Kurt Drexel

4/9/2024

Homework 3

In creating my program, I decided to start from the baseline of homework 2, in which we iterated through the entire corpus of documents, removing stop words and tokenizing each word, and calculated the TF\*IDF frequency for each word in the entire corpus. From this baseline I removed the ability for the program to produce the output files of each tokenized word (output\_001.txt – output504.txt) as this part was only necessary for homework 2. My doing homework 3 my first step was to create and open two output files, dictionary.txt and postings.txt nested within each other. The next step I began to do was to figure out how to iterate through my terms and getting the documents each term occurred in. After reviewing my code I was able to determine I had already set up a dictionary that had done this within the code called occurrences set up as Occurrences = {word1 [ doc1 , doc2, doc3] , word2 [doc2 , doc3 , doc4] …} with this occuances dictionary I was able to iterate through each term and document each term is seen in. I did this with the for loop “for term, document in Occurrences.items():” with term representing the tokenized word and document representing the documents associated with each term. Within this for loop I then take the len(document) which will return the total number of documents that term is seen in, useful for the dictionary file. At this point the dictionary write line was as dict\_file.write(f”{term}\n{len(document)}\n ‘line in postings file placeholder’\n”} . Within the same for loop I then create another for loop to iterate through each document each term is seen in. I do this with the for loop “for doc in document” where doc iterates through each document associated with the term. Within this nested for loop I begin to write my postings file as “posting\_file.write(f”{doc},{place holder for weight within file}\n}” . Once I had the basic for both the postings file and dictionary file figured out I began to figure out what to place in the place holder positions for the location of the terms weights within the postings file and the line within the postings file where the term in the dictionary file can be found. To determine the line in the postings file where the current term is found, I figured I could create a counter right after the opening of the files both postings and dictionary that began at 1 (curr\_line =1). I then figured that since the postings file will have each file where the word is found and its weight on their own lines that I could simply add the amount of files the term is found in to my curr\_word to get the new updated position in the postings file where the new term begins. For example with 3 terms if term 1 occurs in 3 files, the location in the postings file where term 1 occurs at will be line 1 (curr\_word =1) after the postings file has been updated with each document term 1 is found in the closest blank line will be line 4. So I add curr\_line + length(document) to give the newest updated position in the postings file where the next words documents and weights can be found. Using this logic I was able to accurately update the current line in postings file where the term is found. At this point the writing line for dictionary file was as dict\_file.write(*f*"{term}\n{num\_docs\_containing\_term}\n{curr\_line}\n"). With curr\_line actually done the dictionary file was effectively done for the time being with the remaining work to be done to finish the postings file. When determining the weights for each term in the corpus per each document I began to search through my code in order to determine if the code needed to be altered or if I would be able to use an already existing line of code to suit my needs. When searching through my tf\*idf calculation I saw that I had kept all my weights within a dictionary that was stored as “Weights[DocNum][token]” with weights being stored as such I would be able to iterate through weights by a specific document then a specific term getting the weight for that term in the specified document. The weights dictionary would be as Weights = {docnum1={token1=2,token2=5}, docnum2={token1=9,token2=4}} lending itself excellently to the program created. Using this weights dictionary I was then able to place each weight for the term in the specified document easily within the postings file using my two for loops created. With the first for loop giving the specified term and the second for loop giving the specified document that term was found in. Being able to be used as such Weights[document 1][ term 1] = weight of term 1 in document 1. With this information I was able to easily finish my postings file with the current line looking as such postings\_file.write(*f*"{doc},{Weights[doc][term]}\n") . With both my postings file and dictionary file writings line being done I was left with a for loop looking as such. The last thing I completed within within my program was the ordering of my dictionary to have it in alphabetical order. To do this I simply used the line Occurances = OrderedDict(sorted(Occurances.items())). This line sorted my Occurrences dictionary in alphabetical order using both the OrderedDict function and the sorted function, sorting all the items (tokenized words) within my Occurrences dictionary.

A screen shot of a computer code

Description automatically generated

When running the code with different amounts of input files going from 10,20,40,80,100,200,300,400,503. I found that with input files of size 10 the program took 4.4 seconds to run to completion. These input files had a size of 127 KB while the output files had a size of 82.5 KB. When running with input files of size 20 the program took about 4.9 seconds to run to completion. While the input files had a size of 194 KB and the output files had a size of 135 KB. When running with input files of size 40 the program took about 5.3 seconds to run to completion, the input files had a size of 392 KB and the output files had a size of 281 KB. When running with input files of size 80 the program took 8.7 seconds to run to completion, with the input files having a size of 731 KB and the output files having a size of 535 KB. When running the program with input files of size 100 the program took 9.1 seconds to run, with the input files having a size of 807 KB and the output files having a size of 604 KB. When running the program with a size of 200 the program took about 12.5 seconds to run to completion, with the input files having a size of 1.46 MB and the output files having a size of 1.09 MB. When running the program with 300 input files the program took about 21.6 seconds to run to completion. With the input files having a size of 2.97 MB and the output files having a size of 2.31 MB. When running the program with 400 input files the program took 55.7 seconds to run to completion, having input files of size 6.33 MB and output files of size 5.32 MB. Finally when running the program with the complete 503 input files the program took 294.2 seconds to run to completion, with input files having a size of 11.1 MB and the output files having a size of 7.21 MB. This program has a time complexity of O(n­­2). With a space complexity of roughly O(n), as the number of files increases the size of the output files will also increase though they will always be smaller than the total size of the input files. With 503 input files the output files are only 63.9% of the total size of the input files. I believe this additional storage space created by the running of the program is a minor adjustment to the total storage space necessary for the storage of the files. As with these two newly created files the user will be able to more easily search through the total files for specific words or phrase finding all files that word or phrase occurs in more easily due to the dictionary file telling the user where to find the information in the postings file and the Postings file telling the user all files the word is found in and the frequency for which it is found in those files.

I have submitted and turned in two separate files for this assignment. One Notebook which can be simply run by pressing the play button and then typing your input files then the output files. As well as a python file which can be run with the command KurtDrexel\_HW3.py “*Input file*”“*Output file*”.