CMSC 476 Information Retrieval: Phase 2 Report

Hafsa Chaudry
Department of Computer Science
and Electrical Engineering
University of Maryland
Baltimore County

***Objective*—This assignment is to update our tokenizer and calculate the term weight in a collection of HTML documents.**

I. Methods

A. Testing Environment

Throughout the experiment, one computer was used to run and test the data in order to ensure the accuracy and consistency of the results. The UMBC GL server was used to execute the algorithms for reasons of convenience and usability. The computer utilized for testing has the following specifications shown in Table 1 below.

Table 1: Testing Computer Specifications

Processor	Intel Core i7 CPU 2.80 GHz
RAM	16 GB
Operating System	Windows 64 Bit

B. Algorithms

All algorithms were developed in Python and run using Python version 3.7.1. I utilized the python libraries of os, regex, time, sklearn, numpy, pandas, nltk, collections, and glob as well as the libraries previously used in phase 1. OS was used to open and read “files” which contained all 503 html files. Os was used to access all the files to be handled by utilizing the use of “walk”. Regex was used similarly in phase 1 to convert the text file and determine and eliminate tokens with special characters, html special characters, and/or numbers along with the use of [a-zA-Z] to return a match for any character alphabetically between a and z, lower case or upper case. I then implemented a simple formula to compare the updated regex tokens with the stop words provided, and eliminated that token if it matched. Then I implemented another algorithm to remove tokens with a length of one or with an occurrence of one. Sklearn, nltk, numpy and pandas was used together to implent my algorithm for tf*idf to get the term weights as well as the implantation of printing it out within a neat and organized data frame. Some useful operations used from these libraries include: CountVectorizer(), .fit_transform(), TfidfTransformer(), np.sarray(),

.sort_values(), nltk.Text(,name=), nltk.TextCollections(), .tf_id(), pd.DataFrame(,index=). In order to correctly find the term weight as well as the term weight per input/output files, I did a lot of online research and modified sections of public tf*idf tutorial source code to be useful for my own code. The code implemented was a small snippet however, and the rest I modified on my own. Glob was used as another way to access my files through a path. Time and threading were used together to show the elapsed time that it took to run my program. The time and threading section of code was found online through stackoverflow, I used this at the end of my report to verify my hypothesis of how more files added increased the time elapsed. To clarify, this part of the code does not make a difference in my overall code but was used as a “helper tool” to verify my findings and I do not claim to have written it. Tqdm was used to show the progression with a progression bar for user efficiency and validity.

The command format was python phase2.py <input directory> <output directory>

C. *Issues*

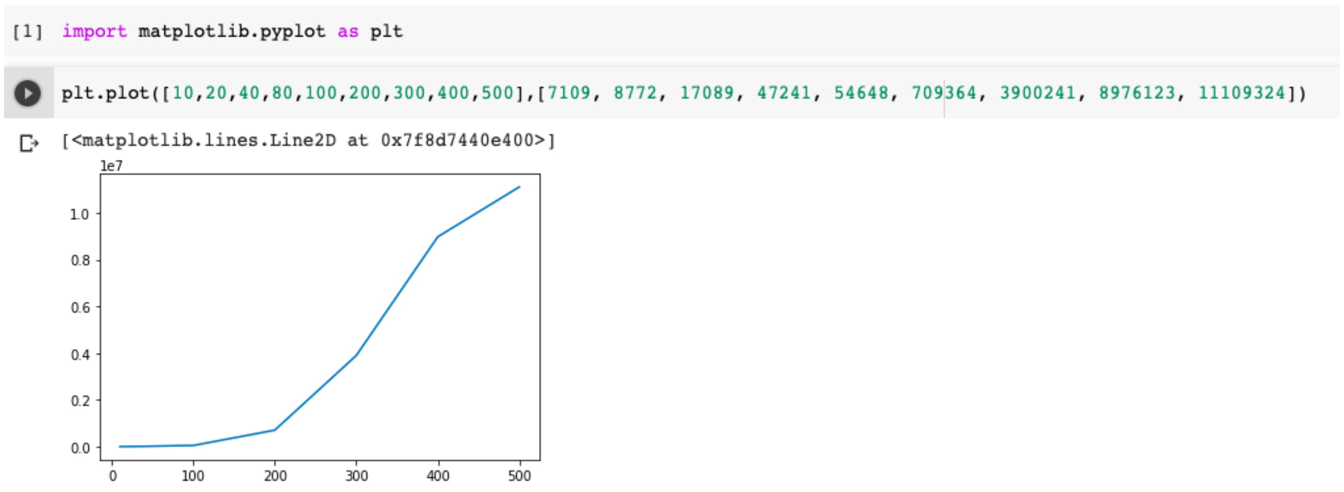
A ran and tested my program in PyCharm, because it is a useful application which makes debugging easier as well as updating files to your path. However, when executing my program, to save space PyCharm automatically compressed both my term weight ouput datas. Both for my first implementation of term weight which tokenized per sentence, and my second implementation of term weight which tokenized per input/output files. Because of this, it is hard to see whether or not my code correctly implemented everything that I wanted it to do. To combat this, I used my own files which were shorter to compare and contrast if each step that I implemented indeed worked. But my files are cleaner than the files given to us but hopefully it will have the same similar outcome. Also, my algorithm works very slow when you increase the number of files to be tested (for input per output). So I suggest you to run a file with >100 files so you do not waste too much time trying to run and test my code in order to grade it. Also, while messing around with the code, I realized that I managed to mess up my BeautifulSoup code that got rid of the html words. I tried using regex to solve this issue as when I tried to implement BeautifulSoup again it was giving me errors and I was running out of time.

II. Results

A. *Input/Output*

After running my code with a select number of input and outputs I created a graph. As you can see, the algorithm created an exponential increase in all running times as n increased. This makes sense as an increase in files being processed will subsequently create an increase in the time it takes to process all the files. A graph showing the running times of each added file is shown below in figure 1.

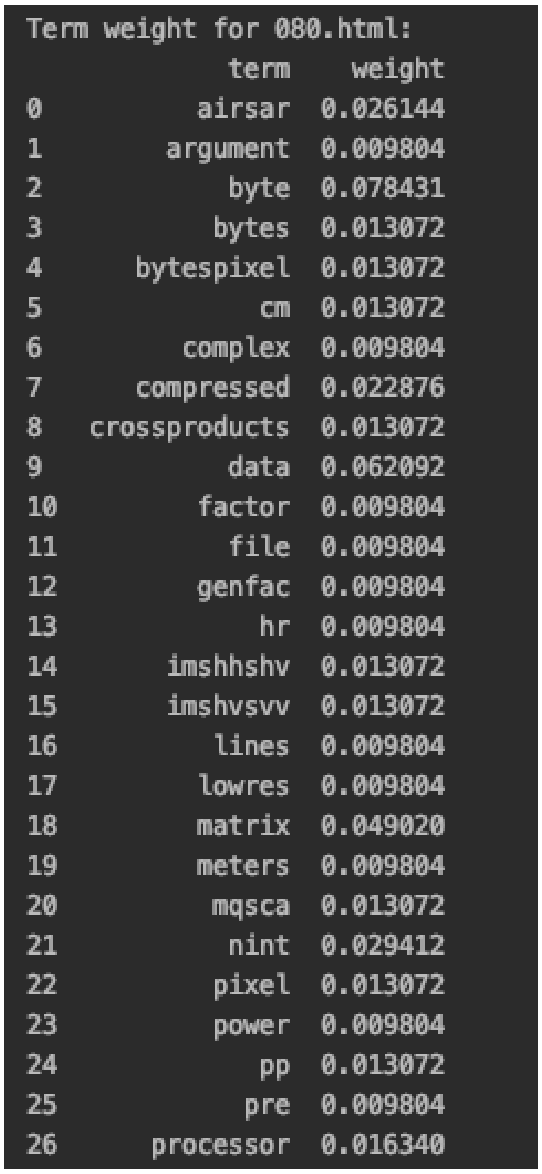
Number of files	10	20	40	80	100	200	300	400	500
Time in millisec.	7,109	8,772	17,089	47,241	54,648	709,364	900,241	8,976,123	11,109,324



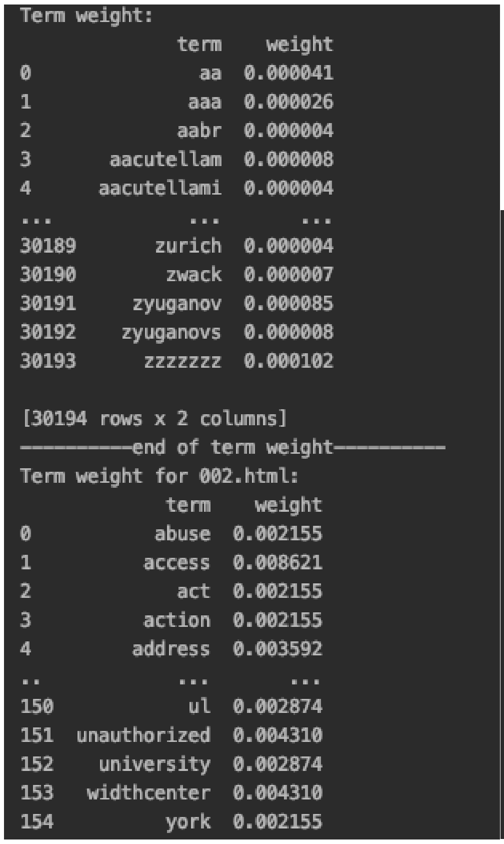
```
[1] import matplotlib.pyplot as plt

plt.plot([10,20,40,80,100,200,300,400,500],[7109, 8772, 17089, 47241, 54648, 709364, 3900241, 8976123, 11109324])

[<matplotlib.lines.Line2D at 0x7f8d7440e400>]
```

Figure 1: Time in milliseconds vs Number files

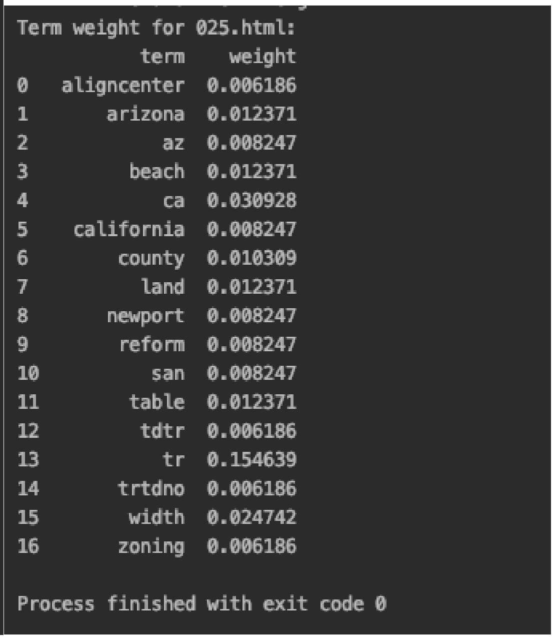
III. "Shell Code" Proof



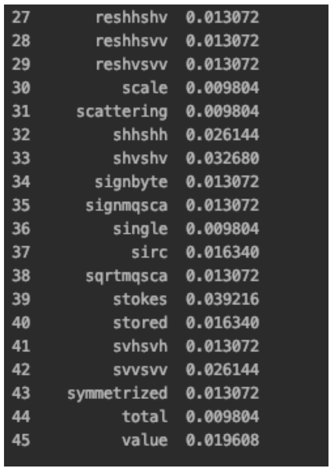
```
Term weight for 080.html:
             term    weight
0          airsar  0.026144
1        argument  0.009804
2            byte  0.078431
3           bytes  0.013072
4      bytespixel  0.013072
5              cm  0.013072
6         complex  0.009804
7      compressed  0.022876
8   crossproducts  0.013072
9            data  0.062092
10         factor  0.009804
11           file  0.009804
12         genfac  0.009804
13             hr  0.009804
14       imshhshv  0.013072
15       imshvsvv  0.013072
16          lines  0.009804
17         lowres  0.009804
18         matrix  0.049020
19         meters  0.009804
20          mqsca  0.013072
21           nint  0.029412
22          pixel  0.013072
23          power  0.009804
24             pp  0.013072
25            pre  0.009804
26      processor  0.016340
```



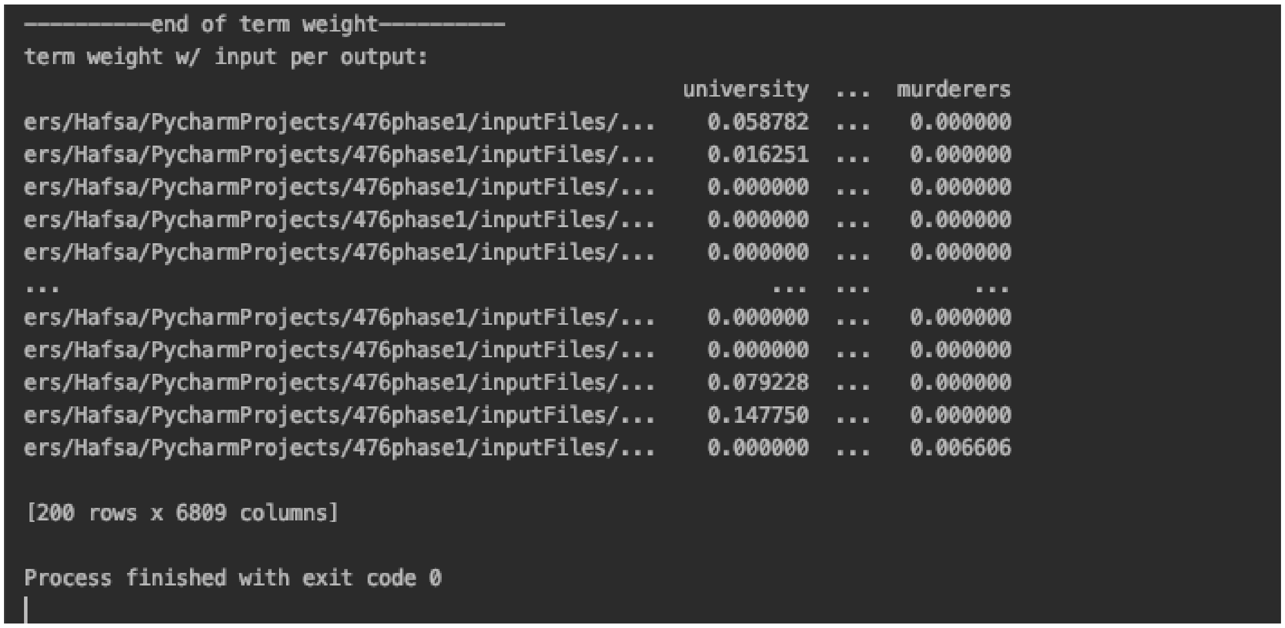
```
Term weight:
              term    weight
0               aa  0.000041
1              aaa  0.000026
2             aabr  0.000004
3       aacutellam  0.000008
4      aacutellami  0.000004
...            ...       ...
30189       zurich  0.000004
30190        zwack  0.000007
30191     zyuganov  0.000085
30192    zyuganovs  0.000008
30193      zzzzzzz  0.000102

[30194 rows x 2 columns]
----------end of term weight----------
Term weight for 002.html:
             term    weight
0           abuse  0.002155
1          access  0.008621
2             act  0.002155
3          action  0.002155
4         address  0.003592
..            ...       ...
150            ul  0.002874
151  unauthorized  0.004310
152    university  0.002874
153   widthcenter  0.004310
154          york  0.002155
```



```
Term weight for 025.html:
           term    weight
0   aligncenter  0.006186
1       arizona  0.012371
2            az  0.008247
3         beach  0.012371
4            ca  0.030928
5    california  0.008247
6        county  0.010309
7          land  0.012371
8       newport  0.008247
9        reform  0.008247
10          san  0.008247
11        table  0.012371
12         tdtr  0.006186
13           tr  0.154639
14       trtdno  0.006186
15        width  0.024742
16       zoning  0.006186

Process finished with exit code 0
```



```
27       reshhshv  0.013072
28       reshhsvv  0.013072
29       reshvsvv  0.013072
30          scale  0.009804
31     scattering  0.009804
32         shhshh  0.026144
33         shvshv  0.032680
34       signbyte  0.013072
35      signmqsca  0.013072
36         single  0.009804
37           sirc  0.016340
38      sqrtmqsca  0.013072
39         stokes  0.039216
40         stored  0.016340
41         svhsvh  0.013072
42         svvsvv  0.026144
43    symmetrized  0.013072
44          total  0.009804
45          value  0.019608
```



```
----------end of term weight----------
term weight w/ input per output:
                                                     university  ...  murderers
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.058782  ...   0.000000
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.016251  ...   0.000000
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.000000  ...   0.000000
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.000000  ...   0.000000
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.000000  ...   0.000000
...                                                         ...  ...        ...
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.000000  ...   0.000000
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.000000  ...   0.000000
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.079228  ...   0.000000
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.147750  ...   0.000000
ers/Hafsa/PycharmProjects/476phase1/inputFiles/...     0.000000  ...   0.006606

[200 rows x 6809 columns]

Process finished with exit code 0
```