HW3 - Ranking Web Pages

Brenden Lewis
Due: 10/18/2020 11:59PM

Q1

Unfortunately, I had to acquire a brand new set of URIs to use for this assignment due to my dataset from HW#2 being comprised almost entirely of YouTube links; these kinds of links provide very little in terms of HTML content. For the sake of time, I only collected about 750 URIs through the Twitter API. I used a variety of keywords related to *tech* and *current events* in an attempt to broaden the contents of each page and provide a larger dictionary of words to utilize for this assignment. Unlike in my HW#2 submission, I was able to eliminate duplicates in my set and was left with 740 URIs before processing; these URIs were passed in as *CleanedURLs.txt*.

There were a few issues with some URIs throwing unique errors that I had to exempt due to not knowing how to work around. One of example of this was a link to a working Amazon page that was throwing a weird 503 error. After a bit of research, in this case, it was due to Amazon having their own API (similar to TWitter) that's required to scrape their data.

I downlaoded the HTML for each URI using the *requests* library and processed the HTML using *boilerpipe.extract* to extract the primary text. The raw HTML and processed text per URI was automatically stored in their own *.txt* files and saved in a local directory using a counter to label file names. This apporach resulted in a *rawHTMLFile#.txt* and *processedHTMLFile#.txt* file combo per URI stored in seperate folders locally.

```python
import requests
import urllib.request
import http.client
import os.path
from boilerpipe.extract import Extractor

count = 1

def download_HTML():
    urlFile=open('CleanedURLs.txt','r')
    #urlFile=open('testRankingURLs.txt','r')
    urlFileOPen= urlFile.readlines()

    global count

    for url in urlFileOPen:
        print("URI {}: Processing".format(count))
        try:
            response = requests.get(url)
            store_HTML(response.text,count)
            store_proc_HTML(url,count)
        except (requests.exceptions.RequestException):
            print("\tError: URI invalid")

        print("URI {}: Finished \n".format(count))
        count+=1

    print("\nDone\nTotal URIs processed: "+str(count-1))



def store_HTML(response,counter):
    rawHtmlOut = open(r"C:\Users\Brenden\Desktop\ODU\Fall 2020\CS432\HW_3\hw3-ranking-blewis1014\Raw_HTML\rawHtmlFile{}.txt".format(counter),"a", encoding="utf-8")
    rawHtmlOut.write(response)
    rawHtmlOut.close()

def store_proc_HTML(source,counter):

    while True:
        try:
            extractor = Extractor(extractor='ArticleExtractor', url=source)
            extracted_text = extractor.getText()
        except (urllib.error.HTTPError, (http.client.IncompleteRead), NameError, LookupError, UnicodeDecodeError):
            print("\tURI invalid")
            return False



        procHtmlOut = open(r"C:\Users\Brenden\Desktop\ODU\Fall 2020\CS432\HW_3\hw3-ranking-blewis1014\Processed_HTML\processedHtmlFile{}.txt".format(counter),"a",encoding="utf-8")
        procHtmlOut.write(str(extracted_text))
        procHtmlOut.close()
        return True




if __name__ == '__main__':
    download_HTML()
```

Listing 1: Python code for downloading and processing html

Q2

I opted to make a Python script specifically for sorting through my data set and collecting URIs that contained a specific query (as shown in Listing 2 below). Due to a large portion of my URIs being related to games and game companies, I chose the query ”Xbox” for collecting URIs to use in ranking later. In order to read through all the processed files I had stored, I used both the *glob* and *os* libraries in conjunction with the path to folder. For each file, text was read line-by-line, word-by-word to check for any matches to the query. To help with ranking the URIs later on, I was keeping count of each time the query was found in a file and the total amount of words the file possessed. Each file was stored as a dictionary with this information including the file name, and written to *QueryResults.txt* if the query ever appeared in the file.

```python
from progressbar import ProgressBar
import glob
import os

perc = ProgressBar()

path = r"C:\Users\Brenden\Desktop\ODU\Fall 2020\CS432\HW_3\hw3-ranking-
    blewis1014\Processed_HTML\*.txt"
query = "Xbox"
count = 1

containsQuery = []
all = []

for file in perc(glob.glob(path)):
    # infile = open(file,"r")
    occur = 0
    totalWords = 0
    with open(file, "r", encoding='ISO-8859-1') as f:
        # text = f.read()
        # occur = text.count(query)
        file_name = os.path.basename(file)

        for lines in f:
            words = lines.split()
            for i in words:
                totalWords+=1
                if i == query:
                    occur+=1

        file_dict = {"file":file_name,"words:":totalWords,"occurences":
    occur}
        count+=1
        if (occur > 0) :
            containsQuery.append(file_dict)
        all.append(file_dict)


outfile = open("QueryResults.txt","w")
outfile.write("Query: "+query+"\n")
for item in containsQuery:
    outfile.write(str(item)+"\n")

outfile.write("\nTotal # of documents: "+str(len(all)))
outfile.write("\nTotal # of documents that contain query: "+str(len(
    containsQuery)))
outfile.close()

results = open("QueryResultsForProcessing.txt","w")
for item in containsQuery:
    results.write(str(item)+"\n")
results.close()

print("Done")
```

Listing 2: Python code for collecting URIs that contain the query, query frequency, and total word count per document

With the query set to 'Xbox', 24 files were found to contain the query out of the 740. The output from this operation is shown Listing 3. Listing 3 shows 701 total documents because after running it the first time, I discovered that a good chunk of rawHTML and processedHTML files collected from Q1 were either empty or nearly empty (either 0 or 1 kb). I decided to delete those entries and run it again for the sake of consistency and not having missing data (GitHub won't allow uploading of empty files).

```
Query: Xbox
{'file': 'processedHtmlFile1.txt', 'words:': 412, 'occurences': 8}
{'file': 'processedHtmlFile103.txt', 'words:': 256, 'occurences': 8}
{'file': 'processedHtmlFile106.txt', 'words:': 407, 'occurences': 4}
{'file': 'processedHtmlFile11.txt', 'words:': 581, 'occurences': 11}
{'file': 'processedHtmlFile110.txt', 'words:': 407, 'occurences': 4}
{'file': 'processedHtmlFile119.txt', 'words:': 587, 'occurences': 11}
{'file': 'processedHtmlFile12.txt', 'words:': 481, 'occurences': 11}
{'file': 'processedHtmlFile120.txt', 'words:': 407, 'occurences': 4}
{'file': 'processedHtmlFile122.txt', 'words:': 252, 'occurences': 4}
{'file': 'processedHtmlFile123.txt', 'words:': 450, 'occurences': 4}
{'file': 'processedHtmlFile13.txt', 'words:': 225, 'occurences': 4}
{'file': 'processedHtmlFile15.txt', 'words:': 138, 'occurences': 2}
{'file': 'processedHtmlFile194.txt', 'words:': 1247, 'occurences': 1}
{'file': 'processedHtmlFile2.txt', 'words:': 48, 'occurences': 1}
{'file': 'processedHtmlFile235.txt', 'words:': 423, 'occurences': 1}
{'file': 'processedHtmlFile3.txt', 'words:': 213, 'occurences': 1}
{'file': 'processedHtmlFile4.txt', 'words:': 496, 'occurences': 6}
```

```
{'file': 'processedHtmlFile5.txt', 'words:': 207, 'occurences': 2}
{'file': 'processedHtmlFile6.txt', 'words:': 55, 'occurences': 1}
{'file': 'processedHtmlFile740.txt', 'words:': 323, 'occurences': 4}
{'file': 'processedHtmlFile8.txt', 'words:': 276, 'occurences': 7}
{'file': 'processedHtmlFile96.txt', 'words:': 326, 'occurences': 5}
{'file': 'processedHtmlFile97.txt', 'words:': 97, 'occurences': 2}
{'file': 'processedHtmlFile99.txt', 'words:': 212, 'occurences': 5}

Total # of documents: 701
Total # of documents that contain query: 24
```

Listing 3: QueryResults.txt

I searched through each file to determine which would be best to use for calculation; this was based on a combination of factors such as content, size, language (several were in Portugese, only opted to include one), and which ones I was curious to see the results of. The final 10 I chose to analyze are listed below in Listing 4. I had to manually add in the URLs for each of the 10 to make things a little easier when putting everything together in the end.

```
{'file': 'processedHtmlFile1.txt', 'uri': 'https://www.resetera.com/
   threads/xbox-brasil-dismisses-new-host-after-she-was-the-victim-of-
   death-and-rape-threats.307873/', 'words': 412, 'occurences': 8}
{'file': 'processedHtmlFile103.txt', 'uri': 'https://www.theverge.com
   /2020/10/15/21517293/xbox-series-x-s-dashboard-ui-october-update-
   release?_lrsc=f9510bb1-a7e2-43c8-9e49-761f32bbdc52', 'words': 256, '
   occurences': 8}
{'file': 'processedHtmlFile2.txt', 'uri': 'https://www.exophase.com/
   user/toniazzo/?1602892894', 'words': 48, 'occurences': 1}
{'file': 'processedHtmlFile194.txt', 'uri': 'https://newszoneweb.com/?p
   =5809','words': 1247, 'occurences': 1}
{'file': 'processedHtmlFile106.txt', 'uri': 'https://www.gamespot.com/
   articles/future-bethesda-titles-dont-need-to-be-on-ps5-says-xboxs-
   phil-spencer/1100-6483415/', 'words': 407, 'occurences': 4}
{'file': 'processedHtmlFile740.txt', 'uri': 'https://www.gamespot.com/
   articles/marvels-avengers-ps5-and-xbox-series-x-s-versions-delayed-
   to-2021/1100-6483398/', 'words': 323, 'occurences': 4}
{'file': 'processedHtmlFile12.txt', 'uri': 'https://tictechbuzz.com/
   remembering-the-classics-the-xbox-podcast-featuring-tim-schafer/', '
   words': 481, 'occurences': 11}
{'file': 'processedHtmlFile8.txt', 'uri': 'https://g1.globo.com/pop-
   arte/games/noticia/2020/10/16/microsoft-demite-apresentadora-do-xbox
   -br-que-sofreu-ameacas.ghtml?utm_source=twitter&utm_medium=social&
   utm_campaign=g1', 'words': 276, 'occurences': 7}
{'file': 'processedHtmlFile13.txt', 'uri': 'https://www.engadget.com/
   lucasarts-double-fine-remasters-game-pass-235238100.html', 'words':
   225, 'occurences': 4}
{'file': 'processedHtmlFile119.txt', 'uri': 'https://www.ign.com/
   articles/phil-spencer-xbox-bethesda-ps5?utm_source=dlvr.it&
```

```
    utm_medium=twitter', 'words': 587, 'occurences': 11}
```

Listing 4: Final set of 10 URIs chsoen for TD-IDF calculations

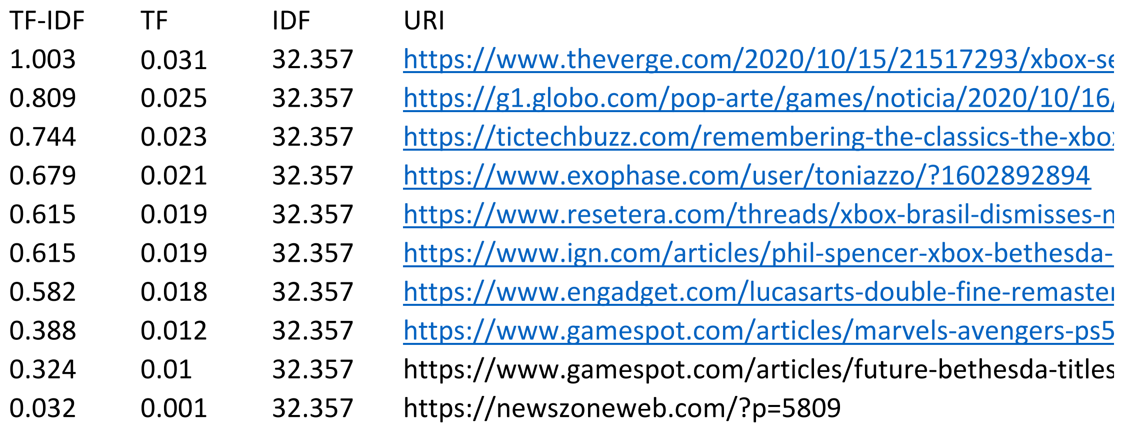
The Python code for calculating TF-IDF values (Listing 5) was relatively straight forward. I had to utilize the *ast* library to pull in the information from *FinalURLsForRanking.txt* as dictionaries to be used. I was using Google as my corpus, so that variable for calculation was 55b. The TF, IDF, and TF-IDF values were each calculated in their own individual function to 3 significant figures to reduce clutter. The *form operator import itemgetter* was needed to sort my final data in descending order by TF-IDF value; this wasn't completely necessary with how I created my table, but it definitely helped speed-up the process.

```
import ast
import math
import pandas as pd
from operator import itemgetter

collection = []
results = []
inputFile = "FinalURLsForRanking.txt"
query = "Xbox"
CORPUS = 55000000000

def calcTF(doc):
    freq = doc["occurences"]
    total_words = doc["words"]
    tf = float(freq / total_words)
    return round(tf,3)

def calcIDF(allDocuments, corpus):
    numDocs = len(allDocuments)
    idf = math.log2(corpus/numDocs)
    return round(idf,3)

def calcTFIDF(tf,idf):
    return round((tf*idf),3)

def drawResultsTable():
    pass

def printResults():
    for item in results:
        print(str(item)+'\n')

def writeResultsToFile():
    with open("TF-IDF_Results.txt","w") as f:
        for item in results:
            f.write(str(item)+'\n')
```

```

def sortTDFDescending():
    sort_list = sorted(results, key=itemgetter('TF-IDF'), reverse=True)
    return sort_list

if __name__ == '__main__':
    with open(inputFile, 'r') as f:
        for line in f:
            content = line.strip()
            collection.append(ast.literal_eval(content))

    idf = calcIDF(collection,CORPUS)

    for elem in collection:
        tf = calcTF(elem)
        tfidf = calcTFIDF(tf,idf)
        uri_ranking = {"uri":elem["uri"], "TF":tf, "IDF":idf, "TF-IDF":
   tfidf}
        results.append(uri_ranking)
    results = sortTDFDescending()
    printResults()
    writeResultsToFile()
```

Listing 5: Python code for calculating the TF-IDF for each URI

Figure 1 below shows the final ranking for each URI and their calculated TF, IDF, and TD-IDF values. For both the TD-IDF table and PageRank tables, I opted to use Excel to show the results as with only a few entries I didn't mind doing it manually. Looking at the results, an article from The Verge had the highest TF ratio of 8/256 which resulted in a larger TF-IDF. This is interesting as it ranked higher than the `https://tictechbuzz.com` link, which I expected to be the highest due to a higher query count; it had a larger query frequency, but also more total words at 11/481, resulting in a lower TF-IDF value. The lowest recorded value only had a single occcurence of the query over 1247 words. I was expecting it to be lower than every other URI, but I was still curious how low of a value would be calculated.

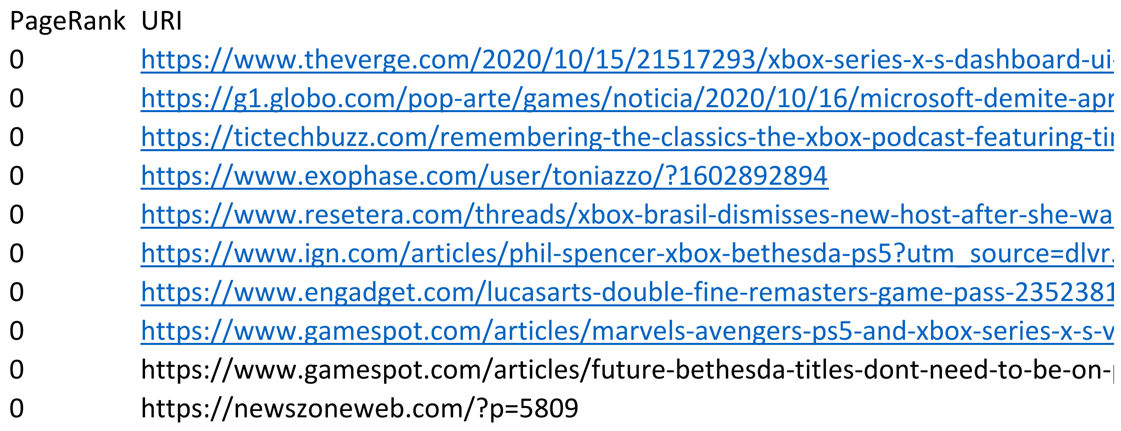


TF-IDF	TF	IDF	URI
1.003	0.031	32.357	https://www.theverge.com/2020/10/15/21517293/xbox-s
0.809	0.025	32.357	https://g1.globo.com/pop-arte/games/noticia/2020/10/16
0.744	0.023	32.357	https://tictechbuzz.com/remembering-the-classics-the-xbo
0.679	0.021	32.357	https://www.exophase.com/user/toniazzo/?1602892894
0.615	0.019	32.357	https://www.resetera.com/threads/xbox-brasil-dismisses-n
0.615	0.019	32.357	https://www.ign.com/articles/phil-spencer-xbox-bethesda-
0.582	0.018	32.357	https://www.engadget.com/lucasarts-double-fine-remaste
0.388	0.012	32.357	https://www.gamespot.com/articles/marvels-avengers-ps5
0.324	0.01	32.357	https://www.gamespot.com/articles/future-bethesda-titles
0.032	0.001	32.357	https://newszoneweb.com/?p=5809

Figure 1: Chart showing the TF-IDF, TF, and IDF for the query 'Xbox' across 10 chosen URIs

Q3

Figure 2 below shows the PageRank ranking for each URI. Each URI was checked using `https://www.prchecker.info/check_page_rank.php`. Unfortunately, due to these being obscure pages, every single URI returned with a 0/10 ranking as per the PR estimater. This is a sharp contrast to the TD-IDF table. I was a expecting a few URIs to have some kind of rank given their TD-IDF values, but was met with zeros across the board.



PageRank	URI
0	https://www.theverge.com/2020/10/15/21517293/xbox-series-x-s-dashboard-ui
0	https://g1.globo.com/pop-arte/games/noticia/2020/10/16/microsoft-demite-apr
0	https://tictechbuzz.com/remembering-the-classics-the-xbox-podcast-featuring-ti
0	https://www.exophase.com/user/toniazzo/?1602892894
0	https://www.resetera.com/threads/xbox-brasil-dismisses-new-host-after-she-wa
0	https://www.ign.com/articles/phil-spencer-xbox-bethesda-ps5?utm_source=dlvr.
0	https://www.engadget.com/lucasarts-double-fine-remasters-game-pass-2352381
0	https://www.gamespot.com/articles/marvels-avengers-ps5-and-xbox-series-x-s-v
0	https://www.gamespot.com/articles/future-bethesda-titles-dont-need-to-be-on-
0	https://newszoneweb.com/?p=5809

Figure 2: Chart showing the results of ranking each URI with PageRank