CMSC 476 Information Retrieval: Phase 1 Report

Hafsa Chaudry

Department of Computer Science

and Electrical Engineering

University of Maryland

Baltimore County

***Objective*—This assignment is to compare two approaches to tokenize and down case all words in a collection of HTML documents.**

1. INTRODUCTION

The purpose of a tokenizer is to break the text of a file into “tokens”, usually by white case or by characters. I used versions of two publicly available python tokenizer. I had to change the tokenizer to allow the directory of files to be accessible for the program as well as to loop through a whole file to create a token for each word instead of to create tokens for specific words, to do so, I merged parts of one tokenizer into the parts of another tokenizer to create one whole updated tokenizer. The steps that were taken in order to achieve the proper objectives of the project include opening the file to read, convert html files into text files, update the text to the same case and remove special characters and numbers, read and put tokens into a dictionary, count the frequency of each token, and finally, create two documents to hold the frequency from most to least as well as the tokens in alphabetical order. I then created two graphs to show the progression of the performances of creating and counting tokens as well as a graph showing the time it took to view through all the files.

# II. METHODS

1. *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 |

1. *Algorithms*

All algorithms were developed in Python and run using Python version 3.7.1. I utilized the python libraries of BeautifulSoup4, regex, sys, os, time, threading, and tqdm. BeautifulSoup4 was used to parse through the html documents. Regex was used to convert the text file into a dictionary by utilizing the use of re.findall() and was also used to determine and eliminate tokens with 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. Sys was used to pass the input and output documents as a parameters into an array by utilizing sys.argv[]. Os was used to access all the files to be handled by utilizing the use of “walk”. 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 phase1.py <input directory> <output directory>. A hash map was used to count the frequencies of the tokens. Other useful operands include .split() and .lower() to update and process the texts as uniform tokens.

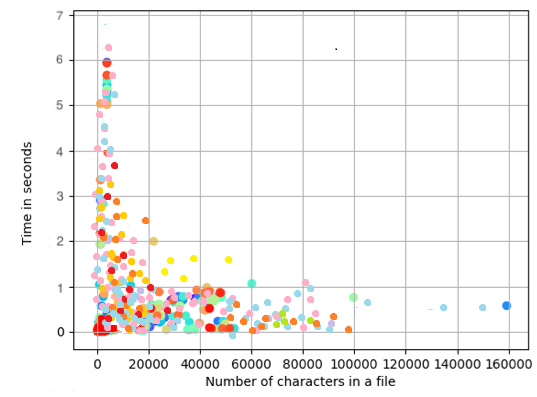
1. *Issues*

The use of regex was to identify eliminate any special characters, numbers, and/or punctuations. In doing so, it updated the tokens without these special chars, and therefor changed the actual token’s meaning or use. For example, if the token originally was a username with special characters then by deleting those special characters also changes the identification of that token. Other example of how an HTML construct or word is incorrectly tokenized is with the examples of passwords, email addresses, addresses, or possessions.

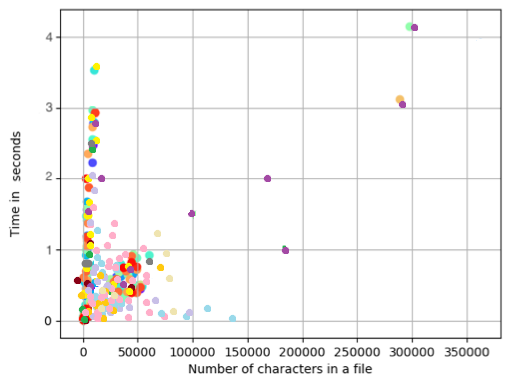
# III. RESULTS

1. *Token Frequency Graphs*

After comparing my approach to the approach of group members Arish Marshall, Aditi Choksi, and Gabriel Nieves-Ponce who found a similar publicly available tokenizer it was observed that their tokenizer performed better than my own. A reason for this could be how their original tokenizer was completed while mines needed updating by merging parts of two different tokenizers. During the merging of two tokenizers, I might have not correctly incorporated the necessary steps in order to achieve the desired outcome. A graph showing my efficiency is shown below in figure 1 with the other groups efficiency shown in figure 2.

**

*Figure 1: Time in seconds vs Number of characters in a file*



*Figure 2: Time in seconds vs Number of characters in a file*

1. *Time Graph*

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 2.

A close up of a map

Description automatically generated

*Figure 3: Time in seconds vs Number of files*

# V. “Shell Code” Proof

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a cell phone

Description automatically generated