Edith Flores
Project 3- A Simple Search Engine based on the Vector Space Model (VSM)
Due: November 4th, 2019

Search Engines: Do preprocessing and weight schemes matter?

Abstract:

The Vector Space Model is an important model in order to determine the similarity between documents in order to return relevant documents and return documents similar to a given document. In this project we will introduce the different filters, and weight schemes that can be used in a simple search engine, we will construct a program that will use three parameters, and we will collect and analyse data from the program in order to compare all eight combinations of the parameters and determine which combination is considered the most appropriate for our test collection. The parameters being the use of word stemming or lack of, the use of stop word filters or lack of, and the use of the tf-idf weight scheme or the tf weight scheme. We will compare the different combinations by looking at the Top-K relevant documents and determine the precision and recall for queries, and looking at the K most similar documents for documents.

Introduction:

Before we can dive into the process of creating a simple search engine based on the VSM we must define important terminology. To begin,a search engine may take specific preprocessing steps in order to assure that queries can return a higher number of relevant documents. Preprocessing steps include tokenization, word stemming, and stop word filtering.Tokenization separates a paragraph of words into separate words. Word stemming take a word and return the stem of the word in order to tie it with a larger group or relevant words. For example, “play”,”playing”, and “played” may be all stemmed to “play” so if a document or a query

contains any of these words it will also weigh as much as the other two words. Lastly, stop word filtering is when we remove any possible stop words from the document because they have little to no impact on the meaning of the document. For example, “a” may be considered a stop word since it appears in most sentences but have little to no impact on the meaning of the sentence itself.

In order to actually return relevant document we use a Vector Space Model (VSM), look at each document and query as a vector in a vector space. This way we can figure out the similarity between documents or queries by using the cosine of the angle between them, with 1 being identical documents (same terms throughout) and 0 being no similarity(share no terms). However, to determine the score, or similarity of a document to another document/query, we must look at the term weight of each term in the document/query. The weight of a term means the impact that term has on the given document, which can be determined by looking at the term frequency, tf, a term has on a document (how many times it appears) or the tf-idf, combination between the term frequency and inverse document frequency (how many documents does a term appear in). In either case, a high weight implies a higher impact on the document.

Lastly, the Top-K represents what percentage of relevant documents were returned. Where recall is the percentage of relevant documents did the search engine return, and precision is the percentage of documents returned that are actually relevant.

Goals/Hypothesis:

The goal of our simple search engine based on the VSM is to determine which combination of parameters will return the highest Top-K results, precision and recall. The

parameters being whether the terms are plain or stemmed words, the use of a stop word filter or lack of, and which weight scheme is being used (tf-idf or tf). Additionally, our search engine should also return the K documents that our program believes to be the most similar to the original document.

Based on my knowledge from class, when using the search engine as a query-look-up (search for documents based on a query) it makes sense that the search engine that uses stemmed words, a stop word filter, and uses tf-idf as the weight scheme will return better similarity results and therefore higher Top-K results than any of the other seven combinations. I believe this combination of parameters is the best because the use of stem words will allow us to associate a wider range of relevant documents to their corresponding queries even if words are used in different tenses, the use of a stop word filter will remove irrelevant words that can skew results, and tf-idf determines the weight of the term not only in respect to one words but in respect to the corpus which allows us to determine the weight of the term overall.

However, if we use the search engine to find similar documents to the original document, then the combination does not matter since the document itself should return itself since they share the same words, verbatim.

Steps:

For this research we used Glasgow IR's LISA test collection as the corpus and queries for our simple search engine. The only aspect that was modified was making sure all the relevant documents in the LISA.QUE file were correct and that any missing queries were added properly. Additionally, three randomly selected documents were added as "queries" in order to determine

what K documents are considered “similar” to the original document, and at what position does the original document show up.

In order to make the program resemble as close to a search engine as possible, I implemented user input in order to obtain the desired parameters (Plain or stemmed words, with or without stop word filtering, the query being used, and the three K). This way the user has full control of the simple search engine, similar to the online search engines. I used python-NLTK In order to implement stop word filtering , tokenization, and word stemming (PorterStemmer). After implementing the proper filtering, tokenizing, and word stemming desired by the user, each term was added to a corpus posting list that followed the format:

{Term:[df,{Document:[tf,(Term weight)],...}],...}

In other words, the postings list was a typical postings list but implements the term’s weight depending on which weighting scheme the user selects. Additional information collected by the program was the number of documents, and which terms belonged to which document. The posting list structure and the additional information collected allowed me to easily obtain term and document information for the VSM.

Once all the documents were processed, the user is asked to choose a specific query (from the LISA collection) and three K values for the Top K results (one K value if it is a document from the corpus). In order to obtain document scores, we used the Cosine Score algorithm and stored all the scores in a sorted dictionary by decreasing scores. (Manning, Fig.6.14) The first K documents in the sorted dictionary were compared to the relevant documents of the given query. This way we were able to return the Top- K percentage for

precision and recall for the three K values, or, if the query is a document, return the Top K similar documents and the position of the document itself.

Results

In order to provide the best results while not being too verbose, only 9 queries out of the 35 possible queries were selected at random using a random number generator (Random Lists). The results are split into two tables based on the weighting scheme being used (tf-idf vs tf), and each table has the four combinations of plain or stemmed words, combined with or without stop word filtering. In order to have a wide range of consistent results, the three K values were 30, 300, and 3000 throughout.

Figure 1.1: Query Look-up using Tf-idf

Tf-Idf				
Query	Stemmed Filtered	Stemmed Not filtered	Not Stemmed Filtered	Not stemmed Not filtered
Q31	**Top 30 :** Recall: 0.4 Precision: 0.1333 **Top 300 :** Recall: 1.0 , Precision: 0.0333 **Top 3000 :** Recall: 1.0 , Precision: 0.0033	**Top 30** : Recall: 0.5 Precision: 0.1666 **Top 300** : Recall: 1.0 Precision: 0.0333 **Top 3000** : Recall: 1.0 Precision: 0.0033	**Top 30** : Recall: 0.5 Precision: 0.1666 **Top 300** : Recall: 0.8 Precision: 0.0266 **Top 3000** : Recall: 0.9 Precision: 0.003	**Top 30** : Recall: 0.5 Precision: 0.1666 **Top 300** : Recall: 0.8 Precision: 0.0266 **Top 3000** : Recall: 0.9 Precision: 0.003
Q2	**Top 30 :** Recall: 1.0 Precision: 0.0666 **Top 300 :** Recall: 1.0 Precision: 0.0066	**Top 30** : Recall: 1.0 Precision: 0.0666 **Top 300** : Recall: 1.0 , Precision: 0.0066	**Top 30** : Recall: 0.5 Precision: 0.0333 **Top 300** : Recall: 1.0 Precision: 0.0066	**Top 30** : Recall: 0.5 Precision: 0.0333 **Top 300** : Recall: 1.0 Precision: 0.0066

	Top 3000 : Recall: 1.0 Precision: 0.0006	**Top 3000** : Recall: 1.0 , Precision: 0.0006	**Top 3000** : Recall: 1.0 Precision: 0.0006	**Top 3000** : Recall: 1.0 Precision: 0.0006
Q21	**Top 30** : Recall: 0.0952 Precision: 0.0666 **Top 300** : Recall: 0.4285 Precision: 0.03 **Top 3000** : Recall: 0.7142 Precision: 0.005	**Top 30** : Recall: 0.0952 Precision: 0.0666 **Top 300** : Recall: 0.4761 Precision: 0.0333 **Top 3000** : Recall: 0.7619 Precision: 0.0053	**Top 30** : Recall: 0.0952 Precision: 0.0666 **Top 300** : Recall: 0.4761 Precision: 0.0333 **Top 3000** : Recall: 0.7142 Precision: 0.005	**Top 30** : Recall: 0.0952 Precision: 0.0666 **Top 300** : Recall: 0.4285 Precision: 0.03 **Top 3000** : Recall: 0.7619 Precision: 0.0053
Q23	**Top 30** : Recall: 0.2777 Precision: 0.1666 **Top 300** : Recall: 0.8333 Precision: 0.05 **Top 3000** : Recall: 1.0 Precision: 0.006	**Top 30** : Recall: 0.1111 Precision: 0.0666 **Top 300** : Recall: 0.8333 Precision: 0.05 **Top 3000** : Recall: 1.0 , Precision: 0.006	**Top 30** : Recall: 0.1666 Precision: 0.1 **Top 300** : Recall: 0.8888 Precision: 0.0533 **Top 3000** : Recall: 1.0 Precision: 0.006	**Top 30** : Recall: 0.1666, Precision: 0.1 **Top 300** : Recall: 0.8888 Precision: 0.0533 **Top 3000** : Recall: 1.0 Precision: 0.006
Q7	**Top 30** : Recall: 0.5714 Precision: 0.2666 **Top 300** : Recall: 0.9285 Precision: 0.04333 **Top 3000** : Recall: 1.0 Precision: 0.0046	**Top 30** : Recall: 0.5714 Precision: 0.2666 **Top 300** : Recall: 0.9285 Precision: 0.0433 **Top 3000** : Recall: 1.0 , Precision: 0.0046	**Top 30** : Recall: 0.5714 Precision: 0.2666 **Top 300** : Recall: 0.8571 Precision: 0.04 **Top 3000** : Recall: 1.0 , Precision: 0.0046	**Top 30** : Recall: 0.4285 Precision: 0.2 **Top 300** : Recall: 0.9285 Precision: 0.0433 **Top 3000** : Recall: 1.0 Precision: 0.0046
Q8	**Top 30** : Recall: 0.2307 Precision: 0.2	**Top 30** : Recall: 0.1923 Precision: 0.1666	**Top 30** : Recall: 0.3846 Precision: 0.3333	**Top 30** : Recall: 0.3461 Precision: 0.3

	Top 300 : Recall: 0.8076 Precision: 0.07 **Top 3000** : Recall: 0.9615 Precision: 0.00833	**Top 300** : Recall: 0.8076 Precision: 0.07 **Top 3000** : Recall: 0.9615 Precision: 0.0083	**Top 300** : Recall: 0.8461 Precision: 0.0733 **Top 3000** : Recall: 0.9615 Precision: 0.0083	**Top 300** : Recall: 0.8461 Precision: 0.0733 **Top 3000** : Recall: 0.9615 Precision: 0.0083
Q6	**Top 30** : Recall: 0.5 Precision: 0.3 **Top 300** : Recall: 0.6111 Precision: 0.0366 **Top 3000** : Recall: 0.9444 Precision: 0.0056	**Top 30** : Recall: 0.5 Precision: 0.3 **Top 300** : Recall: 0.6111 Precision: 0.0366 **Top 3000** : Recall: 0.9444 Precision: 0.0056	**Top 30** : Recall: 0.5 Precision: 0.3 **Top 300** : Recall: 0.6111 Precision: 0.0366 **Top 3000** : Recall: 1.0 Precision: 0.006	**Top 30** : Recall: 0.5 Precision: 0.3 **Top 300** : Recall: 0.6111 Precision: 0.0366 **Top 3000** : Recall:0.9444 Precision: 0.0056
Q15	**Top 30** : Recall: 0.4782 Precision: 0.3666 **Top 300** : Recall: 0.9565 Precision: 0.0733 **Top 3000** : Recall: 1.0 , Precision: 0.0076	**Top 30** : Recall: 0.4782 Precision: 0.3666 **Top 300** : Recall: 0.9565 Precision: 0.0733 **Top 3000** : Recall: 1.0 , Precision: 0.0076	**Top 30** : Recall: 0.4347 Precision: 0.3333 **Top 300** : Recall: 0.9565 Precision: 0.0733 **Top 3000** : Recall: 1.0 Precision: 0.0076	**Top 30** : Recall: 0.4782 Precision: 0.3666 **Top 300** : Recall: 0.9130 Precision: 0.07 **Top 3000** : Recall: 1.0 Precision: 0.0076
Q28	**Top 30** : Recall: 0.3636 Precision: 0.1333 **Top 300** : Recall: 0.9090 Precision: 0.0333 **Top 3000** : Recall: 1.0 Precision: 0.0036	**Top 30** : Recall: 0.3636 Precision: 0.1333 **Top 300** : Recall: 0.9090 Precision: 0.0333 **Top 3000** : Recall: 1.0 Precision: 0.0036	**Top 30** : Recall: 0.3636 Precision: 0.1333 **Top 300** : Recall: 0.8181 Precision: 0.03 **Top 3000** : Recall: 0.9090 Precision: 0.0033	**Top 30** : Recall: 0.3636, Precision: 0.1333 **Top 300** : Recall: 0.6363 Precision: 0.0233 **Top 3000** : Recall: 0.9090 Precision: 0.0033

Figure 1.2: Query Look-up using Tf

Tf				
Query	Stemmed Filtered	Stemmed Not filtered	Not Stemmed Filtered	Not stemmed Not filtered
Q31	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.5 Precision: 0.0016	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.6 Precision: 0.002	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.5 Precision: 0.0016	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.6 Precision: 0.002
Q2	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.5 Precision: 0.0033 **Top 3000** : Recall: 1.0 Precision: 0.0006	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.5 Precision: 0.0033 **Top 3000** : Recall: 1.0 Precision: 0.0006	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.5 Precision: 0.0033 **Top 3000** : Recall: 0.5 Precision: 0.0003	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.5 Precision: 0.0033 **Top 3000** : Recall: 0.5 Precision: 0.0003
Q21	**Top 30 :** Recall: 0.0 Precision: 0.0 **Top 300 :** Recall: 0.0 Precision: 0.0 **Top 3000 :** Recall: 0.6190 Precision: 0.0043	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0476 Precision: 0.0033 **Top 3000** : Recall: 0.6190 Precision: 0.0043	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.6190 Precision: 0.0043	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0476 Precision: 0.0033 **Top 3000** : Recall: 0.6190 Precision: 0.0043
Q23	**Top 30** : Recall: 0.0	**Top 30** : Recall: 0.0	**Top 30** : Recall: 0.0	**Top 30** : Recall: 0.0

	Precision: 0.0 **Top 300** : Recall: 0.222 Precision: 0.0133 **Top 3000** : Recall: 1.0 Precision: 0.006	Precision: 0.0 **Top 300** : Recall: 0.0555 Precision: 0.0033 **Top 3000** : Recall: 0.8888 Precision: 0.0053	Precision: 0.0 **Top 300** : Recall: 0.2222 Precision: 0.0133 **Top 3000** : Recall: 0.88888 Precision: 0.0053	Precision: 0.0 **Top 300** : Recall: 0.0555 Precision: 0.0033 **Top 3000** : Recall: 0.8333 Precision: 0.005
Q7	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.8571 Precision: 0.004	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.7142 Precision: 0.0033	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.7857 Precision: 0.0036	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.7142 Precision: 0.0033
Q8	**Top 30** : Recall: 0.0769 Precision: 0.0666 **Top 300** : Recall: 0.1923 Precision: 0.0166 **Top 3000** : Recall: 0.9230 Precision: 0.008	**Top 30** : Recall: 0.0384 Precision: 0.0333 **Top 300** : Recall: 0.2307 Precision: 0.02 **Top 3000** : Recall: 0.8076, Precision: 0.007	**Top 30** : Recall: 0.1538 Precision: 0.1333 **Top 300** : Recall: 0.1923, Precision: 0.0166 **Top 3000** : Recall: 0.9230 Precision: 0.008	**Top 30** : Recall: 0.0769 Precision: 0.0666 **Top 300** : Recall: 0.2692 Precision: 0.0233 **Top 3000** : Recall: 0.8461 Precision: 0.0073
Q6	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.2222 Precision: 0.0133 **Top 3000** : Recall: 0.8888 Precision: 0.0053	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.1111 Precision: 0.0066 **Top 3000** : Recall: 0.7222 Precision: 0.0043	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.2222 Precision: 0.0133 **Top 3000** : Recall: 0.8333 Precision: 0.005	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.1111 Precision: 0.0066 **Top 3000** : Recall: 0.7222 Precision: 0.0043

Q15	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0434 Precision: 0.0033 **Top 3000** : Recall: 0.6086 Precision: 0.0046	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0434 Precision: 0.0033 **Top 3000** : Recall: 0.5217 Precision: 0.004	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.5652 Precision: 0.0043	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.0 Precision: 0.0 **Top 3000** : Recall: 0.5217 Precision: 0.004
Q28	**Top 30** : Recall: 0.0909 Precision: 0.0333 **Top 300** : Recall: 0.1818 Precision: 0.0066 **Top 3000** : Recall: 0.8181 Precision: 0.003	**Top 30** : Recall: 0.0909 Precision: 0.0333 **Top 300** : Recall: 0.2727 Precision: 0.01 **Top 3000** : Recall: 0.7272 Precision: 0.0026	**Top 30** : Recall: 0.0909 Precision: 0.0333 **Top 300** : Recall: 0.1818 Precision: 0.0066 **Top 3000** : Recall: 0.7272 Precision: 0.0026	**Top 30** : Recall: 0.0 Precision: 0.0 **Top 300** : Recall: 0.2727 Precision: 0.01 **Top 3000** : Recall: 0.7272 Precision: 0.0026

In order to provide the best results while not being too verbose, only 3 documents out of the 6004 possible document queries were selected at random using a random number generator (Random Lists). The results are split into two tables based on the weighting scheme being used (tf-idf vs tf), and each table has the four combinations of plain or stemmed words, combined with or without stop word filtering. In order to have consistent results, the K values for documents returned is 5 throughout, and the position of the document itself is provided.

Figure 2.1: Document similarity using tf-idf

tf-idf				
Document #	Stemmed Filtered	Stemmed Not filtered	Not Stemmed Filtered	Not stemmed Not filtered

Document 299	299 851 4871 166 1717 position:1	299 851 4871 1717 166 position:1	299 851 4336 2816 1503 position:1	299 851 4336 1503 4351 position:1
Document 5235	5235 3992 5238 3993 354 position:1	5235 3992 5238 3993 354 position:1	5235 3992 5238 3993 354 position:1	5235 3992 5238 3993 354 position:1
Document 2130	2130 142 3139 5985 2660 position:1	2130 3139 142 5985 2660 position:1	2130 142 4838 800 1622 position:1	2130 142 3139 4838 1622 position:1

Figure 2.2: Document similarity using tf

tf				
Document #	Stemmed Filtered	Stemmed Not filtered	Not Stemmed Filtered	Not stemmed Not filtered
Document 299	166 299 5222 1717 1503 position:2	4534 3437 166 299 4838 position:4	166 299 5222 1503 1717 position : 2	4534 3437 166 4838 1626 position : 6
Document 5235	5235 4866 4324 960	4534 3437 4940 3205	5235 4866 3437 4171	4534 3437 4940 3205

	4171 Position: 1	4324 position : 21	4324 position : 1	5508 position : 19
Document 2130	5318 4102 4533 3437 3139 position:64	4534 3437 4324 1998 3205 position : 201	5318 3437 4533 4324 607 position : 51	4534 3437 1998 4324 3205 position : 172

Analysis

Part 1: Query Look-up

Looking at the query look-up table of the four tf-idf combinations, Figure 1.1, we can see that none of the four combination were explicitly the "better"combination. For example, for Q23 at Top-30 the Stemmed-Filtered Combination had a recall of 0.2777, Stemmed-NoFilter had a recall of 0.1111, and both NoStem-Filter and NoStem-NoFilter both had a recall of 0.1666. Which made me believe that the Stemmed-Filtered combination was the best combination but when looking at the Top-300 the Stemmed-Filtered combination was less than or equal to the other three combinations. Additionally, when looking at two query that had Top-30 recall and precision identical at all four combination the results, the results did not follow a similar pattern. For example on Q28, all four combinations had a recall of 0.3636 at Top-30 but when looking at the Top-300 the Stemmed-Filtered combination had a higher recall (0.9090) than the other three combinations. However, when looking at Q6, all four combinations have the same recall for Top-30 and Top-300 but at Top-3000 NoStem-Filter had the highest recall (1.0) while the other three only had 0.94444. In other words, there was no clear indication on which combination was the best.

Similarly, the query look-up table of the fours tf combinations, Figure 1.2, shows a clearer indication that the Stemmed-Filtered combination was the best combination. For example on Q15, the Stemmed-Filtered combination appears to have the best recalls throughout with a recall of 0.6086 at Top-3000 while the other three combinations; recall were less than or equal to 0.5652. This pattern follows for the most part for the rest of the queries, save several here or there. For example on Q31. Stemmed-Filtered combination had a lower recall at Top-3000 with 0.5 than the Stemmed-NoFilter combination with 0.6. However, cases like these did not occur often.

Looking at both tf-idf and tf tables, it is obvious that using the tf-idf weight scheme returned better results than the tf weight scheme. We can notice this immediately, since the tf-idf rarely had any recall of 0.00 at any of the Top-K while most entries of the tf weight scheme table had a recall of 0.00 at Top-30 and/or Top-3000. Therefore, based on this test collection the tf-idf weight scheme is better than the tf weight scheme.

Part 2: Similar Documents

Looking at the related documents using tf-idf table, Figure 2.1, every test document returned itself as the most similar document. In other words, no matter which combination of parameters were used, using the tf-idf weight scheme returned appropriate results.

Looking at the related document using tf table, Figure 2.2, the majority of the test documents did not return itself as the most similar document, nor appeared within the list of returned documents (5 documents). For Document 299, the Stemmed-Filtered and NoStem-Filtered combination returned the same position, second document.For Document 5235, the Stemmed-Filtered and NoStem-Filtered combination returned the same position, second

document. So we cannot agree which combination was technically better for these two documents.However, for Document 2130 the best combination was the NoStem-Filtered combination. With this being said, we can agree that for document similarity, using tf weight scheme, the NoStem-Filtered combination had the best results.

Looking at both tables, Figure 2.1 and Figure 2.2, it is obvious that using the tf-idf weight scheme was the best approach to finding similar documents because using this scheme the program returned the original document itself as the most similar document.

Conclusion

Part 1:

My hypothesis was not correct. Although I did hypothesized correctly that the tf-idf weight scheme was a better scheme than the tf weight-scheme, the Stemmed-Filtered combination for the tf-idf scheme was not the best. On the other hand, the tf weight scheme had the best results from the Stemmed-Filtered combination as I had hypothesized. In other words, my hypothesis was incorrect in that the combination was not exactly correct, but the components of my hypothesis were correct separately.

After looking at the results, it makes sense that there is no perfect scheme or parameters to get the best results throughout. The queries and documents themselves play a big role in the results. If a document or query is made up of too many stop words, or if the words cannot be stemmed, then they do not get altered much by the filters placed on to them. On the other hand, the weight scheme does impact results drastically since tf-idf considers odd words (lower df) to mean a higher importance in general, while tf considers all words to have equal importance in general. Therefore, it makes sense that the tf-idf weight scheme had better results.

Part 2:

My hypothesis was not correct. The weight scheme and the parameter combination does alter the position in which a document returns itself, when looking for similar documents. I find this concept interesting because I assumed that since two documents are identical then they will return themselves as the most similar document. However, this shows that the way we weight our terms and the way we pre-process could alter the results that we return. Even if they are not obvious in hindsight.

What I learned from this project is that there is no easy algorithm or structure for a search engine to be "the best", and that it takes more than just filtering and comparing to determine whether a document is relevant or similar. It makes sense that different companies like Google and Yahoo try different approaches for their search engines and why their search engines don't always return the same Top-K.

References:

Glasgow IDOM - LISA Collection, http://ir.dcs.gla.ac.uk/resources/test_collections/lisa/.

Manning, Christopher D., et al. *Introduction to Information Retrieval*. Cambridge University Press, 2017.

Miller, Bradley N., and David L. Ranum. *Python Programming in Context*. Jones & Bartlett Learning, 2014.

"Python: Stemming Words with NLTK." *GeeksforGeeks*, 30 Oct. 2018, https://www.geeksforgeeks.org/python-stemming-words-with-nltk/.

"Random Numbers - Give a Range, Get a List of Numbers." *Random Lists*, https://www.randomlists.com/random-numbers.

"Removing Stop Words with NLTK in Python." *GeeksforGeeks*, 23 May 2017, https://www.geeksforgeeks.org/removing-stop-words-nltk-python/.