CMSC 671 : Information Retrieval

Phase-II

Guide: Prof. Pearce Anurag Pawar (LA22171)

Goal: The goal of the assignment is to calculate weight for every token based on the Phase-I output files.

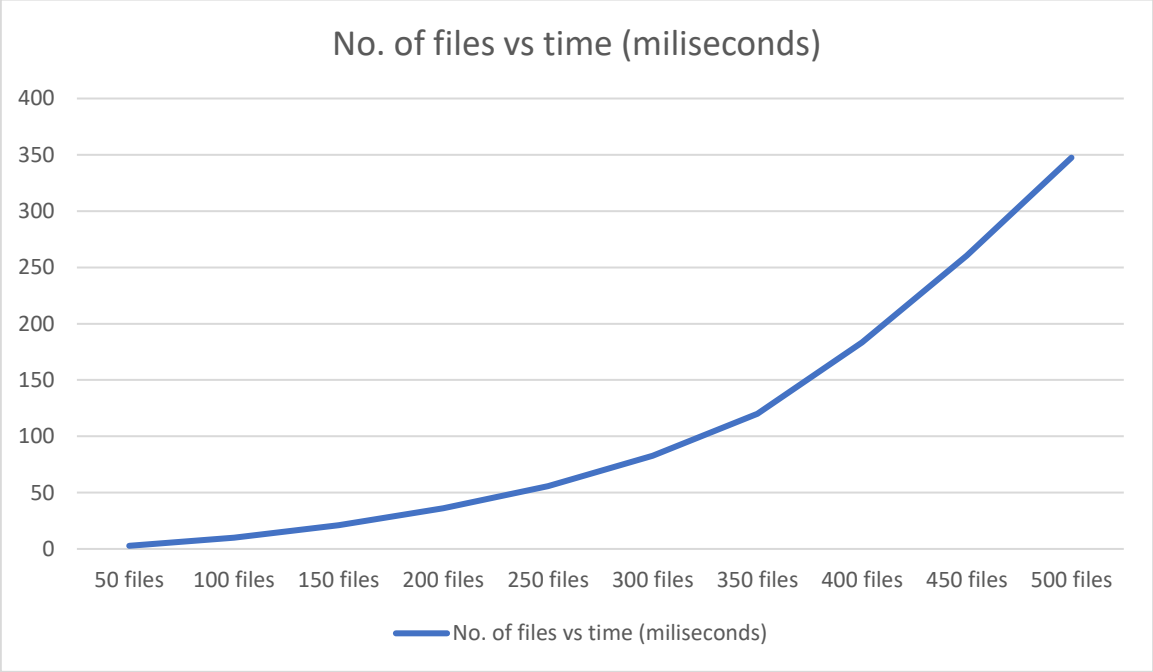
- **Part I : Creating a global corpus dictionary**
- A HashMap is created to calculate the number of occurrences of a token in all the documents.
- This python dictionary will store the count that represents a number for which a particular token has appeared in how many documents.
- To achieve this, 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 output files**
- The token and the calculated weight are written on an output file.
- For each input file, an output file is generated.
- **Part V – Two examples of survived tokens**
- Below are the two examples of two tokens which survived the pre-processing.
- They survived because they are not listed in stop words and their length is greater than one.

1. Example 1
- Token taken from 25th document→ token: arizona || frequency: 6
- Token: arizona || calculated weight: 0.018123507677491758
2. Example 2
- Token taken from 1st document→ token: government || frequency: 10
- Token: government || calculated weight: 0.0023014550897543003

- **Part VI: Performance analysis**
- A ***memory-based algorithm*** is used as HashMaps are implemented for faster calculations.
- Time taken to process all 500 files: 2.6 seconds.
- Below is the screenshot depicting the time (in milliseconds) required to process no. of files.

```
Time taken for 50 files: 2.7631280422210693
Time taken for 100 files: 10.117138862609863
Time taken for 150 files: 21.027690887451172
Time taken for 200 files: 36.13004279136658
Time taken for 250 files: 55.790767192840576
Time taken for 300 files: 82.85896182060242
Time taken for 350 files: 119.96625256538391
Time taken for 400 files: 183.51152205467224
Time taken for 450 files: 260.5251088142395
Time taken for 500 files: 347.48026514053345
```

- 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.
3. Third FOR loop is calculating the values (term frequency, document frequency etc.) mentioned in the earlier part of the document. It is traversing through the local created dictionary for each of the 500 files. This loop is at the level of second FOR.
4. 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).*

- **Steps to execute the program**

- Run program: Python3 <file_name>.

- **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
- https://www.csee.umbc.edu/courses/graduate/676/term%20project/newhw2.html