Austin Bailey

Dr. Claudia Pearce

CMSC 476

10 March 2018

Homework 2 Report: Weighting Terms

In response to the task of weighting surviving terms, the parser first had to be extended to exclude stop words, words that appear once in the corpus and words of length one. To do this, I just added a condition when adding terms to dictionaries that excluded the violating terms. These included words that appeared only once in the corpus and words of length 1. To prevent stop words from being included, I created a dictionary of the stop words. This would allow a conditional statement to quickly check if the word was a stop word and thus prevent it from being included.

The next task involved determining how many documents terms appeared in. This was necessary for the tf-idf formula that was going to be used to weight the terms. Going through the files, it would add terms to another dictionary that kept track of each term and its appearances in all the documents. If it appears again in the same document, it wouldn't increment since it checks to see if it has already appeared (if frequency is equal to 1). It would do this for every term in the html files.

This would be the first loop through the files. The second would go through each html file and calculate the terms and their frequencies as normal. After this, it would calculate the term frequency, calculate the inverse document frequency, and then multiply them together. Idf would use the previously discussed dictionary created to determine how many documents a term appears in. As a hash table, it would quickly gather the value and make the calculation. It would output a file with the term and the term weight associated with it for each html file.

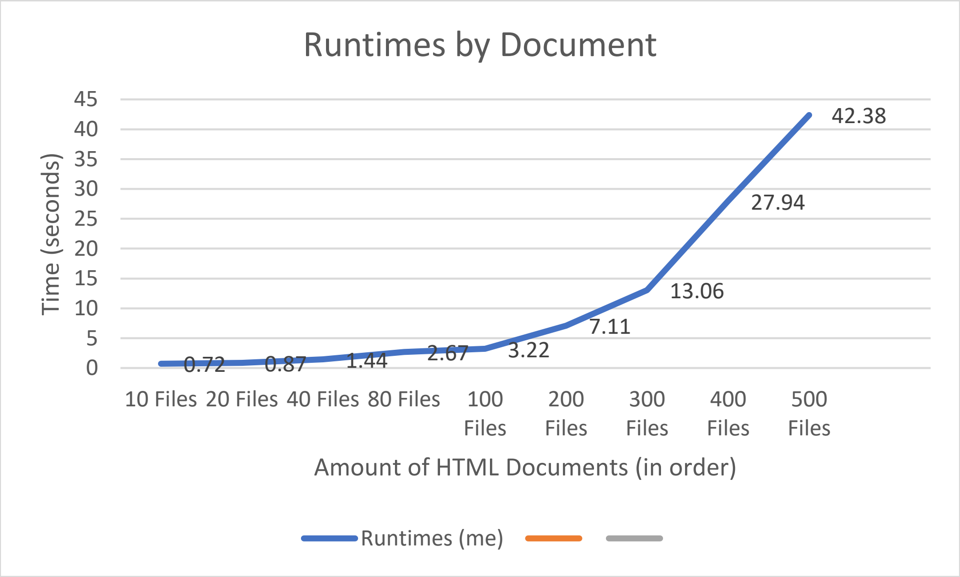
For example, in the file 7tokens.csv the term ‘content’ was associated with a term weight of 0.015. There were also terms ‘caused’, ‘global’, ‘dynamic’ and ‘results’ that contained the weights 0.003, 0.012, 0.004 and 0.015 respectively. The file 25tokens.csv (based on 025.html) had the terms ‘firm’, ‘relations’ and ‘restrictions’ which had the weights 0.022, 0.015 and 0.026 respectively. Many of these weights are low because they are based on using all 503 html files. The larger number of files, the more likely that most terms are going to have smaller weights.

In general, my approach was efficient. I used a hash table (dictionary) to keep track of how many documents each term appeared in so I could use it later on to calculate the inverse document frequency. In fact, most of the tokens and their frequencies/weights were kept inside dictionaries for ease of access. It helped the runtimes become more reasonable, despite needing to check for appearances in documents. Otherwise, it was the general same structure as the previous preprocessor. Some updates were made to keep out certain tokens, but those didn’t really have any impact on runtimes. It was the idf that made the approach more difficult.

Despite this approach, my efficiency could’ve been better. The runtimes got slightly longer than expected with this approach. Perhaps using deprecated documents for storing frequencies on the disk would’ve been a faster approach. Overall, the efficiency was good. Not great, but good enough for a reasonable runtime.

The formula that I used involved calculating tf-idf term weights with term frequency multiplied by inverse document frequency. For tf, I divided the number of times the token appeared in the current document (frequency) with the total number of tokens in the whole document (after ridding of all the violating tokens such as stop words). I multiplied that by the log of idf. Idf was the total number of documents in the collection divided by the number of documents the token appeared in. This result gave the term weight, which I stored in a dictionary along with its token. This was done for every token in the collection.

The preprocessor would then print out each document containing the tokens and their term weights for each html file. After that, the program would close all the files and it would shut down. The reason why I chose the formula that I did was because it was relatively easy to implement and had the least amount of complications. It provided the term weights that were sound and run times that were quick enough for large amounts of documents.


Runtimes by Document
Time (seconds)
10 Files
20 Files
40 Files
80 Files
100 Files
200 Files
300 Files
400 Files
500 Files
Amount of HTML Documents (in order)
Runtimes (me)