Hw2

Information Retrieval

Kurt Drexel

In starting my program I decided to begin from the point me and partner had left off at in Homework 1. The first change I made in my program was I went through the stoplist.txt file provided by the teacher. While going through the .txt file I added commas between all words and effectively transferred the list provided by the teacher into an array. Transferring this list into an array allowed me to simply copy and paste it into the original program over the position of our original stoplist. Doing this seemed the easiest way to implement the stop list to myself as in Homework 1 me and my partner had already implemented a fairly simple stop list so it was simply left down to a matter of altering the current stop list to the new stop list so that the program could account for all the newly added words to the stop list. The Second alteration I made to the program was the removal of the ability to sort alphabetically and by frequency. This was done because they were no longer needed within this assignment, and I deemed it not worth the computation time and storage for these un-needed components. The third alteration I made was the removal of words of length 1, while the stop list removed then words automatically, I decided to include it as it will still complete the necessary functionality removing word of length 1 including single digits. This was done by simply including a new else if statement within the already existing if statements to remove html document syntax. The simple if statement simply checked the length of the word at position i and if it is equal to 1 and then put that word into the already existing discard pile made previously for storage and removal of html syntax. I then created a deep copy of the words array. I decided to use deep copy as a normal copy would have been lost once a new file was tokenized and the words array is reset. The next alteration input into the program was the removal of words that occurred only once within each html document. To do this I iterate through my tokens dictionary created in the first homework, since the dictionary keeps track of the count of each word within it. At each word in the dictionary I checked if its occurrences is equal to 1. If the word is equal to 1 I delete the word from the tokens dictionary at position i. I then calculate its position within the occurrences dictionary and set it to the value of Pos. Using the new value at Pos I then use it to remove the word from the ProccessTok dictionary at position pos with the key i. I then delete it from the occurrences dictionary. I then subtract i by 1 to account for the word removed so that the loop with I can continue normally without skipping any words within the dictionary. The next portion of the program that needed to be added was the TF\*IDF calculations as well as normalization. To do this I first initialized sum on the outside of my for loop to iterate through the total number of documents. The sum variable will be used during the normalization of my weights for each word. I then reset the sum on the inside of this loop so that on every iteration it is set back to 0. As well as create a Weights dictionary to store the newly created weights. I then iterate through the my dictionary storing the tokenized words and the number of occurrences for each word. At each iteration I create a weight variable and set it to 0 this will become the result of the TF\*IDF calculation and later used within the normalization. I then use a if statement that ensures that while within the occurrences dictionary the program is in the correct document by its title. While within the correct document the program then begins to do the TF\*IDF calculation. I first calculate df by using len(Occurrences[token]) this line gets the length of the occurrences at each specific token giving the program the number of documents the token appears in or its document frequency (df). The next line then calculate the term frequency specified in the program at tf. To calculate tf I use ProcessedTok[DocNum].count(token) . This line return the count of the specified token in ProcessedTok while within the specific document number. I then complete TF\*IDF calculation for the specified word, with the line weight = math.log(1 + tf) \* math.log10(503 / df). This equation follows the equation provided in the powerpoint by the teacher with its equation being W(of term t in document d) = log(1 + term frequency) \* log10(Number of documents/document frequency of term t). Referring back to my line of code weight = math.log(1 + tf) \* math.log10(503 / df). In my program the number of documents is 503 for the html files given. I use math.log to calculate the log in both cases with math.log10 for log base 10. From there I simply plugged in my values for tf and df that I calculated in the above lines. The line after my calculations for TF\*IDF I begin to square the weight calculated to be used in my normalization. I do this with Sum += weight \* weight, this effectively squares the current weight and adds it on top of my Sum calculations. I then set the newly calculated weight within my weight dictionary at the specified document and specified token. At the very end of the loop outside of my if statement I then square root sum which at this point will be the updated weights of each word squared and added together. I then iterate through my Weights dictionary updating the new normalized weight for each word at the specified token and specified document dividing its current weight by the square root of Sum.

A computer screen shot of text

Description automatically generated

The next part of the program is very much the same from the original homework assignment printing out each word and its weight to the correct output file.

When testing my program with the allotted 10, 20, 40, 100, 200, 300, 400, then 503 html files I found the times to increase at a exponential rate. As the amount of html files increased the time to process the html files increased as well. I believe the large exponential increase in time seen between 400 and 503 files is due to the fact that there is a large portion of files at the ends that are much larger in size then others.

A screenshot of a computer

Description automatically generated I believe the theoretical runtime of my program can be broken into four parts with the first part being the tokenization of each document, then the removal of the stop words, the TF\*IDF and normalization, and finally the writing to my output files. The tokenization of each document will be O(N\*L) where L will be the length of each document and N will be the number of documents (503). The removal of stop words O(N\*L\*S) where L will be the length of each document and N will be the number of documents and S will be the number of stop words in the stop word array. The TF\*IDF and normalization will take O(N\*T) time where N will be the number of documents and T will be the number of unique tokens within each document after the stop word removal process. Finally writing to the output files will take O(N\*W) where N is the number of documents created and written to, and W will be the number of tokens written to each document. With all these parts of the program taken into account the final theoretical runtime of the program will be O(N\*L) + O(N\*L\*S) + O(N\*T) + O(N\*W).

Two example outputs of my program would have to be 001.html and 004.html. In the previous homework assignment where there was none of the preprocessing done in this homework assignment document 001.html had a output of 732 tokens after the preprocessing and deletion of words that occur only once the file had a length of 687. In html document 004 before preprocessing it had a size of 157 unique token and after preprocessing a size of 150. The best example of the large amounts of words that can be removed in processing is document 005.html where before preprocessing had a output of 3676 but after preprocessing it had 3214 processed tokens. I have attached document 025.txt and included a few lines from its output to show my approach through this process and its output. A screenshot of a computer

Description automatically generatedThis example output includes 20 lines from my 025.html document. In these 20 lines you can see the normalization working by effectively making every terms weigh much less then 1 so that the total of all weights within the output file combined will equal 1. I have also included an example output of 001.txt. In the html file 001.html the word international appears 4 times, while press appears 12 times being showed by its weight being double that of international. The word violence occurs only once easily shown by the fact that its weight is significantly less than that of every other weight within the file produced except those of words who also occur only once. In document 025.txt the word clients appears 3 times within the website as shown in its fairly high weights of .114545~ while the word relationions appears much less with a weight of .0387508 appearing with an actual count of 1.