CMSC 676
Information Retrieval
Project-II

Bhushan Mahajan
m302@umbc.edu

Project overview and brief introduction to TF-IDF:

- This project calculates the term weights for the tokens that occur in each document of given collection.
- These term weights are then stored in a file with extension ".wts" with token.
- The term weights are calculated by the product of term frequency (TF) and inverse document frequency (IDF).
- The weight/score of a word signifies its importance in the document and corpus.

Development Environment:

- OS: MacOS M1 Silicon
- Language: Python 3.8
- Text Editor: VS Code

Steps to execute the program:

- Before executing the program, make sure you have the folder of output files from project-1 since these files are going to be used as input files for the project-2.
- Also, make sure, you have stop-word list file in the project directory.

1. Go to the project directory
2. Run the following command:
    - *python3 calcwts.py <input-files-dir-path> <output-files-dir-path>*

Program Details:

- **High-level functionality:**
    - The program will process a file one at a time from <input-files-dir-path>, pre-processes the data, and calculates weights of each token in a document.
    - Then, the weights are stored in a file along with token. The output is stored in <output-files-dir-path>. If output-files-dir is not present, then the program creates the output-files-dir.
    - The program also calculates execution time of pre-processing and weight calculation operations and generates a graph of execution time vs number of documents.
- **Low-level (Detailed explanation):**
    - Here, I have used memory based stored where, I am storing all the data in hash-table for each file instead of dumping into files.
    - I have also created global hash-table to store count of documents in which a token occurs. This hash-table will be used to calculate document frequency of a token.

1. **Pre-processing**:
   a. The very first step to calculate TF-IDF is to analyze the corpus. As per our problem statement, we have already analyzed the corpus and tokenized the corpus in the Project-1. I had used *Gensim* tokenizer to tokenize the data, and for this project I am using same tokenized data.
   b. Pre-processing is one of the major steps when we deal with any kind of text model. In the Project-1, I had removed all numbers, punctuations, and converted the token in lowercase.
   c. In this project, I am extending the previous preprocessing (used in the Project-1) by removing stop words, tokens that occur only once in entire corpus, and tokens that have length 1.
      i. **Remove stop words**: Since, file of stop-words is already provided, I read the file and stored all the stop-words in a list. For each document, each token is searched in the stop-words list. I have used ***Binary Search*** instead of linear search since binary search has time complexity of ***O(logN)***. If a token is present in the list binary search will return the index of the stop-word and then the token will not be considered for further calculation, if not then -1 is returned. Therefore, I am reducing the time complexity from ***O(M*S) to O(M*logS)*** – where, M = number of tokens in a document; S = Length of stop-word list.
      ii. **Remove tokens that occur only once in entire corpus**: I am simply checking frequency of a token from global hash-table that stores occurrences of tokens in entire corpus. A token will not be considered for further calculation if it's frequency in entire corpus is 1.
      iii. **Remove tokens that have length 1**: I am simply neglecting tokens whose length is 1. Tokens whose length is greater than 1 will be considered for further calculation of TF-IDF.
   d. After pre-processing, all tokens and their frequencies are stored in a local hash-table and passed to a function that calculates TF-IDF of all tokens in the local hash-map.
2. **TF-IDF calculation**:
   a. Terminology:
      i. t: term/token
      ii. d: document (set of tokens)
      iii. N: count of corpus (For this project, N = 503)
      iv. corpus: the total document set
      v. tf: term frequency
      vi. df: count of documents in which token t occurs
      vii. idf: inverse document frequency
      viii. tf-idf(t,d): weights of a token t in document d.
   b. To calculate TF-IDF, I used the formula: ***tf-idf(t,d) = tf(t,d) * idf(t)***,
      Where, ***tf(t,d) = occurrence of t in d / total numbers of tokens in d.***

$$idf(t) = log_e(N/df(t))$$

c. Since we have large corpus, to avoid explosion of *idf* value, I have taken *natural log* of *idf* to dampen the effect.

Demonstration of weight calculation:

- Let's take two examples of from 25th file.
  1. t: "california"
     a. d = "out-25.txt"
     b. N = 503
     c. tf(t,d) = 5/109 = 0.045871559633027525
     d. df(t) = 65
     e. idf = $\log_e$(N/df(t)) = $\log_e$(503/65) = $\log_e$(7.7384615384615385) = 2.046202900204102
     f. tf-idf(t,d) = 0.045871559633027525 * 2.046202900204102 = 0.09386251835798634
  2. t: "oregon"
     a. d = "out-25.txt"
     b. N = 503
     c. tf(t,d) = 2/109 = 0.01834862385321101
     d. df(t) = 34
     e. idf = $\log_e$(N/df(t)) = $\log_e$(503/34) = $\log_e$(14.794117647058824) = 2.6942296454835777
     f. tf-idf(t,d) = 0.01834862385321101 * 2.6942296454835777 = 0.04943540633914822

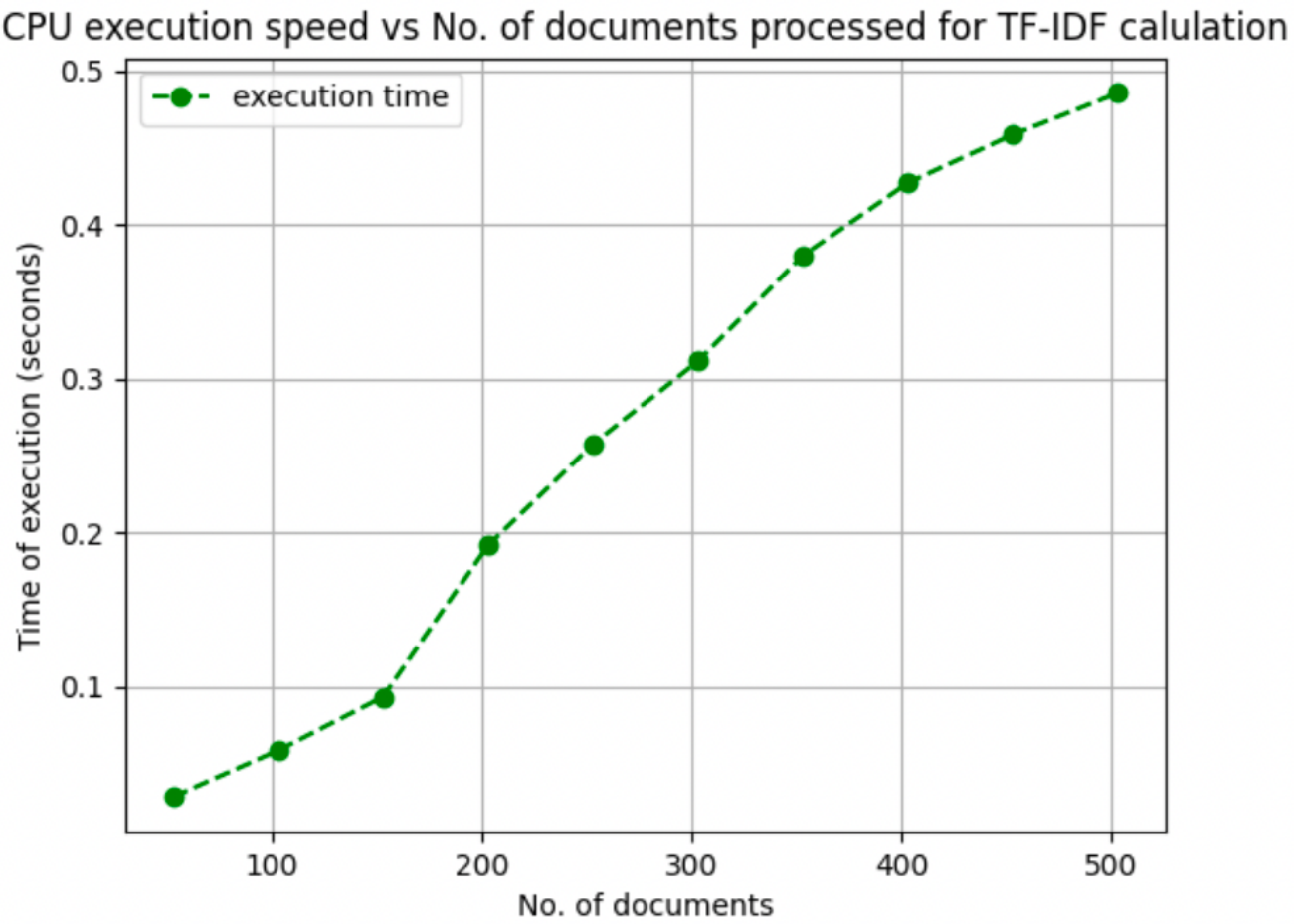
Performance Analysis:

- I have recorded the time taken to pre-process and tf-idf calculation for each document if function. Furthermore, I have generated a graph of execution time vs no. of documents.

Console output:

```
(venv) Bhushans-MacBook-Air:Mahajan_Bhushan_Project_2 bhushan$ python3 calcwts.py files out
CPU time for calculating tf-idf of 53 docs: 0.0290859999999995
CPU time for calculating tf-idf of 103 docs: 0.05944099999999963
CPU time for calculating tf-idf of 153 docs: 0.09392999999999674
CPU time for calculating tf-idf of 203 docs: 0.19327499999999675
CPU time for calculating tf-idf of 253 docs: 0.2622579999999983
CPU time for calculating tf-idf of 303 docs: 0.31436099999999767
CPU time for calculating tf-idf of 353 docs: 0.3831399999999947
CPU time for calculating tf-idf of 403 docs: 0.4306649999999945
CPU time for calculating tf-idf of 453 docs: 0.4620979999999961
CPU time for calculating tf-idf of 503 docs: 0.48976899999999723
(venv) Bhushans-MacBook-Air:Mahajan_Bhushan_Project_2 bhushan$
```

- Graph



References:

[1] https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089