HW2 - Web Archiving

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

Q1

When observing Twitter posts through the API, I decided to opt for posts that contained the string ’Youtube’ in an attempt to keep my scope to YouTube video links. The Tweepy and Request libraries were utilized to extract the links from each Twitter post. I was not able to figure out how to check for duplicate URIs in an efficient manner, but I was able to extract about 1,200 total links related to YouTube in some capacity. Each twitter post streamed in was parsed to JSON format for observation.

I did not keep an exact count of the number of shortened links that redirected elsewhere, but observing all the links I was collecting both in testing and in my final collection were largely comprised of redirecting links. I created a recursive function called *def unshorten_url(short _url)* to ”unshorten” these links and return the final destination after however many redirects are needed.

Due to this method being called for every link to ensure it is the final location, the connection to the API was often breaking due to the large amount of processing happening during the stream. In order to alleviate this issue, I had to collect the links in batches to ensure a stable connection; after a good deal of testing, I had the most consistent success with a maximum of 70 links collected in a single run. With each run of the program, I added the collected links to a text file *TwitterURLs.txt*.

```
import tweepy
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
from tweepy import Stream
import sys
import re
import json
import datetime
import csv
import urllib.request
import requests
import urllib.parse
import http.client
from http.client import IncompleteRead

# Keys ommitted
consumer_key="***"
consumer_secret="***"
access_token="***"
access_secret="***"
```

```

# Handles authorization with Twitter
auth = OAuthHandler(consumer_key,consumer_secret)
auth.set_access_token(access_token,access_secret)
api = tweepy.API(auth)

link_contain = set() # Collection for uri links

class StdOutListener(StreamListener):

    def __init__(self):
        super().__init__()
        self.max_tweets = 70
        self.tweet_count = 0

    def on_data(self,data):
                if self.tweet_count == self.max_tweets:
                    print ("Finished")
                    export_data(link_contain)
                    return False
                else:
                    try:
                        tweet = json.loads(data)
                        for url in tweet["entities"]["urls"]:
                            tweet_url = str(url["expanded_url"])
                            r = requests.head(tweet_url, allow_redirects=True).url
                            full_url = unshorten_url(r)

                            if "https://twitter" in full_url:
                                print("----Twitter URL ommited----")
                            else:
                                self.tweet_count += 1
                                print ("URL"+str(self.tweet_count)+": " + full_url)
                                link_contain.add(full_url)

                                return True
                    except Exception:
                        print("Some issue occurred; Ending process")
                        export_data(link_contain)
                        return False


    def on_error(self,status):
        print (status)

```

```python
# If given a shorten url, expand to final url
def unshorten_url(short_url):
    parsed = urllib.parse.urlparse(short_url)

    if(parsed.scheme == 'https'):
        h = http.client.HTTPSConnection(parsed.netloc)
    else:
        h = http.client.HTTPConnection(parsed.netloc)

    resource = parsed.path
    if parsed.query != "":
        resource += "?" + parsed.query
    h.request('HEAD', resource )
    response = h.getresponse()
    if response.status/100 == 3 and response.getheader('Location'):
        return unshorten_url(response.getheader('Location')) # Recurisive call if not final location
    else:
        return short_url

def export_data(collection):
    url_storage=open("TwitterURLs.txt","a")
    for url in collection:
        url_storage.write(url+"\n")
    url_storage.close()




if __name__ == '__main__':
    l = StdOutListener()
    stream = Stream(auth,l)

    stream.filter(track=['Youtube']) # Filter Twitter posts to thosse that contain the string 'YouTube'
```

Listing 1: Python code for Q1

Q2

I opted to create a Python script to run MemGator from the command line for each link I had collected and return their TimeMaps in JSON format. To make creating the graphs easier after processing, I created a *uri _dict* dictionary for each entry containing the URI, number of mementos for that URI, and the age of its first memento. I capped my amount of processing at 1000 URIs worth of data. The program had to run continuously for about 10 hours to collect the

TimeMap for each link; each link took an average of about 30s - 1:10 mins to process. After all URIs are processed, the program immediately creates several graphs from the data provided by the TimeMaps. Listing 3 shows the terminal output of the program which prints each URI dictionary and the total number of links processed and the number of links that had at least 1 memento.

```
import os
import subprocess
import json
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import datetime
from datetime import date

timeMaps = []    # Holds TimeMaps for each URI
non_zero_data = []  # Holds TimeMaps for each URI with more than 0 mementos
tot_url_count = 0   # Total number of URIs
tot_mem_count = 0   # Total number of URIs with mementos

def collect_Timemaps():
    with open("testUrls.txt") as f:
        for line in f:
            global tot_url_count
            print("#"+str(tot_url_count+1)+": Collecting from: " + line)
            result = subprocess.run([’memgator’, ’-f’, ’JSON’, ’--hdrtimeout=1m30s’, line], shell=True, capture_output=True)
            output = result.stdout
            global tot_mem_count

            if output == None or output == b’’: # URI has no mementos and memgator call returns nothing
                memento_count = 0
                age_days = 0
                tot_url_count += 1
                print("Done")
            else:
                output_json = json.loads(output)     #Parses memgator output to a JSON object
                write_timeMaps_to_file(output_json)
                memento_count = count_mementos(output_json)
                age_days = calculate_age(output_json)
                tot_url_count += 1
                tot_mem_count +=1
                print("Done")

            uri_dict = {"uri":line, "mementos": memento_count, "age": age_days}  #Dictionary for URIs
            if memento_count > 0:
                non_zero_data.append(uri_dict)
            else:
                timeMaps.append(uri_dict)




def write_timeMaps_to_file(output):
    outfile = open("timeMaps.txt","a")
    outfile.write(json.dumps(output, indent=2, sort_keys=True))
    outfile.close()

# Counts the number of mementos a link has
def count_mementos(output):
    mementos = len(output[’mementos’][’list’])
    return mementos

# Calculate the age of the first memento a link contains in days
def calculate_age(output):
    first_mem = output[’mementos’][’first’][’datetime’] # 2020-10-01T03:07:29Z

    collection = datetime.date.today().strftime(’%Y-%m-%d’)
    datetime_coll = datetime.datetime.strptime(collection,’%Y-%m-%d’)

    first_date = datetime.datetime.strptime(first_mem,’%Y-%m-%dT%H:%M:%SZ’).strftime(’%Y-%m-%d’)
    datetime_first = datetime.datetime.strptime(first_date,’%Y-%m-%d’)

    age = datetime_coll - datetime_first
    return age.days

def print_timeMaps():
    global tot_url_count
    global tot_mem_count
    for item in timeMaps:
        print(item)
        # tot_url_count+=1

    print("\nTotal Urls: "+str(tot_url_count))
    print("\nTotal Urls w/ Mementos: "+str(tot_mem_count))


def create_histogram():
    output_data = []
    for diction in timeMaps:
        output_data.append(diction[’mementos’])

    non_zero_output = []
    for item in non_zero_data:
        non_zero_output.append(item[’mementos’])

    range = (0,1000)
    bins = 20

    plt.hist(output_data,bins,range,color="red",histtype=’bar’, rwidth=0.8)
    plt.xlabel("# of Mementos")
    plt.ylabel("Frequency")
    plt.title("Q2: Total URIs")
    plt.show()
    plt.savefig(’HistogramAll.png’)

    # Excludes URIs with zero mementos
    plt.hist(non_zero_output,bins,range,color="blue",histtype=’bar’, rwidth=0.8)
    plt.xlabel("# of Mementos")
    plt.ylabel("Frequency")
    plt.title("Q2: Total URIs with > 0 mementos")
    plt.show()
    plt.savefig(’HistogramNonZero.png’)

def create_scatter():
    x = []
    y = []
    for diction in non_zero_data:
        x.append(diction[’mementos’])
        y.append(diction[’age’])

    plt.xlabel(’# of mementos’)
    plt.ylabel(’Age in days’)
    plt.title("Q3 Total URIs")
    plt.scatter(x,y, c="blue")
    plt.show()
    plt.savefig(’Scatterplot.png’)

if __name__ ==’__main__’:
    collect_Timemaps()
    create_histogram()
    create_scatter()
    print_timeMaps()
```

Listing 2: Python code for Q2 and Q3

```
{'uri': 'https://www.youtube.com/watch?v=5Zk_Vz-wz4g&feature=youtu.be\n', 'mementos': 0, 'age': 0}
{'uri': 'https://www.youtube.com/watch?v=bFUCiVcGK9g&feature=youtu.be\n', 'mementos': 1, 'age': 7}
{'uri': 'https://www.youtube.com/watch?v=3ej_bkYiQnE&feature=youtu.be\n', 'mementos': 0, 'age': 0}
{'uri': 'https://www.youtube.com/watch?v=Vm7l3YxEF8g&feature=youtu.be\n', 'mementos': 1, 'age': 2}
{'uri': 'https://www.youtube.com/watch?v=0zdS8hNM-sc&feature=youtu.be\n', 'mementos': 0, 'age': 0}
{'uri': 'https://exe.io/LgNUIAt\n', 'mementos': 0, 'age': 0}
{'uri': 'https://www.youtube.com/watch?v=5hfYJsQAhl0', 'mementos': 1361, 'age': 3871}

Total Urls: 1000

Total Urls w/ Mementos: 496
```

Listing 3: Terminal output after program execution

Listing 4 below shows the JSON output format for each link; this example in particular is from the very first link in the url text file. The program counts each memento listed under *list* (if available) and the datetime of the first memento to calculate the age. The output for each link is stored collectively in a seperate textfile labeled *timeMaps.txt*. The final size of this file came out to about 33MB, with many of the URIs having several thousands of mementos.

```
{
  "mementos": {
    "first": {
      "datetime": "2020-10-03T13:47:32Z",
      "uri": "https://web.archive.org/web/20201003134732/https://www.youtube.com/watch?v=4yAO5hbnfQQ&gl=US&hl=en"
    },
    "last": {
      "datetime": "2020-10-03T13:47:32Z",
      "uri": "https://web.archive.org/web/20201003134732/https://www.youtube.com/watch?v=4yAO5hbnfQQ&gl=US&hl=en"
    },
    "list": [
      {
        "datetime": "2020-10-03T13:47:32Z",
        "uri": "https://web.archive.org/web/20201003134732/https://www.youtube.com/watch?v=4yAO5hbnfQQ&gl=US&hl=en"
      }
    ]
  },
  "original_uri": "https://www.youtube.com/watch?v=4yAO5hbnfQQ",
  "self": "http://localhost:1208/timemap/json/https://www.youtube.com/watch?v=4yAO5hbnfQQ",
  "timegate_uri": "http://localhost:1208/timegate/https://www.youtube.com/watch?v=4yAO5hbnfQQ",
  "timemap_uri": {
    "cdxj_format": "http://localhost:1208/timemap/cdxj/https://www.youtube.com/watch?v=4yAO5hbnfQQ",
    "json_format": "http://localhost:1208/timemap/json/https://www.youtube.com/watch?v=4yAO5hbnfQQ",
    "link_format": "http://localhost:1208/timemap/link/https://www.youtube.com/watch?v=4yAO5hbnfQQ"
  }
}
```

Listing 4: TimeMap of first link

Figures 1 and 2 below show histograms of the number of mementos from each URI and their frequency. Figure 1 shows the data from all URIs while Figure 2 attempts to show the same data excluding any URIs with 0 mementos. As also indicated in the program output, almost exactly half of the 1000 URIs observed had 0 mementos. Of the 496 links remaining, the large majority had very few mementos ranging between 1-100 mementos. Outside of a few outliers with thousands of mementos, most of the observed links are not well archived.

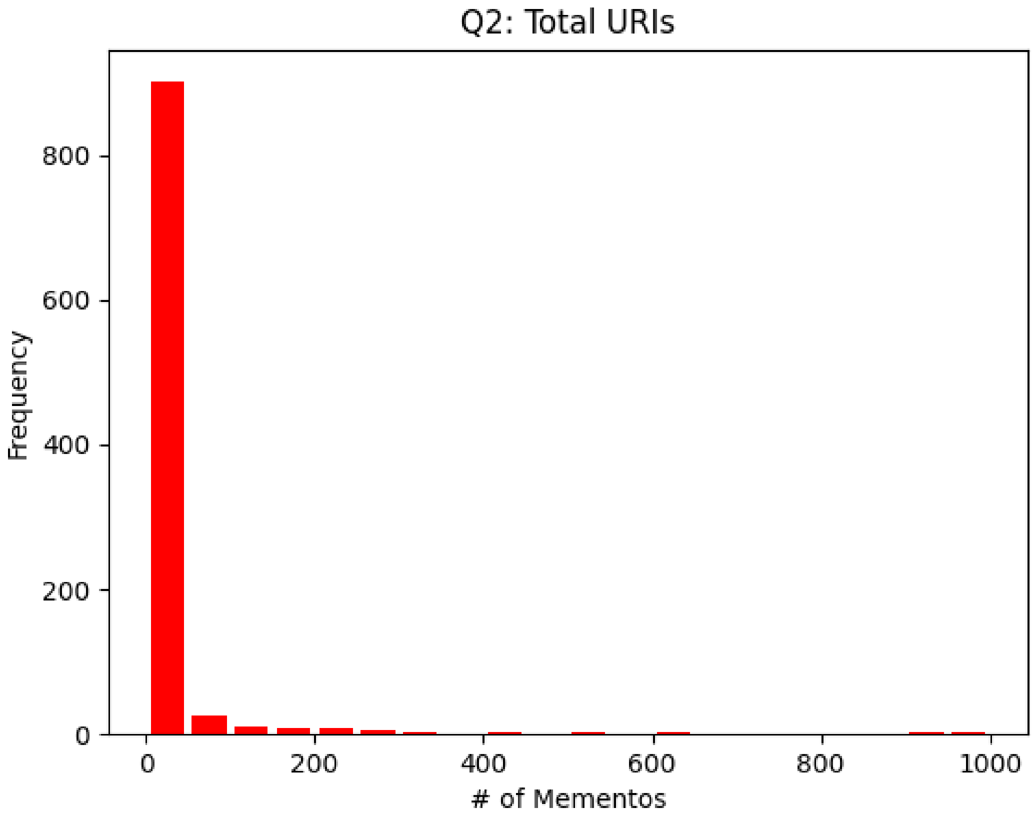

Figure 1: Histogram showing the number of URIs vs. the number of mementos

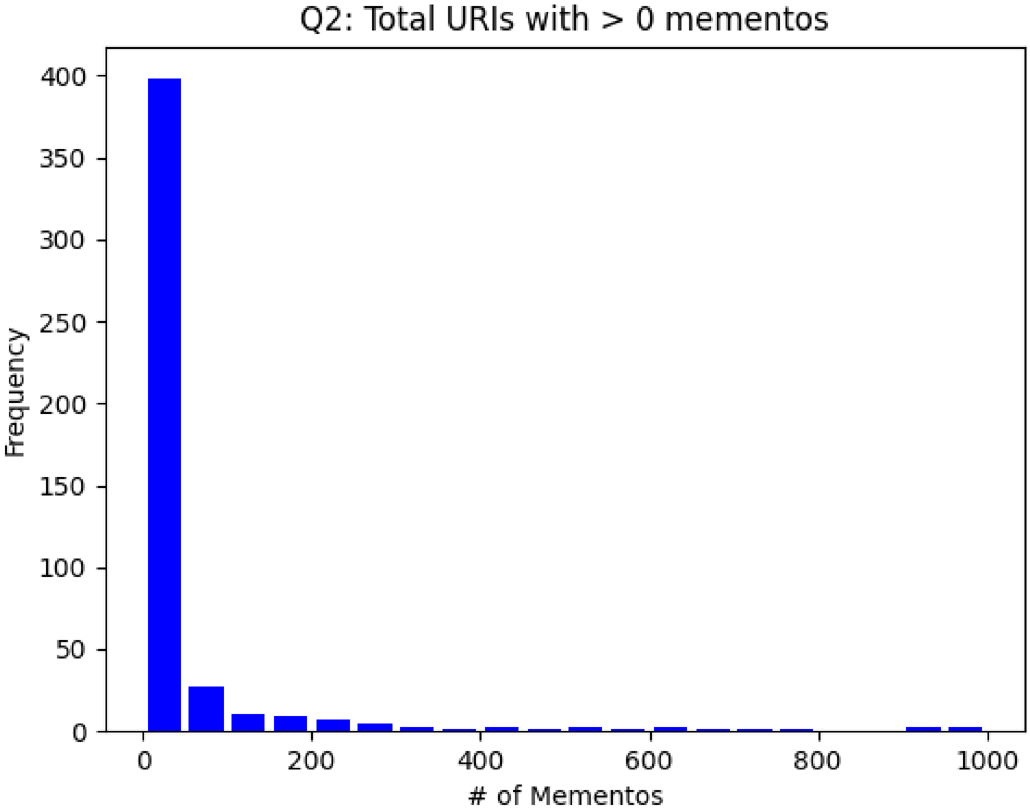

Figure 2: Histogram showing the data from Figure 1 excluding URIs with no mementos

Q3

For each URI with non-zero memento counts, the age (in days) of the first memento is calculated through the *def calculate _age(output)* function that utilizes the *datetime* library and takes the TimeMap of a URI in JSON as an argument. Figure 3 below is a scatterplot of all the URIs with non-zero mementos. Each URI is plotted by the number of mementos its TimeMap contains and the age of its first memento. Unfortunantly, due to an unidentified error, there is an outlier plotted that heavily skews the graph to the right. Observing the rest of the plotted data provides convincing evidence to the hypothesis that, for YouTube links, the number of mementos a URI contains directly correlates with its age.

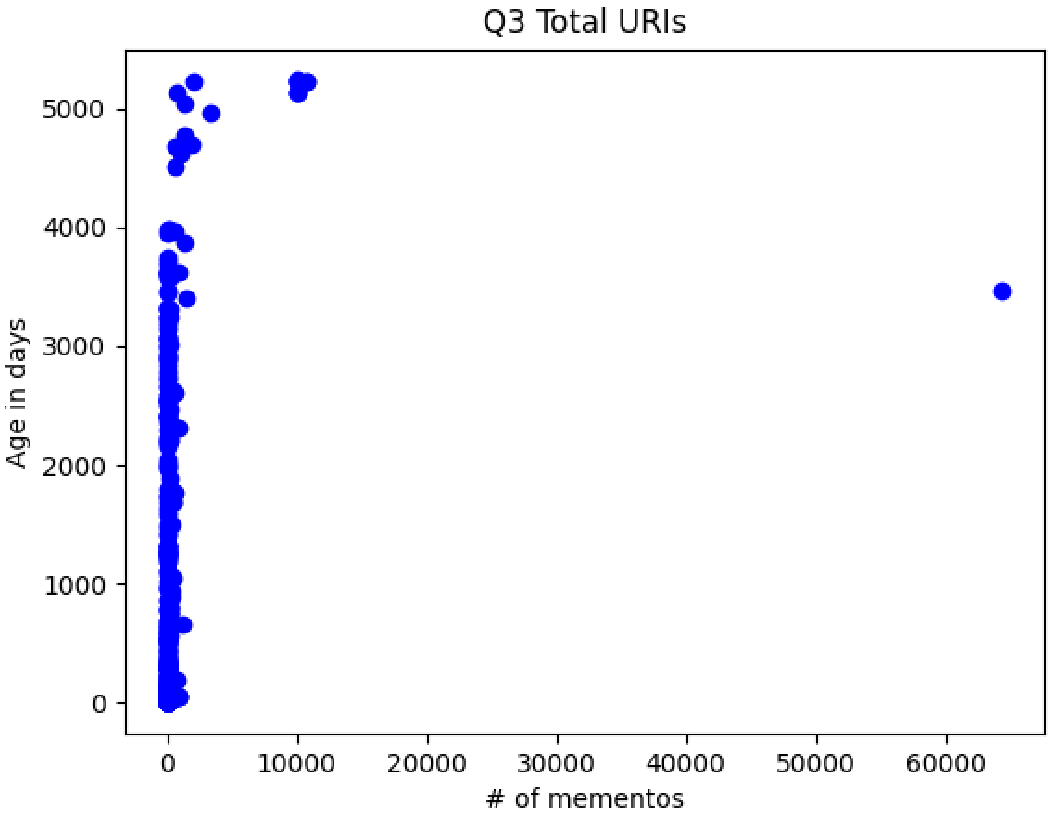

Figure 3: Scatterplot with the number of mementos vs. their age