MSCI 541: HW 5

Jonathan Chen
20722167
December 7, 2021

General

All programs were created using Java (version 17, 2021-09-14 LTS, Java(TM) SE Runtime Environment (build 17+35-LTS-2724), using the IntelliJ IDE.

To run the programs, the pre-built jar files can be used.

- To run the IndexEngine, go to the *out/artifacts/IndexEngine_jar/* directory in the command line and then run *java -jar IndexEngine.jar [path_to_latimes.gz] [output_path] [use of porter stemmer ('True'/'False')]*.
- To run the GetDoc, go to the *out/artifacts/GetDoc_jar/* directory in the command line and then run *java -jar GetDoc.jar [path to indexed data] ['id' or 'docno'] [id or docno value]*.
- To run the BooleanAND program, go to the *out/artifacts/BooleanAND_jar/* directory in the command line and then run *java -jar BooleanAND.jar [path to indexed data] [path to queries file] [output filename]*.
- To run the Evaluate program, go to the *out/artifacts/Evaluate_jar/* directory in the command line and then run *java -jar Evaluate.jar [path to indexed data] [path to qrels file] [path to results file] [output directory] [output csv filename]*.
- To run the BM25 program, go to the *out/artifacts/BM25_jar/* directory in the command line and then run *java -jar BM25.jar [path to indexed data] [path to queries file] [use of porter stemmer ('True'/'False')] [output directory]*.
- To run the Search program, go to the *out/artifacts/Search_jar/* directory in the command line and then run *java -jar Search.jar [path to indexed data]*

Question 1

Given an ***m x n*** matrix with *m* rows of terms and *n* columns of documents, we can determine the words most similar to a given term ***t*** by calculating the covariance matrix, which measures how much two random variables vary together. To do this, the equation for the unnormalized covariance matrix when the data is centred around 0 is $COV = W * W^T$. The resulting matrix is an ***m x m*** square and symmetric matrix showing the vocabulary similarities where the variances of each term are along the diagonal, and the covariances of each term are along the rows and columns. Thus, the terms that are most similar to the given term are the terms in the given term's row or column in the covariance matrix with the highest values.

Question 2

2a) To compute the MLE model of the document collection, the following equation can be used:

$$P(w|c) = \frac{count\ of\ w\ occurrences\ in\ collection\ c\ of\ docs}{total\ word\ occurrences\ in\ collection\ c} = \frac{c(w)}{|c|}$$

2b)
Given:
Jelinek-Mercer's smoothing parameter is set to 0.5 -> $\lambda = 0.5$
Smoothing parameter of Dirichlet prior is set to 100 -> $m = 100$

Dirichlet prior smoothing is a form of Jelinek-Mercer smoothing when:

$$\lambda = 1 - \frac{|D|}{|D|+m}$$

$$\lambda(|D| + m) = |D| + m - |D|$$

$$\lambda|D| = m - \lambda m$$

$$|D| = \frac{m(1-\lambda)}{\lambda}$$

$$|D| = \frac{100(1-0.5)}{0.5}$$

$$|D| = 100$$

Therefore, documents of length 100 will be smoothed exactly the same by Jelinek-Mercer smoothing and Dirichlet prior smoothing.

Question 3
Smoothing short documents more than long documents makes sense because shorter documents are a smaller sample size, so there is a smaller probability that the query words will be found within the document. As a result, the smaller probability will decrease the chances that a relevant document will be found because some of the query words won't be found in the short document. Thus smoothing short documents is important to reduce the emphasis that is placed on a query token being found in the document. On the other hand, reducing smoothing on long documents makes sense as well because the document is a larger sample size, so more confidence can be gained on whether or not the document is relevant given the query tokens, irrespective of prior probabilities.

Question 4
To produce the query-biased snippets, I used the following methodology:

1. Strip the tags from the text sections of the documents
2. Split text on the following punctuation marks: '.', '!', '?'
3. Throw out sentences that are less than 5 words
4. For each sentence:
    a. Let $l = 2$ if it is the first sentence, $l = 1$ if it is the second sentence, $l = 0$ otherwise
    b. Let c = the number of query words that are in the sentence, including repetitions
    c. Let d = the number of distinct query terms that match a word in the sentence

    d. Let k = the longest contiguous run of query terms in the sentence
    e. Calculate the score to be $v = c + d + k + l$
    f. Insert the score and sentence into a max priority queue, sorted by the score for the sentence
5. Use the two highest scored sentences to form the summary