CMSC 676: Project Phase 2 Akarsh Kashamshetty

PHASE 2 – REPORT

Preprocessing:
In this phase, I made sure to remove the words given in the stopwords list and I have removed the unique words that found in the documents whose frequency is 1. Also, the words that are of length 1. After this preprocessing most of the tokens looked meaningful and better when compared to the phase 2 Tokenizer.

Approach:

To calculate the term frequency and inverse document frequency I have used the following formulae and normalized the term weights.

- Term frequency = freq(wordi in documentj) / totalfreq(all tokens in documentj)
- Inverse Document frequency = log (# total documents / # of documents in word appeared).

First, I have calculated the term frequencies by maintaining a local dictionary and written the tokens with their term freq into the wts files for each input file. While writing I have calculated the tf using the above formula and written it to the file.

Secondly, to calculate the idf I have used a global dictionary (hash map) and updated the dictionary with key as the word and value as the number of documents the word has appeared. when I parsed each input document for the tf's whenever I see the word, I have incremented its counter once per document as it appeared.

Finally, I have tried reading the wts document earlier I have written with tokens and tfs into a dictionary and updated its value with the tf * idf. And later tried writing the final results to the wts file. However, I have come across decoding error on the utf-8. I figured that the error is due to the way I am writing and reading. I believe datatype conversions led to the errors. I spent a lot of time solving that issue but could not be able to do that.

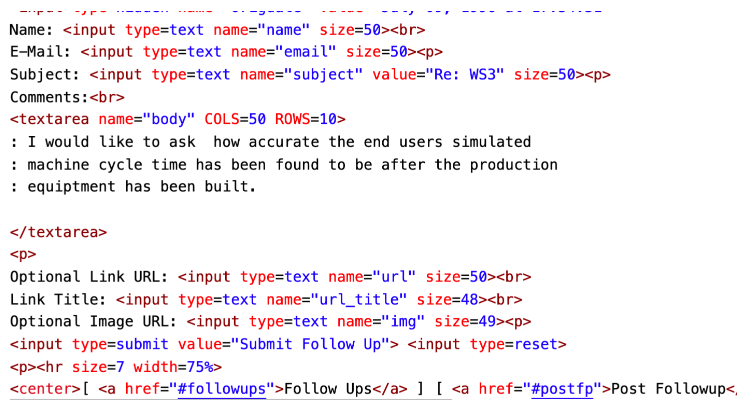
Another Approach:

Eventually, I figured a new way instead of writing several times into the file I ran the same functions twice. Once to calculate the idf's and second time to calculate the tf and final results.
As my tf is using local dictionary and I do not want to increase the time complexity by storing all tf's in global I have chosen this order. This worked out very efficiently and later I have written the final tf*idf into the wts file for each input file.

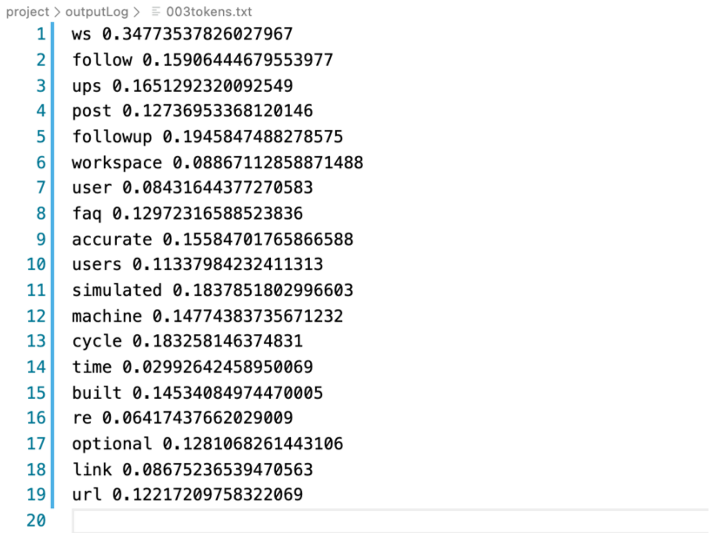
command: python3 main2.py /inputDirectory /outputDirectory

Examples of the input documents with the tokens and weights :
Now my tokens seem to be looking more meaningful after removing several unnecessary words. If we look at the example 003token.wts file, we can see token and their respective weight. We can observe below that all the other words like a, I, been have been filtered out in

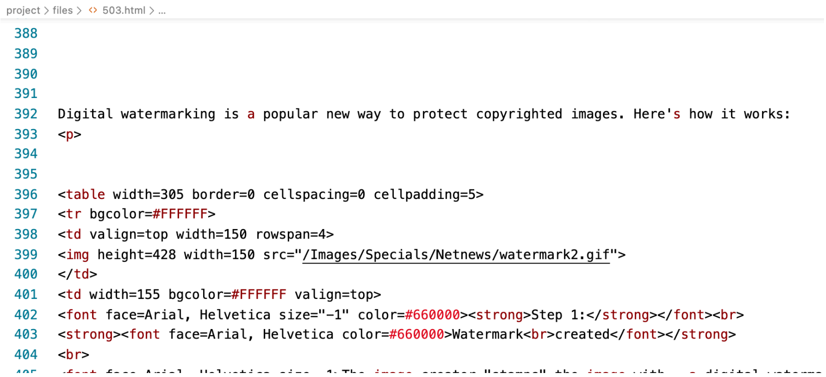
out final token weights documents. Similarly, in the 503.html file we can observe the same thing in our final 503.txt document. The words like “the, is, to ..” have been filtered out.



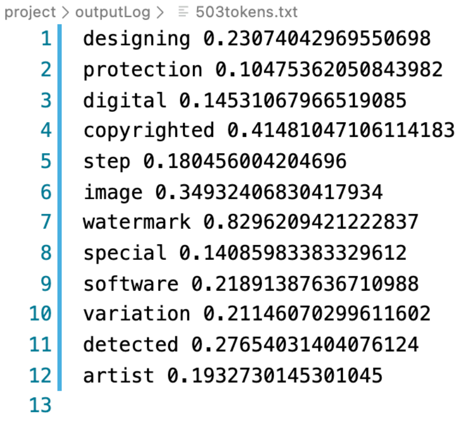
```
Name: <input type=text name="name" size=50><br>
E-Mail: <input type=text name="email" size=50><p>
Subject: <input type=text name="subject" value="Re: WS3" size=50><p>
Comments:<br>
<textarea name="body" COLS=50 ROWS=10>
: I would like to ask  how accurate the end users simulated
: machine cycle time has been found to be after the production
: equiptment has been built.

</textarea>
<p>
Optional Link URL: <input type=text name="url" size=50><br>
Link Title: <input type=text name="url_title" size=48><br>
Optional Image URL: <input type=text name="img" size=49><p>
<input type=submit value="Submit Follow Up"> <input type=reset>
<p><hr size=7 width=75%>
<center>[ <a href="#followups">Follow Ups</a> ] [ <a href="#postfp">Post Followup<
```



```
ws 0.34773537826027967
follow 0.15906444679553977
ups 0.1651292320092549
post 0.12736953368120146
followup 0.1945847488278575
workspace 0.08867112858871488
user 0.08431644377270583
faq 0.12972316588523836
accurate 0.15584701765866588
users 0.11337984232411313
simulated 0.1837851802996603
machine 0.14774383735671232
cycle 0.183258146374831
time 0.02992642458950069
built 0.14534084974470005
re 0.06417437662029009
optional 0.1281068261443106
link 0.08675236539470563
url 0.12217209758322069
```



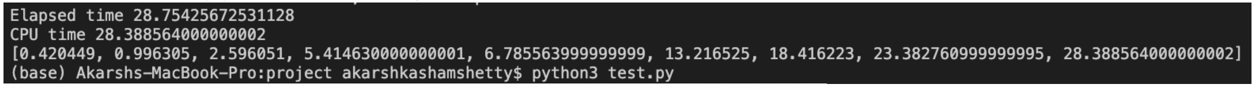
```




Digital watermarking is a popular new way to protect copyrighted images. Here's how it works:
<p>


<table width=305 border=0 cellspacing=0 cellpadding=5>
<tr bgcolor=#FFFFFF>
<td valign=top width=150 rowspan=4>
<img height=428 width=150 src="/Images/Specials/Netnews/watermark2.gif">
</td>
<td width=155 bgcolor=#FFFFFF valign=top>
<font face=Arial, Helvetica size="-1" color=#660000><strong>Step 1:</strong></font><br>
<strong><font face=Arial, Helvetica color=#660000>Watermark<br>created</font></strong>
<br>
```



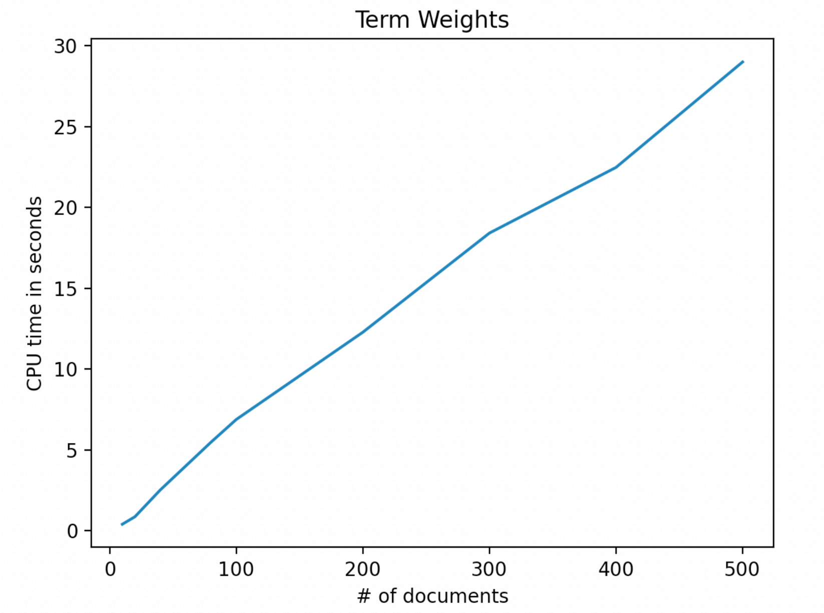
```
designing 0.23074042969550698
protection 0.10475362050843982
digital 0.14531067966519085
copyrighted 0.41481047106114183
step 0.180456004204696
image 0.34932406830417934
watermark 0.8296209421222837
special 0.14085983383329612
software 0.21891387636710988
variation 0.21146070299611602
detected 0.27654031404076124
artist 0.1932730145301045
```

Efficiency of the approach:

Below is the graph of the time complexities against number of documents processed. I have used CPU time for the time. However, we can see both the CPU time and elapsed time in the terminal. This approach is taking maximum of 29 seconds to run for all 503 documents.



```
Elapsed time 28.75425672531128
CPU time 28.388564000000002
[0.420449, 0.996305, 2.596051, 5.414630000000001, 6.785563999999999, 13.216525, 18.416223, 23.382760999999995, 28.388564000000002]
(base) Akarshs-MacBook-Pro:project akarshkashamshetty$ python3 test.py
```



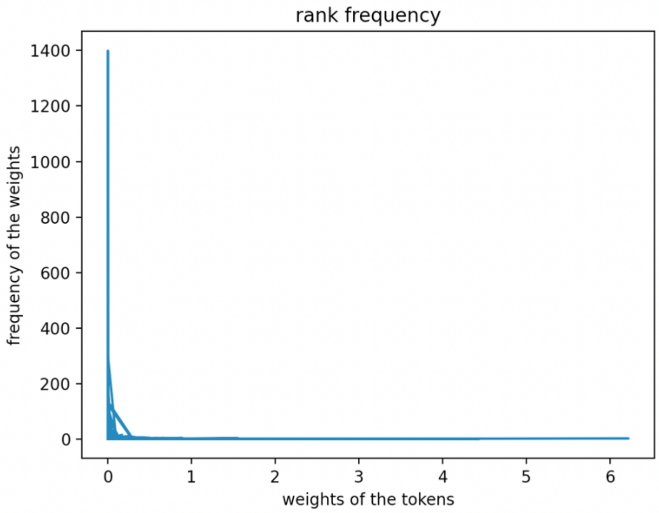
We can observe that as the number of documents increasing the time is also increasing which is directly proportional.

Future Improvements:

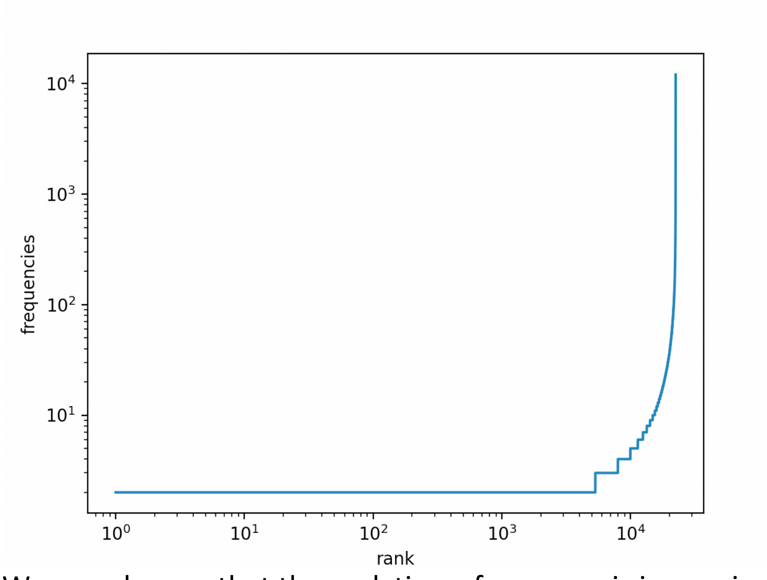
I could improve the approach by using a in memory approach and less function calls, which should improve the efficiency.

Extra Credit 1:

I have worked on calculating the frequencies of the weights and I have dictionary to store the weights of the tokens and their frequencies. And the following is the plot.



The following is the loglog plot of the rank vs frequencies of the documents.



We can observe that the rank times frequency is increasing as the frequency of the token is increasing.