Austin Bailey

Dr. Claudia Pearce

CMSC 476

19 February 2018

**Homework 1 Report: Tokenize HTML Files**

My idea to parse and tokenize the HTML documents involved an algorithm that heavily involved dictionaries. Using the os module, my program goes through and reads how many html files are in a directory, and then creates a list of them to use in the program. From there, it would go through each html file in a loop, create a tokenized document, and add it to a dictionary. This dictionary would be outputted to its own file with the word and frequency of said word being paired with it. I also used a dictionary to hold all words and the frequencies of those words. This would be used to create a file with the tokens and frequencies sorted by tokens and frequencies, respectively.

My program used the isalnum() method to handle punctuation and numbers. Each document would be converted to text using get\_text() (bs4) and then a for loop was used to go through and get rid of the punctuation and other symbols. I would follow that by splitting the document, creating a list of the tokenized html file.

I would then use 2 dictionaries for frequencies: one for the specific document and one for all the words in every document. Using a for loop, my program would go through and check for previous words in each dictionary. If it wasn’t inside, the frequency was given a 1, and if it was it was incremented by 1. This was done for both dictionaries.

Comparing my results to Jack Allen, I found that he had the better tokens. Some of my tokens ended up being concatenated, while most of his were not. I also found that he had some tokens that I couldn’t read at all. Comparing efficiency, however, my program had the edge. According to the table, we generally had the same times for 1, 10 and 100 documents. When increased to 500, the differences show with a whole 6 seconds of difference in runtime. My program having slightly less tokens might attribute to this disparity, however.

As for why some of my tokens didn’t properly tokenize, it might just be because I didn’t account for all circumstances surrounding the possible combinations of characters. One example being ‘mid\u0151n’. The \ shouldn’t be in the token, yet it still is. This leads me to believe that the simplicity of just going through a text version of the html document and then running a for loop to eliminate characters such as punctuation failed to account for objects like tables. ‘u00f6z\u00e9pen’ is another example of this type of situation.

Overall, compared to Jack my program might be faster but his has the better handling of certain characters and situational handling. The running time is a little slow, but it does a good job of dealing with the text in the html files. I feel that my method of creating a list of html files is noteworthy in this regard though because it makes the tokenizer more flexible to directories. Along with the more efficient results, my program and his feel overall equal in quality.