IRE Assignment 1

Tasks

1. The code for task 1 uses 4 functions to read the raw data, clean it by removing non-alphabetic characters and applying tokenization (for 15 documents). This is the preprocessing step.
2. The stemmer function in task 2 generates the stems for each document (stemmed_text) and the overall frequency of each stem across all documents (stemmed_dict). This helps in the later steps, like generating the word cloud or for tf-idf. The get_top_stems function returns the most popular stems across all documents. This is also used in the word cloud. Here, only 15 documents are used, which can be randomly selected or sequentially selected.
3. In task 3, the preprocessing step is performed again, but for all files. Then, the term frequency dictionary is generated and converted to a dataframe (tf_matrix). The code for doing this is similar to the stemmer function, except there is the option to not stem the words as well. The idf dictionary is calculated separately, since idf is document-independent and remains fixed for a word/term. The idf dict and tf_matrix are used to get the tf-idf matrix. Next, the get_top_stems_per_doc function extracts the columns (documents) of the tf or tf-idf matrix and returns the top p stems for each document (column) as a list of series and as a dataframe. The function get_avg_stem returns the row-wise (term-wise) average across the dataframe (tf_matrix or tfidf_matrix). The top p stems per document as a list and dataframe are used to compare with keywords and compare the top stems across all documents. They are also used in task 4.
4. In task 4, the top stems from task 3 are used to form the boolean and vector models based on the top p stems. The next few functions are used to generate and vectorize a query using the same preprocessing steps as before. The retrieval function is the main function that computes the dot product between the query vector and the document vector to find the most likely matching documents for a given query. The query is randomly generated from the raw text of a subset of documents in the corpus. This subset is used to evaluate the performance of the model (based on how well the model ranks the relevant documents). The test_model function is the driver code for evaluating a model, given a query and a model (boolean or vector). Finally, k queries are given to the models and the ranking information is stored in a dictionary for future comparison (or as a benchmark).
5. Task 5 involves redoing task 2 but with all documents and after removing stop words. Stop words are removed before stemming.
6. Task 6 involves redoing tasks 3 and 4, but now without stop words. The new result of the k queries is stored in a separate dictionary and compared with the

old dictionary from task 4. The model_comparison function plots the differences for each query. Overall, the tf-idf models seem to be performing better than the tf models. The graphs also indicate that the presence of stopwords seem to decrease the performance by increasing the overall rank of relevant documents. Also, the boolean models seem to be performing similarly (with tfidf being slightly better than tf) regardless of the presence of stop words, while there is a marked difference between the vector models of tf and tf-idf.

7. In task 7, I have used sklearn Kmeans clustering with PCA to cluster similar documents based on the tf-idf matrix. From the clustering output, it is evident of a general association among most documents, since the documents appear to contain content on a common topic (or topics) based on the first 2 principal components.

Practical Requirements

I have included the required functions in the same jupyter notebook. The createTermDocumentMatrix function combines the preprocessing step and the tf_matrix, tfidf_matrix generation steps into a single function. The function for query vector representation uses tfidf (tf of term in query * idf of term) to vectorize the query. I have combined compQueryBoolean and compQueryVector functions into the compQuery function, where either the boolean representation or vector representation of the query can be passed as a parameter. Since the retrieval function is already computing the similarity measure (dot product) of the query with each document and retrieving the ranked list, the retrieval function acts as the similarityMeasure function.