CMSC 676 : Information Retrieval(Spring 2020)

Student Name : Ambrose Tuscano
Student ID : RE90613

Homework 1

Background:

Documents in Information Retrieval systems need to be index a corpus to provide users the ability to query them.These documents can range from basic text files to PDF documents, emails, etc. Therefore an intelligent IR system needs to have methods for identifying and handling these varying document formats and thus to get information from it. I designed a document parser which tokenizes HTML formatted documents, or basic web pages.

Implementation:

Having worked with python before but with no experience using it to scrap and tokenize files; I felt it a good sandbox for working on the document parser. As a modern programming language it automatically handles strings and file streams in a convenient way.

I started with a step by step implementation as per the Homework Documentation provided.

A. **Call DIR_CHECK()** :

1) I read the input files from the input directory and saved them in an array.
2) Next, I handled creating a corpus of the documents and in the same loop I tackled creating tokens for individual documents.
    - I first read individual files as f and used the BeautifulSoup Library to retrieve the text..
    - I stored the file in document variable and through the loop added all of them together.
    - Now, as I have scrapped the text out. I create tokens for the words in it. For this, I have defined a create_token_files function.
    - Now, comes sorting the corpus as a whole. For this I have defined a sorted_entire function and passed the entire_corpus as a parameter.

B. **Def CREATE_TOKEN_FILES(individual_file_name,data) :**

1) Here I first tackled creating a directory to save the tokenized files. I used OS library function- path.exists() and makedirs() for the same.
2) Next I imported ***Natural Language Toolkit(NLTK)*** library and used is word_tokenize function on the document to create tokens, now to deal with the punctuations and stop words, I used the lower() function to convert all tokens to lower case and isalpha() to just search just alphabets and in turn remove alphanumeric, Numerical and punctuation.
3) I looped and opened all the individual documents in the directory as mentioned.

4) Here if word was in token as discussed in point 2 and it was not a stop_word(another defination fro NLTK

Library), I wrote it in the file and closed the file. This looped till individual files were called up.

C. **Def SORTED_ENTIRE**(enire_corpus):

1) Here, there were two solutions. I either make a corpus of all the input files, which were not tokenized or make a corpos of the files I had just tokenized earlier.
2) I went the traditional way and added all the untokenized documents in the corpus first. Then in the defination of **sorted_entire**, I ran the word_tokenizer to tokenize the entire corpus.
   Now, I maintained two arrays, token_new, which had all the recent tokens, and token_new_1 which had all Of tokens, but not stop words.
3) Now, I counted the word frequency and made a dictionary of it to be used. I assigned the same dictionary to Freqdict.
4) I sorted the freqdict as per its items(tokens) and stored it in a string a.
5) Then I opened a file, Sorted_by_frequency and printed a in it if a was not in stop word. With this my sorting Tokens by frequency was done.
6) Now, I took the same FreqDict and sorted the same by its key, which holded for the sorting by tokens.
7) Now I saved both of the text documents in the Data Folder.

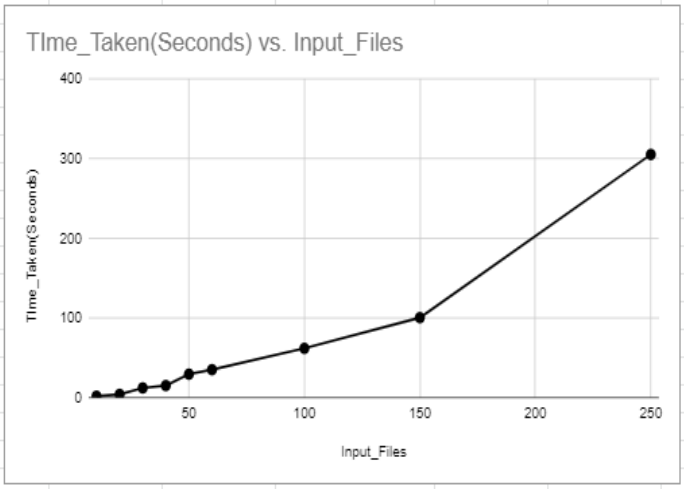
Drawbacks :

This approach though effective is time consuming, due to putting the whole documents in a corpos, I observed that it took much more time to execute.

The time taken to execute was inversely proportional to the number of files, with the 503 files it took up over 45 minutes to get results for sorting based on frequency and tokens.

So, my solution though it gives output, would never be optimal in any case. And it is better to maintain a dictionary of tokens rather than this method.

Input_Files	TIme_Taken(Seconds)
10	1.7843555
20	4.0262035
30	12.1326301
40	15.116694
50	29.4208038
60	35.0702518
100	61.8949155
150	100.4921097
250	305.5433997



Input_Files	TIme_Taken(Seconds)
10	1.7843555
20	4.0262035
30	12.1326301
40	15.116694
50	29.4208038
60	35.0702518
100	61.8949155
150	100.4921097
250	305.5433997
503	184532.8746