Daniel Choo
CMSC476 – Phase 2
03/11/20

For Phase Two of our Information Retrieval project, I have created some amendments to the preprocessing portion of the project and implemented the term weighting as instructed. Changes to the preprocessing includes implementing two new functions to accommodate lambda filter functions and removing words that only appeared once in the whole tokenized corpus. Then, I implemented the Term Frequency-Inverse Document Frequency portion of the project via three separate functions: TF, IDF, and TFIDF. We will now examine the changes and additions made in Phase Two of the project.

Regarding the preprocessing changes of the program, it was not just a stylistic change, but also a necessary change. I encountered some difficulty while manipulating with the *split*( ) function as it would not save the changes that I would make to the array e.g., removing stopwords and removing single or repeating characters. Thus, I utilized the *filter*( ) function with a lambda function to create a deep copy of the array and keep manipulating the array in that fashion. I also elected to store all of the stopwords in memory as it was too time consuming to reload in all of the stopwords into memory every time we had to access the preprocessor—in the case of my program, approximately nine times. This probably is not a significant difference in runtime, but I thought it was applicable enough to implement.

Following this preprocessing, the TFIDF function will handle the rest of the weighting. The TFIDF algorithm I chose to implement is

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}, \quad idf(w) = \log\left(\frac{N}{df_t}\right), \quad w_{i,j} = tf_{i,j} \cdot idf.$$

I implemented the TFIDF function to call the TF and IDF functions and then calculate all of the weights at the end. The TF function takes in the preprocessed tokenized array, which contains a dictionary of a document's tokens at each index, then creates a new array that stores

another dictionary that stores the term frequency of the word—counter / length of document. The IDF array takes the preprocessed array and takes the log of size of the corpus divided by the frequency of the word in the document. Finally, both of the TF and IDF arrays are returned to TFIDF and computes the weight by multiplying TF and IDF together. A graph is provided below demonstrating the amount of time it took to process *x* number of files. Each of these tests were run independently and were handled by a for-loop for consistency. It appears that the chart is logarithmic.