Austin Bailey

Dr. Claudia Pearce

CMSC 476

15 April 2018

Homework 4: Command Line Retrieval Engine

In this homework, the goal was to create a command line search engine that would fetch documents that had the highest cosine similarity scores. The user would enter the executable, along with a series of terms in query, which would be used to determine which documents had the high cosine similarity score. To do this, I implemented an idea into four separate stages that would ultimately print out the top ten documents based on the terms entered.

In the first stage, the idea was to calculate all of the query tf-idf weights for each term. To do this, I created a dictionary called queryDict, which would hold the term and its tf-idf weight regarding the query. In this sense, I was treating the query as a document, and giving each one its own term weight. I would go through and count the frequency of each term in the query, and following this call calcQueryWeight, which would calculate the tf-idf of the terms in the query. It would take in the frequency calculated earlier, the amount of terms total in the query, the amount of documents the term appears in in the corpus (calculated earlier in the program in a dictionary called totalDocDict), the term itself and the amount of documents in the corpus. After the returning value is stored in a variable, it is then stored in queryDict (using the term as a key). It would go through each term and perform these operations, unless it wasn't found in the corpus, which it'll be skipped over.

In the second stage, the idea was to compile all the documents that contains at least one of the terms in the query. To do this, I went back and modified the part where I appended to the dictionary list

(term, iterations, first position), and added a condition. If the term that the loop is currently on matches one of the terms in the query, it'll append those three entries to another list called shortList. It was used to improve efficiency and quickness, so I don't need to go through potentially the entire dictionary file to get the data I need. From there, the program would start at the first position in the postings list for the first term and go through the amount of iterations specified for that particular term (from shortList). Once accessed, the program would append the document and the term weight associated to a list called dictWithSingleTerm. This would be done for every document that contained at least one of the terms. Afterwards, it would enter the documents into another dictionary called dictWithOneTerm. This would be used to sort in ascending order, and it would be used for going through and calculating the cosine similarity in stage 3.

In stage 3, I would take the previous calculated dictionaries and lists to calculate the cosine similarity for each document with at least one term. To do this, I start off by calculating the norm of the query. This will be used in the cosine similarity formula. A loop is then created that goes through each document, which calculates the norm of the current document and the dot product associated between the query and the current document. They're added continuously until the end of the dictWithSingleTerm list. It'll then take the dotProduct and norm of the current document along with the norm of the query to calculate the cosine similarity. It does this for every document in the list and puts it into the dictionary retrievalDict (uses document as a key).

In the final stage, the program takes retrievalDict and sorts it by value. It'll follow that by printing the first 10 documents in the dictionary. If there are less than 10 total documents, then it just prints them ranked by what similarity score they have. In addition, if all the terms entered aren't in the corpus, then the program exits earlier and prints out a message telling the user that the terms entered didn't produce any results.

As for results, it was interesting how some were more significant than others. Entering the query identity theft, the results show that documents 292, 379 and 380 all had a higher cosine similarity score than the rest of the top ten documents. Entering hydrotherapy just gets one document, 273, instead of printing out numerous documents that don't contain any value. By entering asdfuiop cat, the program will tell the user that it couldn't find asdfuiop in corpus, and still displays the documents related to the cat search. This displays a certain level of flexibility that allows users to enter many terms and most likely get results for some of the terms they entered.

Efficiency was improved upon from the last assignment. Instead of the 53.84 seconds from the previous graph, the program hovered around 50 to 51 seconds per run. This is most likely due to the removal of outputting files and formatting. Without those, it decreased a bit per runtime. Otherwise, the amount of time both took were about the same with diffent amounts of files.

Graph on left: HW3, Graph on right: HW4