15	all	0.0794
P_20	all	0.0714
P_30	all	0.0677
P_100	all	0.0306
P_200	all	0.0153
P_500	all	0.0061
P_1000	all	0.0031

runid	all	Relevance
num_q	all	63
num_ret	all	3150
num_rel	all	3205
num_rel_ret	all	193
map	all	0.0210
gm_map	all	0.0011
Rprec	all	0.0582
bpref	all	0.0791
recip_rank	all	0.2114
iprec_at_recall_0.00	all	0.2228
iprec_at_recall_0.10	all	0.0706
iprec_at_recall_0.20	all	0.0321
iprec_at_recall_0.30	all	0.0201
iprec_at_recall_0.40	all	0.0020
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	0.0857
P_10	all	0.0825
P_15	all	0.0772
P_20	all	0.0722
P_30	all	0.0672
P_100	all	0.0306
P_200	all	0.0153
P_500	all	0.0061
P_1000	all	0.0031

runid	all	Custom
num_q	all	63
num_ret	all	3150
num_rel	all	3205
num_rel_ret	all	491
map	all	0.0734
gm_map	all	0.0121
Rprec	all	0.1353
bpref	all	0.1652
recip_rank	all	0.5133
iprec_at_recall_0.00	all	0.5471
iprec_at_recall_0.10	all	0.2647
iprec_at_recall_0.20	all	0.1566
iprec_at_recall_0.30	all	0.0476
iprec_at_recall_0.40	all	0.0387
iprec_at_recall_0.50	all	0.0216
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	0.2825
P_10	all	0.2540
P_15	all	0.2370
P_20	all	0.2143
P_30	all	0.1847
P_100	all	0.0779
P_200	all	0.0390
P_500	all	0.0156
P_1000	all	0.0078

c) TF-IDF Ranking Results d) Custom Relevance Ranking Results e) Custom Combination Results

Figure 1: Results of the four ranking algorithms

Discussion:

The data cleaning and query parsing modules demonstrate their effectiveness in extracting and preparing the data for further analysis. The cleaning process itself is efficient, taking approximately 1 minute and 37 seconds, while an additional 35 seconds are dedicated to generating the term indices and inverted document indices. These modules contribute to the subsequent analysis's overall data quality and accuracy.

The Boolean ranking algorithm and the combination ranking algorithm performed comparatively better among the five implemented algorithms. One possible reason for this outcome is that the algorithms were implemented from scratch without utilizing standard modules such as Lucene or Whoosh to obtain TF and TF-IDF scores. Consequently, the custom relevance ranking algorithm's overall score was impacted. With an optimized implementation, the TF-IDF and custom algorithms would likely achieve higher scores in ideal scenarios.

In conclusion, while the Boolean ranking algorithm and the custom combination ranking algorithm showcased superior performance, it is important to acknowledge the potential for improved results from the TF-IDF and custom algorithms under optimized implementations.

Learnings:

This assignment helped me gain a thorough understanding of the data cleaning and parsing techniques that are essential for a Natural Language Processing task. While I knew how Boolean, TF and TF-IDF algorithms worked theoretically, implementing them in code and executing and analyzing their results gave me a better understanding.