Information Retrieval Homework 1

Pradeep Agrawal (IQ57213)

Language used: Python

Following is the step-by-step explanation of how I did this:

1. I downloaded the tarfile and then unzipped it. By doing this, I got 503 HTML files. I put all the files in a directory named ‘files’.
2. Then I wrote a python code in which I used os.scandir to iterate through all the HTML files. In each iteration, I was checking if the current iteration holds a file.
3. For each valid file, I was calling a method ‘extract\_tokens’ by passing the filepath. This method was responsible for extracting the words out of the HTML file.
4. For this word extraction, I am using BeautifulSoup (a python library). This library is providing me all the text present in the file. Once I got the text, I broke that into words by using split() method. After getting the words, I am using a Regular Expression to check if the words have any special characters or numbers. If they had such things, then I removed those characters. And then I took only those words which were greater than 2 in length and lowercase all of them.
5. Once I got those tokens, I passed those tokens to a method called ‘create\_token\_files’. This method also requires output directory path. So, this method will check if the output directory exists. If not, will create that directory and then write these tokens into a text file separated by newline. So, this will result in having 503 txt files of tokens.

Text

Description automatically generatedText

Description automatically generated

1. I wrote a method ‘create\_frequency\_files’ to create frequency files. So, when I got the token files, I called this method by passing output\_dir as parameter. This method iterates over these text files and read words in them. Here I am using python dictionary to hold frequency of each word. So, if the word does not exist in the dictionary, then I add that word as key with 1 as its value. And if it exists then I increase its count by 1. At the end of this loop, I get frequencies of every word respectively in this dictionary. Using this, I created 2 files. One having words sorted by tokens alphabetically and second sorted by their respective frequency.

**Code Execution Guide:**

We just need to run the python file with 2 command line arguments (input files path, output files path). Keep in mind that we have to have the input-directory in the same path where we have the tokenize\_search.py file.

python tokenize\_search.py <input-directory> <output-directory>

Ex. python tokenize\_search.py html-files-dir text-files-dir

In above command ‘tokenize\_search.py’ if the python file name. ‘html-files-dir’ if the directory which holds all the HTML files and then ‘text-files-dir’ is the directory name where we will put all the text token files.

Following are screenshots of the output:

Text

Description automatically generated

Above shown the time taken in parsing those HTML files and converting them into token files. And finally, creating 2 frequency files.

**Code Execution Flow:**

When we run the command, it will start execution and will call the ‘main’ method that will run the entire script in the required sequence as shown in the screenshot.

Text

Description automatically generated

**Incorrectly Tokenized Words:**

After this entire process, I investigated both the generated frequency files and noticed that there are some random words that does not make any sense are there. Those words are some random collection of characters.

Ex. abaloldalizasnak, abortusztanacsadas, aceliparreorganizacios, acsaladtamogatasok etc

It seems like there were some special characters in between these words inside somewhere which got removed while cleaning the data and I end up getting such words.

**Results Comparison:**

* I compared my results with Vinayak’s and Koby’s works. And I noticed that Vinayak is using Jsoup parser to parse the HTML files while I used BeautifulSoup.
* Vinayak is removing all the special characters and converting it into lowercase and then splitting them by whitespace. While I am removing special characters, numbers and symbols. In addition to that I am keeping only the tokens which have character length of greater than 2.
* Koby’s frequency file have tokens like ‘a’, ‘’s’, ‘79yr’, ‘€’ etc. While my frequency file contains only words which does not have any integers in them.
* Koby’s code is taking approx. 83 seconds in the execution of code. While my code takes ~6 seconds in worst case.

**Graphs:**

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

**Token files sorted by alphabets:**

**Text

Description automatically generated with medium confidenceText

Description automatically generated with medium confidenceScatter chart

Description automatically generated with medium confidence**

**Token files sorted by frequencies:**

**A picture containing text

Description automatically generatedA screenshot of a computer screen

Description automatically generated with low confidenceA picture containing text

Description automatically generated**