

CMSC 676

Information Retrieval

Homework-1
Bhushan Mahajan
m302@umbc.edu

Summary:

- The objective of this assignment as stated in the given problem statement is to compare two approaches to tokenize and downcase all words in a collection of 503 HTML documents.
- Approach to tokenize the HTML documents and count frequency of tokens:
 - For all documents in input folder: Parse the HTML document to remove HTML tags and get the text data within HTML tags. (Here, html2text parser is used. More information is provided below)
 - Tokenize the text data using a tokenizer. (Here the Gensim tokenizer is used. More information about the Gensim is provided below). This will give list of tokens from the text data.
 - After getting token list use regular expression to remove unnecessary symbols, spaces, etc. (This is a bottleneck of the Gensim)
 - After RegEX, add the token into hashmap. Here 2 hashmaps are used. One for generating for solution 1.1 and other for solution 1.2 and 1.3. Hashmap is used to store the frequency of the token in the HTML document. Therefore, one hashmap will keep count of tokens per HTML document and another will keep count of tokens throughout all the HTML documents. (We will use 2nd hashmap to sort tokens alphabetically and by frequency)
 - Increment the count of token to get frequency of the token. Default frequency of a token is 0. (**Frequency Counting mechanism**)
 - Store the token and its frequency in text file.
 - After processing all the HTML files, sort the 2nd HashMap alphabetically as well as by frequency and store the results into 2 separate text files.
 - While tokenizing the text data, record the CPU execution time and after processing all the documents store execution time and number of documents processed in CSV file.
 - Draw a graph of execution time vs number of documents processed.

Steps to execute the program:

- Install required libraries:
 - python3 -m pip install -r requirement.txt
 - Make sure pip is upgraded to latest version.

- Run program:
 - Python3 <file_name> <input_files_dir> <output_files_dir>
- For solution 1.1: all 503 output file will get stored in <output_files_dir>
- For solution 1.2 and 1.3: Please check files:- “output-sol-2.txt” and “output-sol-3.txt”.

Regular Expressions (To solve error in tokenization):

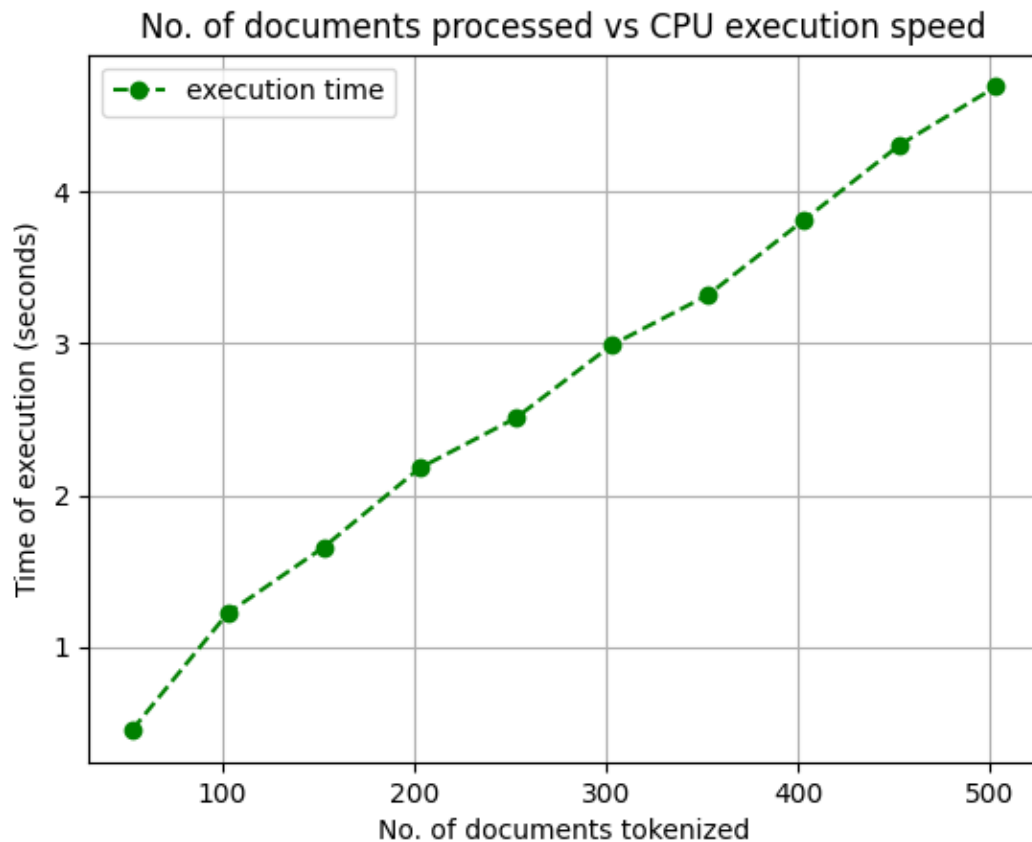
- Although Gensim removed numbers, symbols and punctuations there were few non-alphabets such as empty spaces were added to tokens as underscore ('_'). These were removed using regular expression. Therefore, I simply removed underscores using regular expression, while taking care of not removing Latin letters such as “ç”, “é”, etc. This is very important while dealing with search queries.

Performance Table and graph:

- time module (Python) is used to calculate CPU performance for processing the documents.

Number of documents processed	CPU execution time (seconds)
53	0.4519
103	1.2247
153	1.6595
203	2.1807
253	2.5102
303	2.9911
353	3.3196
403	3.8139
453	4.3097
503	4.6922

- HashMap uses quick look up with time complexity of $O(1)$ which is faster.



As we can see here, all the documents were tokenized in approx. 4.7 seconds.

Output of the program:

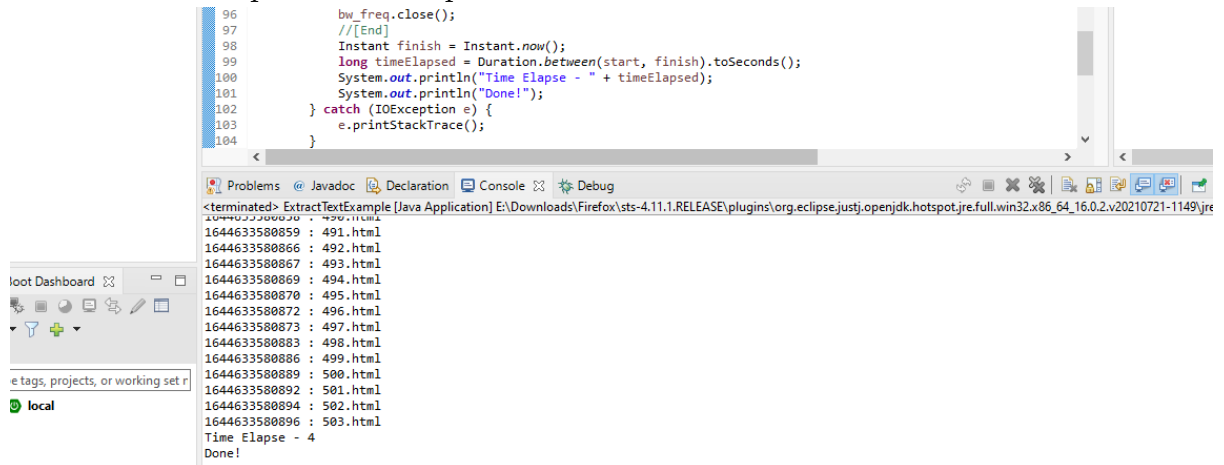
```
(venv) Bhushans-MacBook-Air:Project-1 bhushan$ python cmsc_676_p1.py files output
CPU time for tokenizing 53 docs: 0.45194199999999896
CPU time for tokenizing 103 docs: 1.2247249999999985
CPU time for tokenizing 153 docs: 1.6595040000000028
CPU time for tokenizing 203 docs: 2.1807870000000005
CPU time for tokenizing 253 docs: 2.5102650000000001
CPU time for tokenizing 303 docs: 2.9911040000000014
CPU time for tokenizing 353 docs: 3.3196170000000015
CPU time for tokenizing 403 docs: 3.8139330000000156
CPU time for tokenizing 453 docs: 4.309758000000001
CPU time for tokenizing 503 docs: 4.692298000000001
(venv) Bhushans-MacBook-Air:Project-1 bhushan$
```

Comparison with another Tokenization technique:

I compared my tokenization mechanism using the Gensim with Devang Vaidya's tokenization mechanism (Email: devangv1@umbc.edu)

- He has used TreeMap which internally uses HashMap which in turn uses hashing internally.

- He performed tokenization in Java in which he used JSoup library.
- To manipulate the text data by fetching HTML file he used “.text()” API where he used regular expression- “[^a-zA-Z]” to handle the punctuations.
- After comparing his results, it is evident that his mechanism and my mechanism has almost same performance speed.



The screenshot shows the Eclipse IDE interface. The main editor displays a Java class named `ExtractTextExample` with the following code:

```

96         bw_freq.close();
97         //[[End]]
98         Instant finish = Instant.now();
99         long timeElapsed = Duration.between(start, finish).toSeconds();
100        System.out.println("Time Elapse - " + timeElapsed);
101        System.out.println("Done!");
102    } catch (IOException e) {
103        e.printStackTrace();
104    }

```

The bottom of the IDE shows the **Console** view with the following output:

```

<terminated> ExtractTextExample [Java Application] E:\Downloads\Firefox\sts-4.11.1.RELEASE\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre
1644633580859 : 491.html
1644633580866 : 492.html
1644633580867 : 493.html
1644633580869 : 494.html
1644633580870 : 495.html
1644633580872 : 496.html
1644633580873 : 497.html
1644633580883 : 498.html
1644633580886 : 499.html
1644633580889 : 500.html
1644633580892 : 501.html
1644633580894 : 502.html
1644633580896 : 503.html
Time Elapse - 4
Done!

```

On the left side, there is a **loot Dashboard** panel showing a list of tags, projects, or working sets, with a **local** tag selected.

Above image is Devang Vaidya's result.