2017

Assignment Two

ROSS REELACHART

PROFESSOR MICHAEL L. NELSON

Question 1

Before I began, I decided that it would be incredibly prudent to break the larger task of finding 1000 unique links on Twitter into smaller, individual programs. This minimized the danger of something going wrong, and taking all my work with it.

All python programs are written in python 2.7.

Associated github repository for this report located at:

https://github.com/rreelachart/cs532-s17/tree/master/submissions/assignment%202

In order to extract 1000 unique links from Twitter, I first needed to register an application with the Twitter API. Doing so created a consumer key, a consumer secret, an oauth token, and an oauth token secret. All four were needed if I wanted to retrieve[1] any tweets from Twitter. I found some code[2] that was originally intended for retrieving my own tweets from Twitter. Most of that code went unchanged, and I added two functions. "call_api" was used for getting tweets from other people using their screenname. "print_tweets" called "call_api" and printed out any retrieved tweets.

"getTweets.py" was the first python program I ran, multiple times with different accounts, in order to gather my initial batch of tweets. When run from the command line it took four arguments: <startingId><tweetPerCall><apiCalls><screenname>.

So, for example, if I wanted to make three API calls and get 200 tweets per call from director James Gunn's Twitter, starting with this tweet https://twitter.com/JamesGunn/status/829379702032736256, I would type into the command line:

"$ python getTweets.py 829379702032736256 200 3 James Gunn"

Then I would use the ">" output redirection to put all the grabbed tweets into a text file named "gunnTweets.txt" This process was repeated for several different accounts listed below:

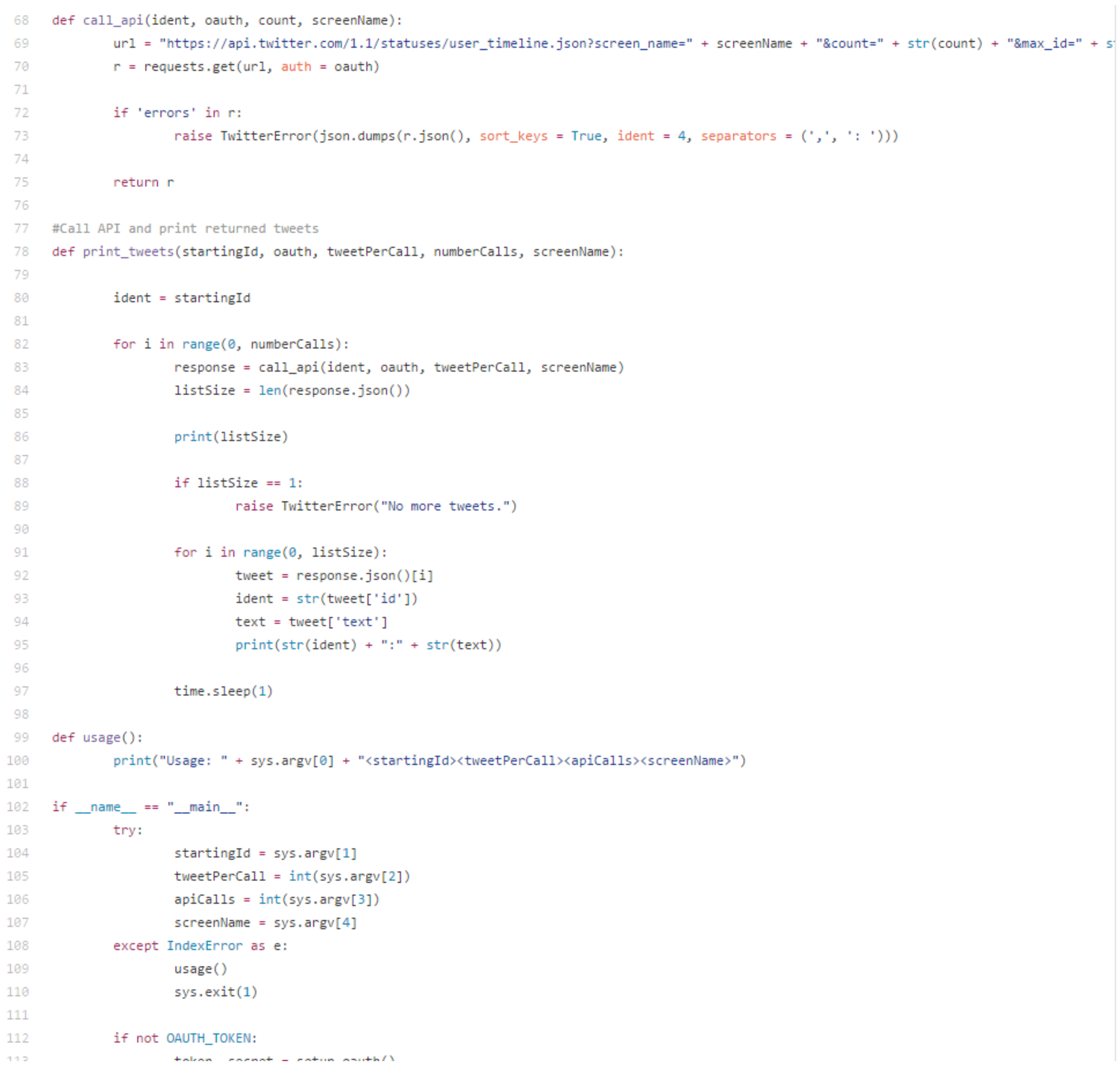
- "@JamesGunn" James Gunn. A director I really enjoy, probably best known for directing the "Guardians of the Galaxy" movies.
- "@RealGDT" Guillermo del Toro. Another director I really enjoy. He made "Pacific Rim," a giant robot movie I love.
- "@ID_AA_Carmack" John Carmack. Current CTO of Oculus VR and all around computer genius. Tweets about arcane programming stuff as if it was entirely normal.
- "@FilmCritHULK" Film Crit Hulk. A professional screen writer and film analyst whose writings I really enjoy. Used to write only in ALL CAPS.
- "@redlettermedia" Red Letter Media. Independent film makers and makers of funny YouTube videos. Responsible for a multi-hour video critique of the Star Wars prequels.
- "@Seanbabydotcom" Seanbaby. An internet comedy writer that I've been following for the better part of a decade.
- "@Coelasquid" Kelly Turnbill. A weirdo illustrator and web comic artist who dresses like a Mad Max character in everyday life.

These were all outputted into their own respective text files for further use.

Once all the tweets had been gathered into text files, they were run through a second python program that extracted all the URIs from the tweets and then tested to see if they end in a successful 200 code, "getURIFromTweets.py." This program took the name of a file as an argument.

Using the URI pattern of 'http://' or 'https://' the program extracted URIs and then made a request to them. If they ended in anything that wasn't a 200 status code, they were ignored. Again, the ">" output redirection was used to output the list of successful URIs into an associated text file that had a similar name to the tweet file. For example:

"$ python getURIFromTweets.py toroTweets.txt > toroURIs.txt"



```
def call_api(ident, oauth, count, screenName):
        url = "https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=" + screenName + "&count=" + str(count) + "&max_id=" + s
        r = requests.get(url, auth = oauth)

        if 'errors' in r:
                raise TwitterError(json.dumps(r.json(), sort_keys = True, ident = 4, separators = (',', ': ')))

        return r

#Call API and print returned tweets
def print_tweets(startingId, oauth, tweetPerCall, numberCalls, screenName):

        ident = startingId

        for i in range(0, numberCalls):
                response = call_api(ident, oauth, tweetPerCall, screenName)
                listSize = len(response.json())

                print(listSize)

                if listSize == 1:
                        raise TwitterError("No more tweets.")

                for i in range(0, listSize):
                        tweet = response.json()[i]
                        ident = str(tweet['id'])
                        text = tweet['text']
                        print(str(ident) + ":" + str(text))

                time.sleep(1)

def usage():
        print("Usage: " + sys.argv[0] + "<startingId><tweetPerCall><apiCalls><screenName>")

if __name__ == "__main__":
        try:
                startingId = sys.argv[1]
                tweetPerCall = int(sys.argv[2])
                apiCalls = int(sys.argv[3])
                screenName = sys.argv[4]
        except IndexError as e:
                usage()
                sys.exit(1)

        if not OAUTH_TOKEN:
                token, secret = setup_oauth()
```

Figure 1 A selection of my code from getTweets.py

This resulted in a number of files filled with URIs, extracted from the tweets. I then used a small python program “combineURILists.py” to combine all those URI text files into a single one. The program also eliminated any duplicates. This program took the names of any files as arguments.

“$ python combineURILists.py gunnURIs.txt toroURIs.txt carmackURIs.txt …. > combinedURIList.txt”

Again, the “>” redirection was used to put the result into one text file. The final file “combinedURIList.txt” had a total of 1506 unique URIs.

To get the final roster of exactly 1000 URIs, I made another python program that extracted 1000 random URIs from the combined list.

“$ python get1000Links.py combinedURIList.txt > 1000Links.txt”

“get1000Links.py” also ensured that no duplicate URIs were added to the final collection of 1000 URIs in “1000Links.txt”

Question 2

Before I began this step, I made the decision to, again, split up the workload. This time, it was both to ensure that I didn’t lose too much work if something went wrong, but also because it made the more time-consuming task easier to manage. I wasn’t willing to simply let a program run through my 1000 URIs all at once over a long period of time because that increases the changes of something going wrong, and because it made it more convenient for me. This way I could run this task over multiple sitting if needed.

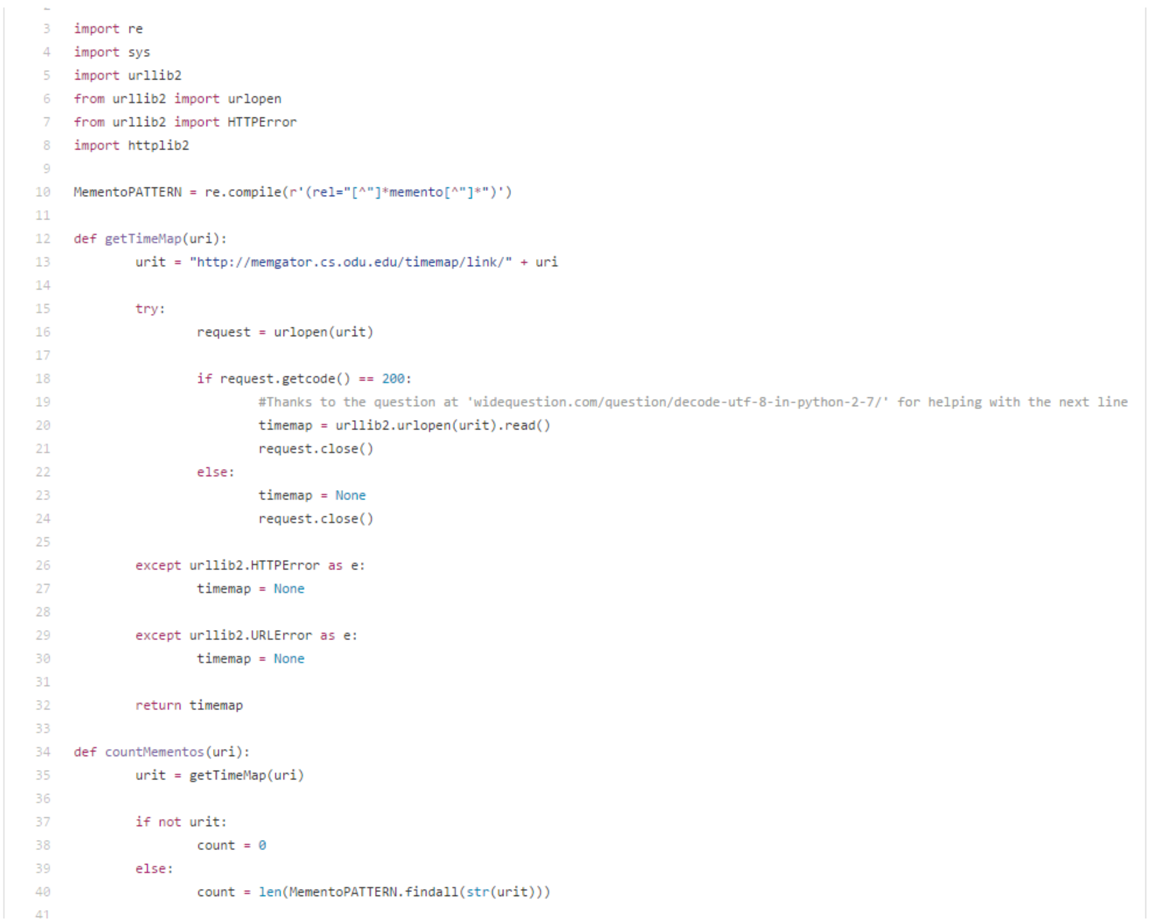
So, I took the original “1000Links.txt” file and split it up using the “split[3]” command.

“$ split –l 200 1000Links.txt” Links”

This resulted in five files, each with 200 links, named “Linksaa” through “Linksae.”

These files were then fed into the python program “getMementos.py” to extract the number of mementos each link had. This program used the TmeMap aggregator made available by ODU. The result was an associated text file that had two columns. The first was the number of mementos a URI had, and the second was the URI itself.

“$ python getMementos.py Linksaa > Linksaa.txt”



```python
import re
import sys
import urllib2
from urllib2 import urlopen
from urllib2 import HTTPError
import httplib2

MementoPATTERN = re.compile(r'(rel="[^"]*memento[^"]*")')

def getTimeMap(uri):
	urit = "http://memgator.cs.odu.edu/timemap/link/" + uri

	try:
		request = urlopen(urit)

		if request.getcode() == 200:
			#Thanks to the question at 'widequestion.com/question/decode-utf-8-in-python-2-7/' for helping with the next line
			timemap = urllib2.urlopen(urit).read()
			request.close()
		else:
			timemap = None
			request.close()

	except urllib2.HTTPError as e:
		timemap = None

	except urllib2.URLError as e:
		timemap = None

	return timemap

def countMementos(uri):
	urit = getTimeMap(uri)

	if not urit:
		count = 0
	else:
		count = len(MementoPATTERN.findall(str(urit)))
```

Figure 2 A selection of code from getMementos.py

The program simply took the URI, appended it to the aggregator URI, and then made a request using that new URI. The response would always be a lengthy file showing each individual memento and date for the URI. So the program used a pattern to find the string of characters that always indicated an archived memento, and counted the number of times that occurred. It was in this program that I also found that some of the URIs had encoding differences that resulted in errors. Thankfully, I found an easy that got around the decoding problems[4].

The resulting five text files were then combined using the concatenation command into a final memento list text file.

"$ cat Linksaa.txt Linksab.txt…. Linksae.txt > finalMementoList.txt"

This final memento list was used in R to create a histogram of the number of mementos vs. URIs. With the full list, which had one URI that had over 12,000 mementos, the histogram barely had any actual real data on it. The vast majority of URIs had a very low number of mementos, with only a handful having more than 100, and then even smaller number had magnitudes more. The second try with the histogram, which eliminated any URIs with >10,000 mementos barely changed the graph. Noticing the number of "bumps" on the far left side of the histogram, I decided to chart only URIs that >100 mementos, which resulted in a far more readable graph. This selection of URIs contains 966 out of a possible 1000 URIs. It seemed that my selection of URIs was mostly comprised of URIs with no mementos.

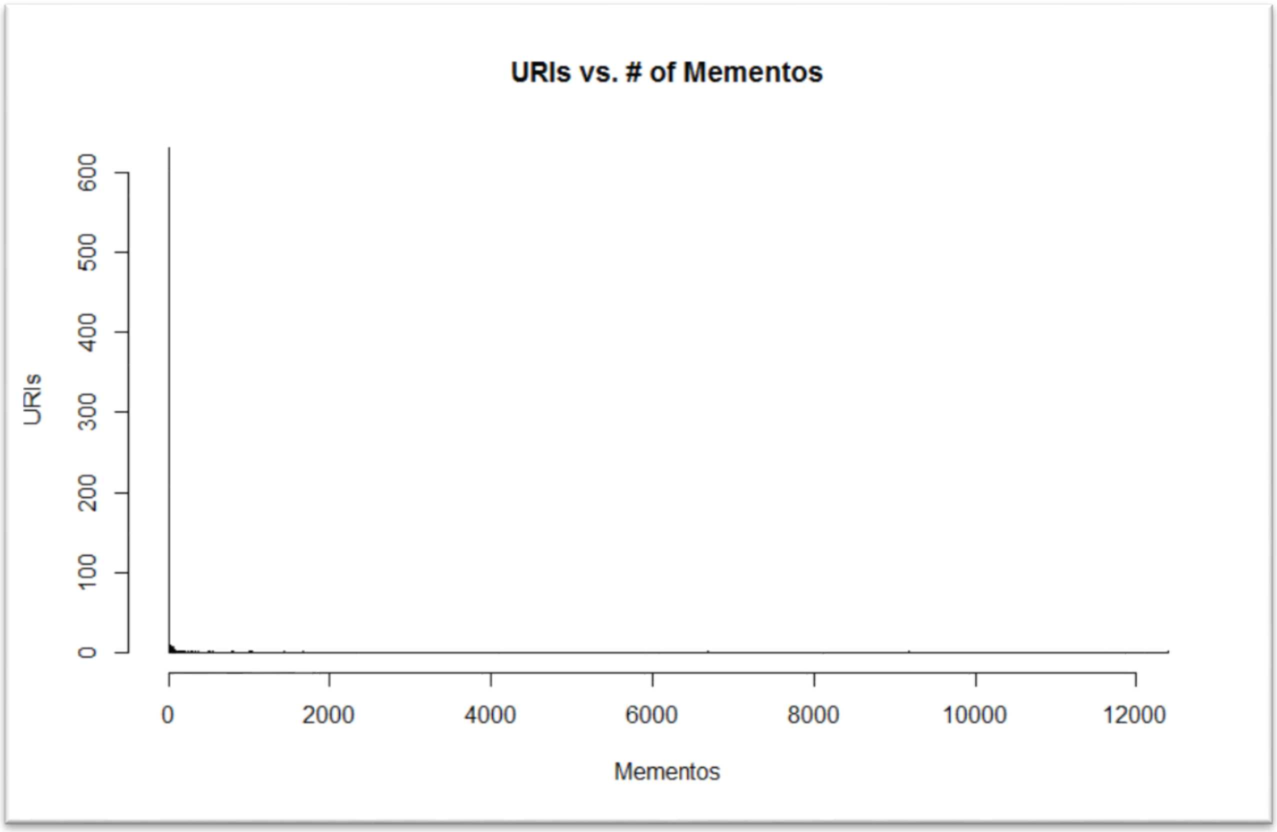


Figure 3 The first iteration of the histogram had very usable detail because of the outliers on the right.

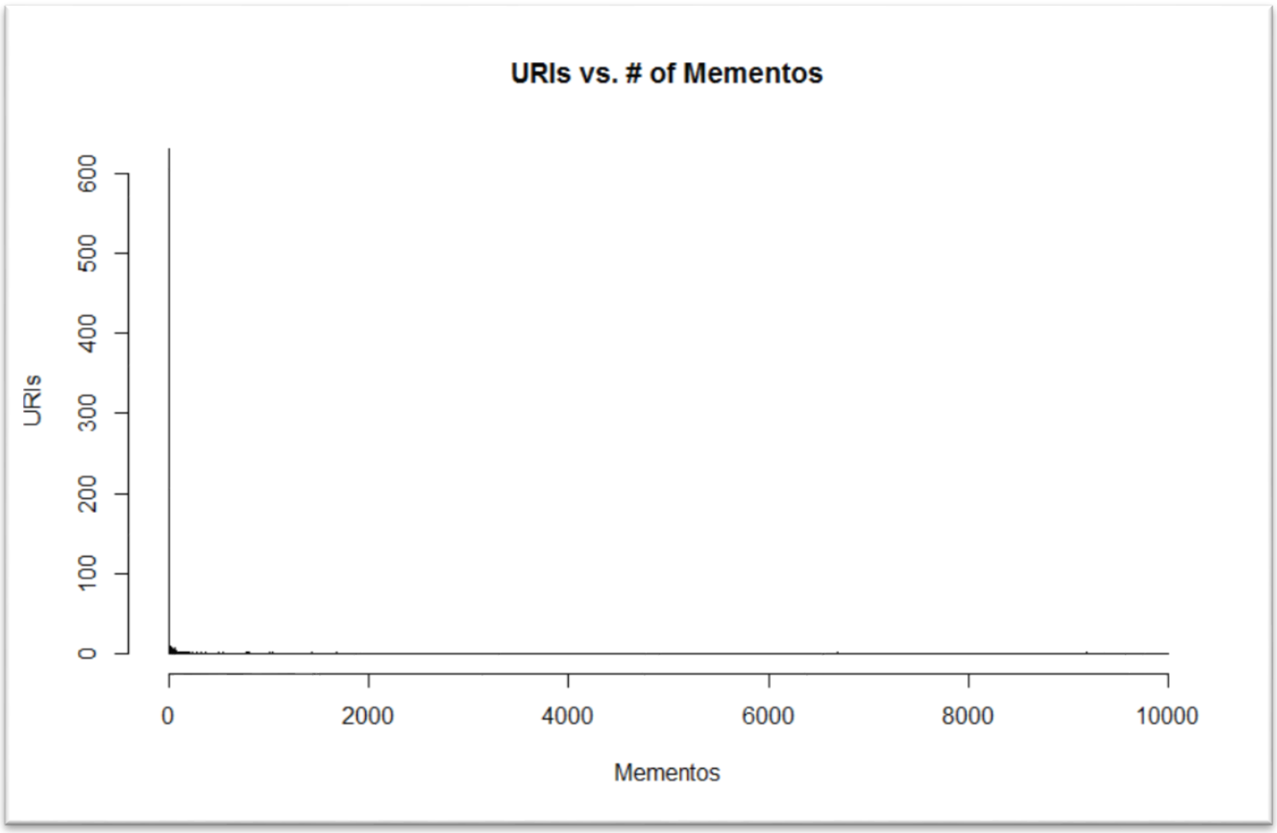


Figure 4 Even after getting rid of the farthest outliter, the histogram was still detail-less

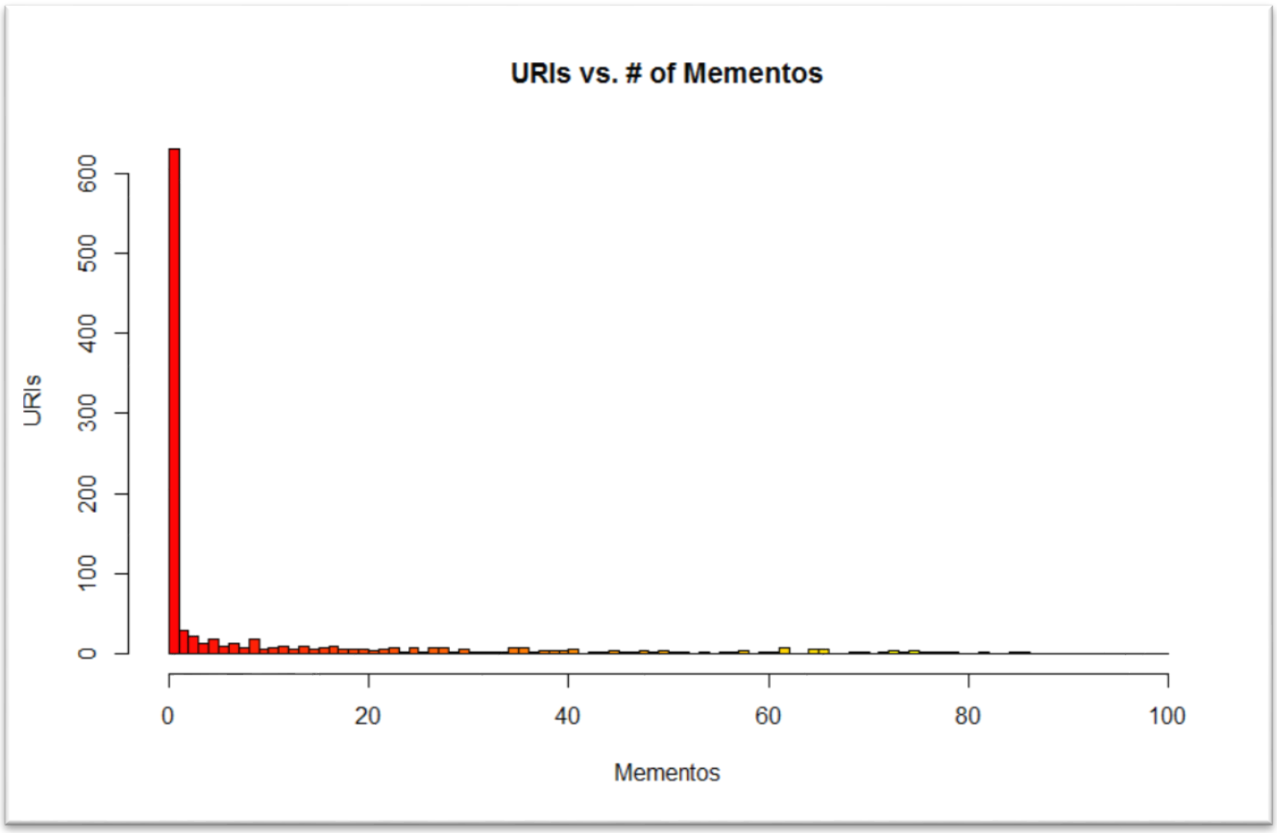


Figure 5 Only after limiting the selection to URIs with <100 mementos, does the histogram become useful. Even with this limitation, this iteration contains about 96% of all the URIs I gathered

Question 3

At this point, I needed a way to query the carbon dating service with the list of links I already had on hand. Instead of writing a way to query the service myself, I chose to use the already working code provided by the makers. By downloading the carbon dating code from their github[5], and after installing the necessary prerequisites, I was able to run the carbon dating service locally using “local.py” and a URI as an argument.

“$ python local.py http://netflix.com/”

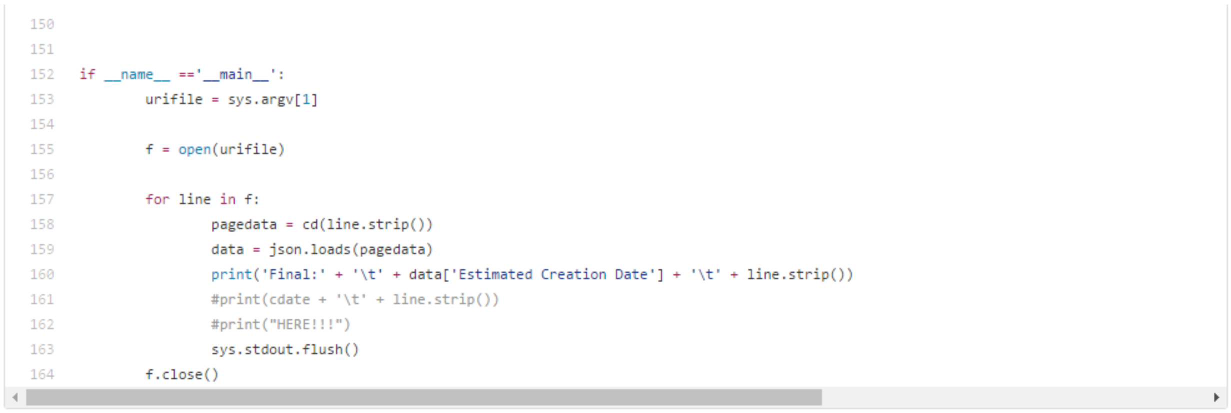
While this worked, I couldn’t just do one URI at a time. So I added my own code to “local.py” that would allow it to take a file as an argument. Only “local.py” was modified, and my code is lines 152 through 164, and that is why “local.py” is included in my repository. The rest is available at the original github repository. This modified “local.py” took as an argument one of the “Linksaa” files used to find mementos. This was to split up the load again. The “>” redirection was used to put the results into a text file.

“$ python local.py Linksaa > TimeMapCountsaa.txt”

Now this was not a perfect solution. The output of the new "local.py" did indeed query every URI in the provided, however the output of the carbon dating service were several lines of JSON like the following:

While this was helpful, the only line I was interested in was "Estimated Creation Date." So in addition to modifiying "local.py" to take a file as an argument, I also added code that would search the resulting large JSON file and return only the "Estimated Creation Date" header. Once found, it would print out an additional life after each query that read:

"Final: <estimated creation date> <uri>"



```python


if __name__ =='__main__':
        urifile = sys.argv[1]

        f = open(urifile)

        for line in f:
                pagedata = cd(line.strip())
                data = json.loads(pagedata)
                print('Final:' + '\t' + data['Estimated Creation Date'] + '\t' + line.strip())
                #print(cdate + '\t' + line.strip())
                #print("HERE!!!")
                sys.stdout.flush()
        f.close()
```

Figure 6 The code I added to local.py in order for it to take a file of URIs to query instead of just a single URI

Now that the giant JSON file was now scattered with these lines, I used the follow-up python program "getDateList.py" to extract ONLY those lines from the file and put them into a file of the same name. They were found using that "Final" flag word I inserted into each line. That is why the final "TimeMapCountaa.txt" files have an entire column reading "Final:" This was fine because I would ignore that column in the following steps.

Now that I had a series of five files with the Estimated Creation Date of each URI, I needed to find out their ages in days. Using the program "getAgeDays.py" I was able to calculate that for each URI. It was in this file that I ignored the "Final" column by using "line.split() function and using tabs [\t] as the signal for a new column. From there, ages in days were calculated using the given time format and the system time. If there was no Estimated Creation Date, they were given zero days. Once more, the ">" redirection was command was used to create the "AgeCountaa.txt" files.

"$ python getAgeDays.py TimeMapCountaa.txt > AgeCountaa.txt"

Just as in question 2, the resulting files were combined into one file: "finalAgeCount.txt"

Finally, in order to create a file that was usable in R, I used the program "makeRListing.py" to combine "finalMementoList.txt" and "finalAgeCount.txt" into one text file, "rMementosVsAge.txt".

"$ python makeRListing.py finalMementoList.txt finalAgeCount.txt > rMementosVsAge.txt"

Similarly to how the AgeCount files were made, I used the line.strip() function in each input file to create columns based on tabs. The final text file now has three columns: URI, number of mementos, and age in days). This also removed any URIs with an age of zero or less days, which reduced the final URI count to 411.

“rMementosVsAge.txt” was used in R to create the final scatterplot.

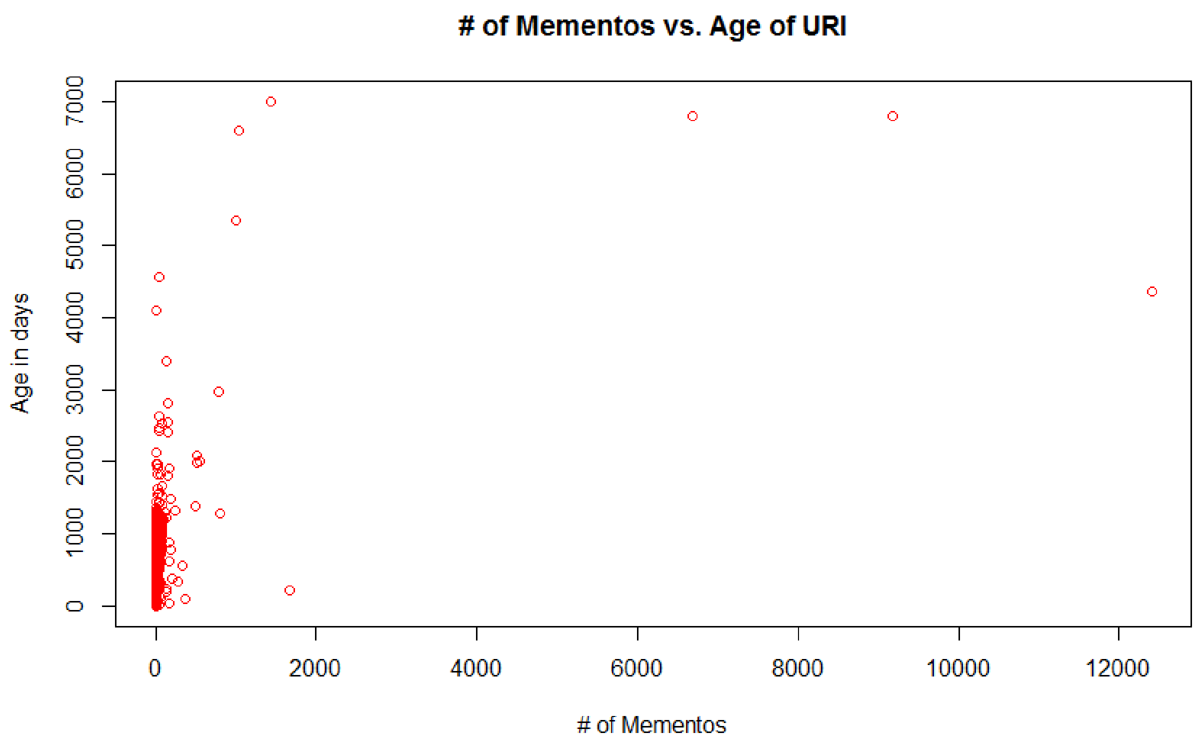


Figure 7 The final scatterplot

References

1.) http://adilmoujahid.com/posts/2014/07/twitter-analytics/
2.) https://thomassileo.name/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/
3.) https://kb.iu.edu/d/afar
4.) http://widequestion.com/question/decode-utf-8-in-python-2-7/
5.) https://github.com/HanySalahEldeen/CarbonDate