CMSC 671 : Information Retrieval

Phase-III

Guide: Prof. Pearce Anurag Pawar (LA22171)

Goal: The goal of the assignment is to create posting and dictionary files based on the number of mathematical terms.

- **Part I : Creating a global corpus dictionary**
- A HashMap is created to calculate the number of occurrences of a token in all the documents.
- This python dictionary will store the count that represents a number for which a particular token has appeared in how many documents.
- To achieve this, all the previous 500 files are parsed.
- Time taken: *O(n*l)* where 'n' is total no. of files and 'l' is no. lines in each document.

- **Part II : Pre-processing each file according to given conditions**
- All the 500 files are fetched again in one-by-one manner to pre-process the tokens which are from the list of stop words, and which has length as one.
- To check the words from the list of stopwords, I have created a HashMap of stopwords provided on the website. While iterating through the input files, I am checking if the current token is present in stopwords HashMap then I am excluding it. This is achieved in *O(1)* time.
- A local hash is used for each file which stores the value of each pre-processed token.

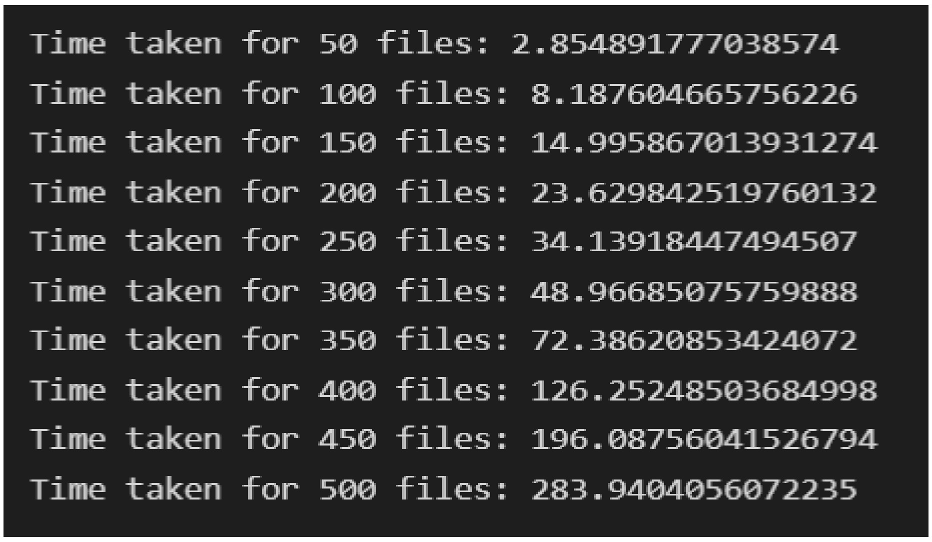
- **Part III – Calculating all the required values and formulae**
- Following terms are calculated with below formulae
- Term frequency = *no. of occurrences token/total no. of tokens in the document*
- Document frequency = *no. of occurrences of a token in all 500 documents*
- Inverse document frequency = *log(no. of documents/ document frequency)*
- Weight of token = *term frequency * inverse document frequency*

- **Part IV – Writing the Posting file**
- To create posting a HashMap of tokens and array of HashMap is created where token is the word and array of HashMap contains the number in which that word occurs and its *tfidf* in that document.
- This HashMap is then passed to a function which writes the files with the number of documents in which that token has occurred and its *tfidf.*

- **Part IV – Writing the Dictionary file**
- HashMap created in the part IV is used again to create the dictionary file.
- This HashMap is passed to another function which writes the token, number of occurrences of that token in all documents and sum of the frequency of previous token and its occurrence.

- **Part VI: Performance analysis**

- **Size of the input/output files:** There are 503 input files which contains documents with tokens and their frequency. These files are compressed into two files, first is posting file and second is dictionary file with the method explained in the part III and part IV respectively.
- A ***memory-based algorithm*** is used as HashMaps are implemented for faster calculations.
- Time taken to process all 500 files: 3.2 seconds.
- Below is the screenshot depicting the time (in milliseconds) required to process no. of files.

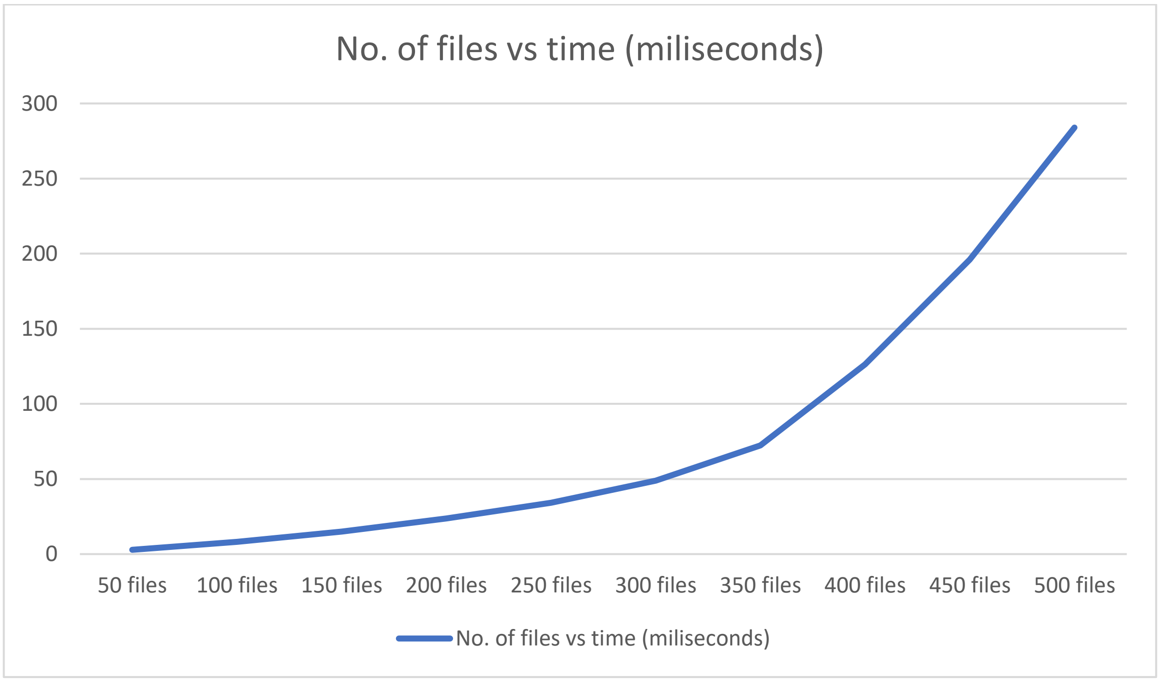


```
Time taken for 50 files: 2.854891777038574
Time taken for 100 files: 8.187604665756226
Time taken for 150 files: 14.995867013931274
Time taken for 200 files: 23.629842519760132
Time taken for 250 files: 34.13918447494507
Time taken for 300 files: 48.96685075759888
Time taken for 350 files: 72.38620853424072
Time taken for 400 files: 126.25248503684998
Time taken for 450 files: 196.08756041526794
Time taken for 500 files: 283.9404056072235
```

-

-

- Below is a graph representing the above data.



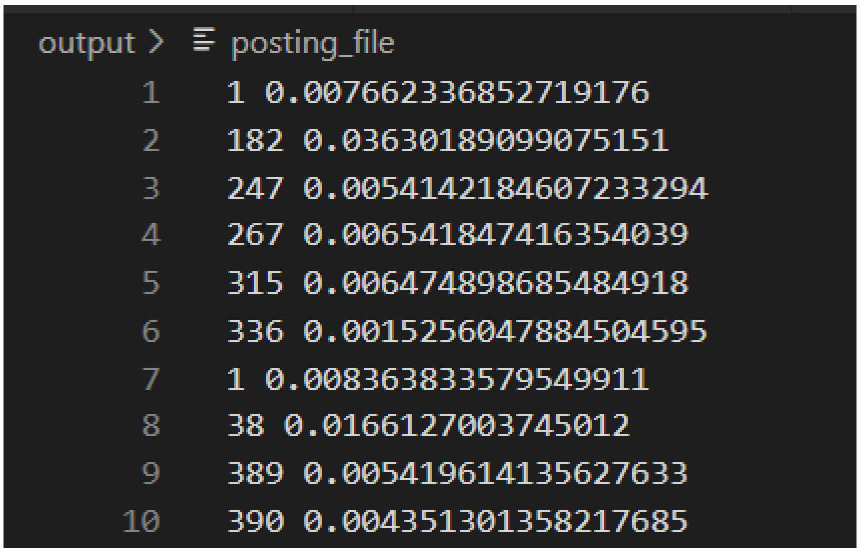
- There are three FOR loops used in the code, complexity analysis as follows:

1. First FOR loop iterates through all the 503 files and reads them i.e., *O(n)* where 'n' is total number of files.
2. Second FOR loop is iterating through each line of the 503 files and pre-processing the tokens i.e., *O(n*l)* where 'l' is no. of lines in each document, in the same loop all the mentioned mathematical values in the part III.
3. Time taken: *O(n*l)* where 'n' is total no. of files and 'l' is no. lines in each document

- Note: Worst case complexity of Python dictionary is *O(n).*

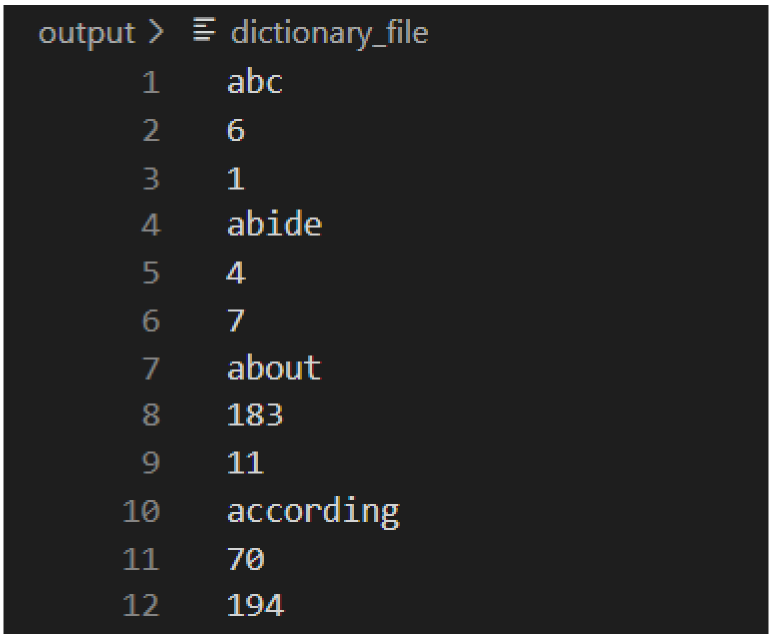
- **Part VII: Output Screenshots**

- Posting file:



```
output > posting_file
1 0.007662336852719176
182 0.03630189099075151
247 0.0054142184607233294
267 0.006541847416354039
315 0.006474898685484918
336 0.0015256047884504595
1 0.008363833579549911
38 0.0166127003745012
389 0.005419614135627633
390 0.004351301358217685
```

- Dictionary file:



```
output > dictionary_file
abc
6
1
abide
4
7
about
183
11
according
70
194
```

- **Steps to execute the program**

- Run program: Python3 <file_name>.py

- **References**

- https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089
- https://towardsdatascience.com/natural-language-processing-feature-engineering-using-tf-idf-e8b9d00e7e76
- https://towardsdatascience.com/text-summarization-using-tf-idf-e64a0644ace3