ASSIGNMENT 2 REPORT

SUBMITTED BY

MRINAL AICH (CS16MTECH11009)

SHAMIK KUNDU (CS16MTECH11015)

VAIBHAV SINGH CS16MTECH11017)

Files indexed: 1000

Index Created: 4

Graph Created: 2

CODE DESCRIPTION

Code is divide into different functions to perform tasks. First various libraries are needed to be imported. Explanation of various functions are given below:

createStopWord() :

In this function, a list of stop words are extracted by crawling the data present in the website given the problem statement . For crawling, BeautifulSoup library is used.

IsStopWord() :

The function checks whether a given word is a stop word or not.

calculateAvgGap() :

This function is used to calculate the average gap that is asked in the section 2 of the problem statement. The input passed to the function is the list of the terms whose average gap is needed.

StatisticalAnalysis() :

In this function, statistical analysis as mentioned in the section 2 of the problem statement. It uses the global variable dct(type-defaultdict). It extracts the most, least and median frequent words from this dictionary and makes a list and further calls the function calculateAvgGap() .

stemming(term) :

This function uses a popular Snowball stemmer to do stemming on the given terms. It is found in the 'nltk' package.

calculateIndexChar(indexName, dictionary) :

Dictionary and the name of the index whose characteristics are required are passed to the function. It calculates various characteristics of the index created and output of the index is printed on the python shell.

makeInvertedIndex(indexName, dictonary) :

This function makes the inverted index text file for a given index name and dictionary. In our program we have created 4 different types of indexes that are given in the problem statement.

makeI1(dctOld) :

This function is used to create a simple inverted index without any processing of data in it. The parameter passed is the dictionary variable 'dct'.

makeI2(dctOld) :

This function is used to remove the stop words from the previously generated index and making a new index and storing its value to a new text file. The parameter passed here is dictionary variable 'dct'. The output is a revised dictionary.

makeI3(dctOld) :

In this function, a dictionary is passed. We use this function to do stemming on the previously created index. The output is a revised dictionary.

makeI4(dctOld) :

In this function, the state of dictionary after going through the previous filtering processes of index are passed. This function separates the terms which are present in more than the 2% of the documents. The output of this function also creates a file I4.txt.

createGraph(graphList, xLabel, yLabel, plotName) :

It creates the graph given as a list of tuples having x and y co-ordinates values in it. For this we used ‘matplotlib.pyplot’ library. The file created here will be of .png format. We have used this function for plotting two graphs.

Flow of main function () :

A data structure *default dictionary with list as the value field* is used for creating inverted index. First the existing files are traversed and terms are extracted from it and with their respective doc-Ids. We used regular expression to take care of dates and filter out other not so useful data. We have generated terms from a file by splitting them about spaces. Then this dictionary is passed to function makeI1 , makeI2, makeI3 , makeI4 and StatisticalAnalysis. After that it is used to create two graphs.

Conclusion drawn from the graphs :

From graph 1 :

With increase in rank, the collection frequency is decreasing exponentially as more the collection frequency of the term, higher is the rank in our model. The graph is linear because it has been plotted between log(rank) and log(collection frequency). The graph obtained is similar to results obtained by zipf’s law i.e. collection frequency is inversely proportional to rank of the term.

From graph 2 :

Initially, the value of the number of terms in vocabulary is increasing with the increase in the collection size as new words are added in the vocabulary in the initial stages.
After a certain point it is getting parallel to the axis for collection size as the tokens encountered are already present in the vocabulary.
The graph is similar to that of y=root(x) which resides with the result of Heap’s law that is vocabulary size=k*(collection size)^b , where k is between 30 to 100 and b is approximately 0.5 .