Homework 2

Preprocessing: In homework 1, preprocessing was done to an extent. The stopwords, numbers and words with length 1 were removed. This has now been extended with removal of words that occur just once. Around 51230 words were of frequency 1, which when removed improved the processing time immensely. The stopwords in the previous assignment were taken from Stanford NLTK English Corpus but now has been taken from the link provided by the professor - https://www.csee.umbc.edu/courses/graduate/676/term%20project/stoplist.txt.

Term Weighting:

Term Frequency is calculated by (1)

$$\frac{\text{(Number of times term t appears in a document)}}{\text{(Total number of terms in the document)}}$$

Inverse Term Frequency is calculated by (2)

$$\log_e(\text{Total number of documents} / \text{Number of documents with term t in it})$$

In TF, each document is normalized to length 1 so that the shorter and the longer documents are treated equally, and we do not consider absolute frequency. Term counts, frequency of occurrence of the word is taken and compared with the total words in that document. Every document has different lengths and a term could occur a greater number of times only because the document in which the term is occurring is high in length, therefore dividing it by length of the document normalizes the frequency value.

In IDF, every word is normalized with inverse in corpus frequency to put less weight on common words which do not give a lot of intuition about the document and more weights are given to rare words. IDF basically helps in weighing down the common words. A natural log is involved just to scale down huge values. for example, in real data retrieval there could be over a million documents which is used to find IDF and this million in the numerator, would be a huge number to handle if log not taken

Then both tf and Idf is multiplied to get the final score.

The word ‘privacy’ is present in the 2’nd document 67 times, this word is present in 50 documents. Because of the occurrence of 67 times we would be giving very high weight to the word but the thing to be noted is that the document is also quite long (1998) words. To reduce the bias on long documents we find TF, normalizing the frequency by document length.

TF=67/1998 = 0.0335

The word is however not occurring in too many documents which means we need to give higher weight for this and normalize is across all documents, this is done by

IDF = ln(503/50) = ln(10.06)= 2.30

We now do a product of TF and IDF = 0.077337. This is an in-between weight value given to the word privacy in the document 2.

The word ‘Independent’ is present 3 times in the first document, the first document has 677 words.
Total no of documents is 503 and the number of documents containing international is 114 documents.
Applying formula 1 and 2

TF= (3/677) = 0.00443

IDF=ln(500/114)=1.47840965
TF*IDF= 0.00657. This is approximately how much is the TFIDF score of International in Document 1. This shows that the word independent is quite common in this set of 503 files, hence doesnot carry too much weight

Algorithm: Memory based approach is used to store information about the frequency distribution of every term. A temp variable is used to store the TF, another temp variable to store the IDF of every word(token) as dictionary. This dictionary contains the key ,value for document id, the term (token) and the TF ,IDF respectively. Then multiplication is performed for every pair of TF and IDF. Here the approach followed is processes every document one after the other, so as to get intermediate results instead of processing the whole bulk of documents at once.

Procedure

Every document is listed out in the input directory, done in the function readFromDirectory . The input to this is the input directory path in the system argument 1, output is a list of 503 files html files.

This is then passed through the function htmlparser() which reads the text section of the html page. Output of this is a chunk of plain text without any html tags. The encoding used id ISO-8859-1 this makes sure that a few characters like the ones present in the 13th file (\x97) will be handled. If utf-8 encoding is used instead ,we get stuck with characters or strings like (\x97) and we will not be able to proceed.

The text contains a lot of stopwords as one would expect in any English document. These words do not give us any important insight about the document. Hence, we should not waste resource processing these words any further. We remove the stopwords (the words to be removed is taken from https://www.csee.umbc.edu/courses/graduate/676/term%20project/stoplist.txt). The buildcustomstopwords() function builds a word list from English punctuations and the given stopwords. Punctuations are removed just to reduce the effort being wasted in calculating TF IDF for '/' ,',' etc.

The words of length 1 ,that occurs only once and numbers are removed . All the words are down cased to avoid the assumption by the system that words like Independent and independent are 2 different words. This process is in the cleanText function which uses the plain text output after html parsing as input and returns separate tokens as the output.

Every document's every word goes through term frequency calculation function freqdictionary() , forming the frequency dictionary,(eg. [{{'doc_id': 10, 'freq_dict': {'cpj': 1, 'press': 8, 'freedom': 2, 'briefing': 1, 'private': 9, 'challenges': 2, 'facing': 2, 'independent': 4, 'journalists':……..., '|index': 1, '|cpj': 1, 'website|': 1}}])

This is the dictionary formed for the 10th html document. The dotted lines are for all the tokens that are present in between (have not placed it to reduce the line consumption for the example).

This is a dictionary of a dictionary. The first key is the document number, second key is the frequency dictionary which stores every word present in that document as a key and it's frequency as value.

Now we calculate the TF score for every token, each document is passed through the function getTermFreq() where every word's frequency is taken and divided by the total number of words in that document. This is done for every word in every document. A list of tokens with their TF score is obtained as the output. This is stored in a temp dictionary (in Memory)

The tokens are then passed through getIDF, where the every document is checked separately to see if a particular word is present in it , If yes then the counter is incremented. In the end we have a count of how many documents out of 503 have the term in it. The total number of documents (503) (calculated by noOfDocuments) is divided by this counter value. This gives us the normalized IDF score for every token. The rare words would have higher scores and the common words would have lower scores. This value is stored in a temp variable.

Now that we have TF scores and IDF scores, we find the product of both for every token to get the final TFIDF score, which is done in the function calculateTFIDF() . The check happens as to, for every key in the TF(temp)

dictionary if there is a corresponding key in the IDF (temp) dictionary and if they both belong to the same document (ensured by doc id) then we multiply the values of these keys. This produces the TFIDF score.

Once the calculation is done every html input documents output is written to corresponding output txt document. This output document would contain every token that has survived prepossessing and it's TFIDF score. Therefore, we would have 503 output files with weights for 503 input files.

Program Testing and Output:

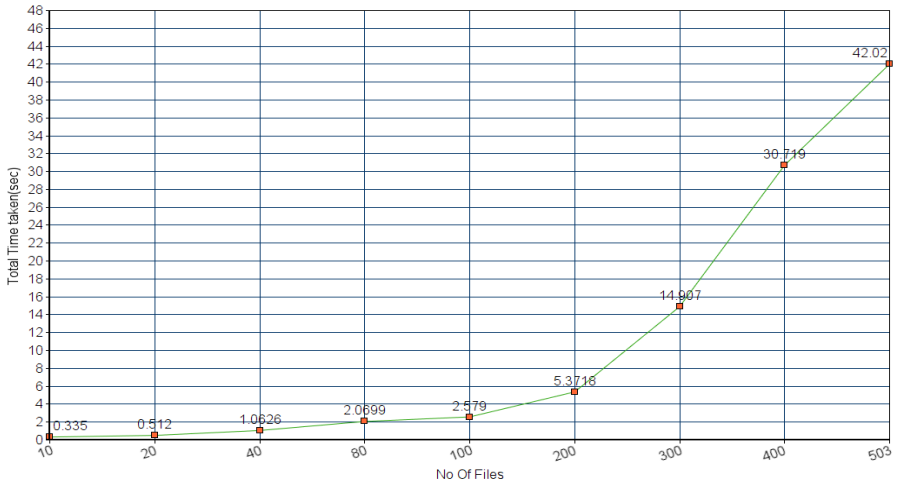
Calcwts input-directory output directory
Calwts-Name of the Program
Input-Directory - Directory containing 503 html files
Output-Directory - Directory containing 503 txt files, each file containing the tokens in the file and its calculated TF-IDF value.

Table for Time Comparison:

No_of_documents	Time (in sec)
10	0.335
20	0.512
40	1.0626
80	2.0699
100	2.579
200	5.3718
300	14.907
400	30.719
503	42.02

We see that as the no of files increases the time taken to process then increases linearly. It is of O(n) time complexity.

Graph of Total time taken vs No if Files:



Improvements:

Its noticed that reading the stopwords file from the course website instead of reading from locally stored file takes double the time, however for easy testing by the TA, I have included the reading from website in the submitted code (therefore might show higher time than usual). There could be improvements in the way the scrapping from website is done.

The program is taking a lot of time in processing the 503 files. This could be improved by maybe breaking down the elements or by using a faster language like Java for in memory fetching of data. A different data structure could also make a difference in performance time.