**Homework 3**

An index for the documents is built. This is otherwise called the inverted index because instead of looking up a file to see the terms that occur we look at the terms to see which documents they occur in.

Preprocessing: The preprocessing has been kept the same as the previous parts of the homework as no problem was encountered with respect to preprocessing.

Methodology: The list of files from the input directory is read, this data is converted to text from html using htmlparser. The parsed text is then tokenized using NLTK tokenizer. This is then cleaned to a desired extend. The special characters are removed, stopwords from the given list in the previous homework. Now the data is ready for processing. Every Input file is name is taken the html part of it is stripped and the rest is retained to place it as document id for later use.

To get an inverted index, we need the following elements, the document id, the term and it’s weight in that document, the position of a term in the postings list and the number of documents the term occurs in.

To fetch all the elements, an efficient method would be to create the Term Document Matrix. In the function buildDF TDM is created using DataFrame from the Pandas. A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. Our Column here would be the document id, the index would be the terms and the cells would contain the weights if any for that document. The Data frame created would be as below.

|  |  |  |
| --- | --- | --- |
|  | Doc 1 | Doc 2 |
| Term1 | Weights | Weights |
| Term2 | Weights | Weights |

**Figure 1 :**

The weights here are the normalized TFIDF value. Term Frequency is calculated by

 (Number of times term t appears in a document)

(Total number of terms in the document)

IDF is calculated by :

df= No of documents that contain the term

N= Total number of documents

ln(N + 1 / df + 1) + 1

TFIDF=Tf\*IDF. With this formula we have observed better results compared to regular formula used in the previous assignment where the extra +1 was not added with N , df and the + 1 to the logarithm of the ratio. If a term is not there in corpus it will lead to division by zero, to avoid this we add a +1.

This could also be calculated using TFIDF vectorizer present in sklearn.feature\_extraction.text. We calculate the TDIDF value by using TFIDF vectorizer or TDIDF transform. We use fit(raw\_documents, y=None) which takes the raw documents and gives out TFIDFVectorizer. The weights are now obtained. Now that we have the elements, we build the term document matrix using the fit\_transform (raw\_documents, y=None) . This function returns the TDM, like shown in Figure 1. The Index of the dataframe is the term, the column name would be the document id and the values of each row or column would be the TFIDF value of every term in that document if present. The TDM or data frame built is in Figure 10.

Every column is saved as a dictionary in docAndWords where the key is the document id and the value is the list of words in that document. Then for every key (document id) the list of words is placed in a list. Every document id is also placed in a list of document ids so as to form the ultimate column in the TDM.

Now we iterate through this data frame (TDM). For every term we fetch the document it id present in (doc id) and the weights it has in those document. The iterator if printed would be as below for the word representative, where it shows that the term representative and its weights in every document starting from 001 and ending with 503. The variable row contains document id and the weights. The variable o1word contains the term. This is done for every single term.

**Figure 2:**

|  |
| --- |
| o1Word : representative |
| row : 001 0.000000 |
| 002 0.032687 |
| 003 0.000000 |
| 004 0.000000 |
| 005 0.000000 |
| 006 0.000000 |
| 007 0.000000 |
| 008 0.000000 |
| 009 0.000000 |
| 010 0.000000 |

Then we iterate though the row in the above example using the column name and the weight. The occurrence of the word in the document is counted, here we count the number of documents the term appears in. We also initialize a variable location, this is helpful to find the position of a term in the postings list where it first occurs or the location of the first record for that word in the postings file.

When each iteration is printed out, it shows as below. For a particular word, the columns and weight of that word in that column (document id). In column 005 the weight is 0.008236961086551706 for that word and so on.

**Figure 3:**

|  |
| --- |
|  |
| Col and Weight |
|  |
| col : 005 |
| weight : 0.008236961086551706 |
| col : 006 |
| weight : 0.0 |
| col : 007 |
| weight : 0.0 |

We set location to 1 in the beginning and once one term is done iterating, the count would give the number of documents it occurs in, (the number of entries in the postings list). Now we add count to location, this would give the beginning point for the next term, or the occurrence of the first record in the postings list for the next term. for example: if the term 1 is present in 3 documents, the value of count would be 3, the term 1 would have 3 entries in the postings list. The location of term 2 would start from location+count (1+3) 4th position in the postings list.

Output 1(Dictionary) is now generated

the word – **the term (olword)**  
the number of documents that contain that word (this corresponds to the number of records that word gets in the postings file)- **count**   
the location of the first record for that word in the postings file-**location**

To generate Output 2 we use the values from figure 3.

Output 2(PsotingsList)

the document id-**col**   
the normalized weight of the word in the document-**weight**

Write the outputs on the text files using the function writeOutput dictionaryFile(Output1) and postingsFile(Output2)

**Figure 5 No\_of Files vs Execution Time in sec :**

|  |  |
| --- | --- |
| No\_of\_Files | Execution\_Time\_in\_Sec |
| 10 | 2.6157 |
| 20 | 5.0525 |
| 40 | 9.365 |
| 80 | 16.338 |
| 100 | 21.253 |
| 200 | 39.533 |
| 300 | 64.056 |
| 400 | 110.6618 |
| 503 | 151.6258 |

**Figure 6 and 7 -Graphs for Figure 5 :**

We observe that as the number of files increases the time to create the dictionary and the postingsfiles also increases as more terms occur for whom the processing needs to be done, therefore graph increases has a high rate. The time is started before preprocessing starts and stopped after writing the 2 outputs on to output files dictionary and postingsfile.

**Figure 8: Analysis of the variation of dictionary and postings file size with respect to size of input file.**

|  |  |  |
| --- | --- | --- |
| Size of Input File-KB | Size of Output File -KB | |
|  | Dictionary | PsotingsList |
| 127 | 72 | 126 |
| 194 | 92 | 177 |
| 127 | 139 | 320 |
| 731 | 205 | 555 |
| 807 | 217 | 621 |
| 1460 | 317 | 1122 |
| 2970 | 439 | 2275 |
| 6330 | 1753 | 6269 |
| 11110 | 2057 | 7852 |

**Figure 9 -Plot of the table obtained above**

It is observed from figure 8 and 9 that as the size of the input file increases (10, 20, 40..like in Fig 5) the size of the output files increase too. It is also observed that it is the postingsList which proportionality increases with input size increase, whereas the increase in file size is not too much for the dictionary as the entries in it would be lesser compared to that of entries in postings file. In postings file we have separate entry for every occurrence of the term ,whereas in dictionary there would be one entry for a term with the documents it occurs in and the position of the first record in postings file.

We include the clear output to delete the output if already the file contains some entry and replace with the new output.

Verification:

The word Belgium in Dictionary would be as below, it is present in 11 documents and the postion of this word in the postings list would start at 38923 and continue for 11 values (as present in 11 documents)

**Dictionary**

Belgium

11

38923

**PostingsLsit**

A screenshot of text

Description generated with very high confidence

The word Belgium is first present in document 007 at position 38923 then in 156 and so on as shown in the above output. As it is present in 11 documents we have 11 entries for this word in the postings list. Here we verify the 2 outputs for correctness with the help of each other.

**Figure 10 -DataFrame or the TDM built**

**A screenshot of a computer

Description generated with high confidence**

Execution:

Python filename input\_directory output\_directory.

Improvements:

It is observed that words like believe, believable, believing have been considered here as separate words. This should not be the case, as weight would be calculated separately and when searched that specific words needs