**Ryan Condotta**

**Final Report Assignment 3**

**Question 1:**

In order to get the raw html from the 1000 pages from assignment two, I first created a python script name A3Q1.py. I separated the processes into three different functions of begin(), outputfile(), and removetags(). The begin() function works by going line by line of the output.txt file which gets one url at a time then passes the url, value(which is the label for each raw html file), and count was just a measure of debugging for my purposes. This process is shown below:

with open(filename1, "r") as file:

count = 0

temp = 0

for line in file:

value = value + 1

url = line

count = outputfile(url, value, count)

When it is passed to the outputfile function, it creates a new txt file in my desired location of Raw\_html\_Files with all of the html from BeautifulSoup function. It writes the response from the BeautifulSoup function to the new txt file and then continues till all of the files are done and there are 1000 txt files in the Raw\_html\_Files directory. The process is shown below:

response = urllib.request.urlopen(link)

soup = BeautifulSoup(response, 'html.parser')

filename = r"C:\Users\Ryan\Documents\WebScience\Assignment3\Raw\_html\_Files\\"+str(value)+".txt"

outfile = open(filename, 'w')

outfile.write(str(soup.encode('utf8')))

outfile.close()

The last step for this problem is to remove most of the html markups. This aspect of the problem is defined in the removetags() function. It opens each of the files in the Raw\_html\_Files directory one by one and removes the html markups through the re library and stores the clean text in a variable called cleantext which is eventually written to a new output file with the same number name and in the directory of Processed\_html\_Files. The removetags() process is shown below for removing the html markups as described above:

temp = 1

while temp < 1001:

stringers = ''

print("Iterator:", temp)

filename = r"C:\Users\Ryan\Documents\WebScience\Assignment3\Raw\_html\_Files\\"+str(temp)+".txt"

text = open(filename, 'r')

for line in text:

stringers += line

cleanr =re.compile('<.\*?>')

cleantext = re.sub(cleanr,'', stringers)

filename = r"C:\Users\Ryan\Documents\WebScience\Assignment3\Processed\_html\_Files\\"+str(temp)+".txt"

outfile = open(filename, 'w')

outfile.write(str(cleantext))

outfile.close()

temp += 1

Through the combinations of these functions, I was able to accomplish the task of getting the raw html and storing them into a location and then taking those files and taking out html markups and putting them into another directory. This accomplished the problem of getting the html and removing the html markups for all of the links.

**Question 2:**

To solve the problem of choosing a query term that appears in at least 10 files, I first created the python script A3Q2.py which contains two main functions of QuerySearch() and DetermineTF(). A lot of my tweets were based off of sports so I decided to use the query term of ‘game’ to search in each file. So, to begin I plugged ‘game’ into the function QuerySearch. The function works by going through each processed file one by one and determining if the desired word is found in that file and then outputting it to a new list. This way I could determine how many of the files have the word in it at least once. This process is shown below:

temp = 1

while temp < 1001:

print("Iterator:", temp)

filename=r"C:\Users\Ryan\Documents\WebScience\Assignment3\Processed\_html\_Files\\"+str(temp)+".txt"

with open(filename, 'r') as text:

for line in text:

if word in line:

print("Found", str(temp))

storenumber.append(str(temp))

temp += 1

Once I determined which files had the word ‘game’, I chose 10 files which were the 10 found with the words. I used these 10 files and then I used the DetermineTF() function to find the number of occurrences of the word in the designated 10 files that I have chosen. It will output the number of that ‘game’ in the designated files, as well as, the total number of words in the file to use in the excel file ‘table’. The DetermineTF() process is shown below:

for temp in list:

filename=r"C:\Users\Ryan\Documents\WebScience\Assignment3\Processed\_html\_Files\\"+str(temp)+".txt"

text = open(filename, 'r')

contents = text.read()

text.close()

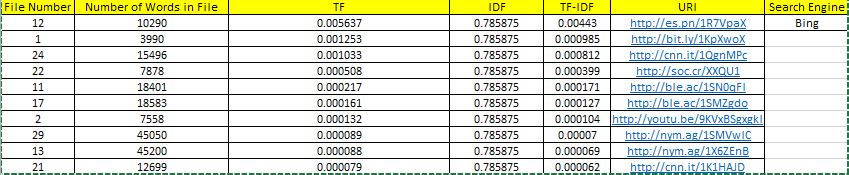
words = re.split("\W+",contents.lower())

words\_count = len(words)

print("Number of '" + word + "' in file", contents.count(word))

print("Number of Total Words in file", words\_count)

This was the first step for helping to basically initialize all of the variables that would be used in the table to determine the TF, IDF, and TF-IDF. The initialized variables are the frequency of the words in each file and total number count of the files. The table is shown below with the TF, IDF, and TF-IDF values, as well as the uri, total word count, and search engine used.



In order to determine TF, I used the frequency of the word/ total number of words in the file. For IDF, I used the search engine Bing and assume it has 2 Billion documents indexed and then divided it by the number of documents with that term. Then, I took the log base 2 of that number to get that answer. That is why the IDF is the same for all of the URI’s in this problem. Lastly, to calculate the TF-IDF I multiplied the TF\*IDF to get the TF-IDF and the ranked them in order from largest to smallest and also rounded all of my numbers to 6 digits for consistency. Through the combination of the python script, as well as, the excel file calculations I was able to accomplish the task of creating a table with TF, IDF, and TF-IDF.

**Question 3:**

For getting the page rank for each of the website, I used the website http://www.page-rank-calculator.com/ to get the page rank for each of the URI’s. Since there was only 10 URI’s to get the page rank for, I just did it by plugging and physically writing the information into table. Once, I wrote down all of the page ranks, I had to normalize all of the data based on the page rank. To do this, I used the formula of (xi- max(x))/ (max(x)-min(x)) to make sure the normalized data was between 0 and 1. I ranked the data from largest to smallest based on the normalized data. The table is shown below of the page rank, uri, page checker website used and normalized score.

|  |  |  |  |
| --- | --- | --- | --- |
| Page Rank | URI | Page Checker Website Used | Normalized Score |
| 0.7 | <http://soc.cr/XXQU1> | http://www.page-rank-calculator.com/ | 1 |
| 0.7 | <http://youtu.be/9KVxBSgxgkI> |  | 1 |
| 0 | <http://es.pn/1R7VpaX> |  | 0 |
| 0 | <http://bit.ly/1KpXwoX> |  | 0 |
| 0 | <http://cnn.it/1QgnMPc> |  | 0 |
| 0 | <http://ble.ac/1SN0qFI> |  | 0 |
| 0 | <http://ble.ac/1SMZgdo> |  | 0 |
| 0 | <http://nym.ag/1SMVwIC> |  | 0 |
| 0 | <http://nym.ag/1X6ZEnB> |  | 0 |
| 0 | <http://cnn.it/1K1HAJD> |  | 0 |

Comparing the two tables from Question 2 and Question 3, it seems as if there is no correlation between the page ranking the TF-IDF. This is based off the information given where the URI’s with higher page rank do not necessarily have the highest TF-IDF or lowest TF-IDF. This is just what the data has represented to me and I could make conclusive evidence off of.

**Question 4**

In order to compute the Kendal Tau value, I created the R script of A3Q4 which creates a table and 2 scatterplots which are shown below. Computing the Kendall Tau value, I used the normalized values from the page rankings and the TF-IDF values from Question 2 for the list. I began by creating two separate scatterplots with the x-values (TF-IDF) and y-value (normalized page rankings). One of the charts was just a scatter plot of the values and the other was a scatterplots of the rankings. I had to make sure the values were sorted in the correct order before I began to make sure that each value corresponding with each other value. I also put the Kendall Tau, 2 sided p-value, score, variance of the score and denominator into the table below to get some insight into what the vales mean. Since the tau value is 0, it shows that the two lists are independent of each other which makes sense why the score is zero and the p-value is so 1 as well.

|  |  |
| --- | --- |
| Kendall tau Rank Correlation | |
| Kendall tau | 0 |
| p-value | 1 |
| Score | 0 |
| Var(Score) | 58.6666679382324 |
| Denominator | 26.8328151702881 |