Assignment 2

CS 595: Introduction to Web Science

Fall 2013

Shawn M. Jones

Finished on September 26, 2013

1

Question

```
Write a Python program that extracts 1000 unique links from
Twitter.  You might want to take a look at:

http://thomassileo.com/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/

But there are many other similar resources available on the web.  Note
that only Twitter API 1.1 is currently available; version 1 code will
no longer work.

Also note that you need to verify that the final target URI (i.e., the
one that responds with a 200) is unique.  You could have different
shortened URIs for www.cnn.com.

You might want to use the search feature to find URIs, or you can
pull them from the feed of someone famous (e.g., Tim O'Reilly).

Hold on to this collection -- we'll use it later throughout the semester.
```

Answer

Extracting 1000 unique links from Twitter was a more complex endeavor than expected.

It required the use of 4 distinct scripts run in the following order:

1. `gimmetweets.py` <starting tweet id><tweet count per call><# of api calls ><screen name>
    - Starts at a given tweet ID and works backward, printing out all tweets for the given screen name, making only a certain number of api calls to ensure Twitter does not lock out the application.
    - The intention is to run this once for each screen name for which we want to gather tweets, then pass the output to a file via redirection ('`>`')
2. `extractURIsFromTweets.py` <filename #1 containing tweets><filename #2... >
    - Runs through all of the tweets, extracting URIs that match a simple URI pattern of `"http://[\S]*"`
    - Each URI is dereferenced, passing through all redirects. If a status code of 200 comes back, the URI is saved for the next step.
3. `combineLists.py` <filename #1 containing uris><filename #2... >
    - Combine all of the files gathered together in the last step, sort the entries, and eliminate duplicates.
4. `extractRandom1000Links.py` <input file containing links>
    - Because more than 1000 links were acquired, this script randomly generates a representative sample of 1000 links form the file generated in the last step, preventing needless hours processing more than 1000 links in the rest of this assignment.

The 4 scripts allowed resumption of processing if an error state was encountered at any point. For example, `gimmetweets.py` could be cut off because it reached Twitter's hourly request limit. Because it was time consuming (in minutes, sometimes hours) to complete some of these tasks, splitting the scripts up in this way allowed work to be saved across runs. Even if there was some overlap, `combineLists.py` took care of any extra URIs that were processed repeatedly.

Early testing of the *expanded_url* field provided by the Twitter API showed that even though a URI was present in the tweet text, sometimes it did not appear as a value in *expanded_url*, so it was inconsistent. To account for this, all tweets were extracted by `gimmetweets.py` and later processed with `extractURIsFromTweets.py`.

The following screen names were followed for data:

- BarackObama - 44th President of the United States
- BillGates - Former Microsoft CEO, now attempting to rid the world of Malaria, in addition to other feats
- CNN - original 24 hour news channel
- GeorgeTakei - Former helmsman on Star Trek, now Internet celebrity
- MaddowBlog - left leaning Rhodes Scholar political pundit Rachel Maddow
- SethMacFarlane - writer and comedian, best known as the creator of the TV series *Family Guy*
- dailykos - faaaaar left leaning news source
- nprnews - National Public Radio

In retrospect, I could have saved a lot of time with `extractURIsFromTweets.py` by using parallel execution. The program could have broken a given list of tweets up and then executed against each sublist, combining the whole once all were done.

Parallel execution would have saved little for `gimmetweets.py` because of the number of requests limited by Twitter.

The following sections show the source code of each of these scripts.

gimmetweets.py

```
#!/usr/local/bin/python3

# -*- encoding: utf-8 -*-
from __future__ import unicode_literals
import requests
from requests_oauthlib import OAuth1
from urllib.parse import parse_qs
import json
import time
import sys

# ugly, but necessary, globals; saw no need to change this
# strategy from the example
REQUEST_TOKEN_URL = "https://api.twitter.com/oauth/request_token"
AUTHORIZE_URL = "https://api.twitter.com/oauth/authorize?oauth_token="
ACCESS_TOKEN_URL = "https://api.twitter.com/oauth/access_token"

CONSUMER_KEY = "n7jt1uMTwGCcIzDvey8g0A"
CONSUMER_SECRET = "0r6HUrVD36W4MULgWETKMxrQsCICNy1OFFNc2iW4o"

OAUTH_TOKEN = "528649269-SffJ0Rei5PzLYd2NSJPnnm28dP5nlAnt7E1gRGwo"
OAUTH_TOKEN_SECRET = "htrwXF09pS8tP8cMzFrxmMryavdPXd0zPiJHRnLs"

class APIError(Exception):
    """
        If something goes wrong with the API, throw one of these.
        (avoids sys.exit in the middle of the program)
    """

    def __init__(self, value):
        self.value = value

    def __str__(self):
        return repr(self.value)

def setup_oauth():
    """
        Authorize your app via identifier.
        Code inspired by:
        http://thomassileo.com/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/
    """

    # Request token
    oauth = OAuth1(CONSUMER_KEY, client_secret=CONSUMER_SECRET)
    r = requests.post(url=REQUEST_TOKEN_URL, auth=oauth)

    credentials = parse_qs(r.content)

    resource_owner_key = credentials[b'oauth_token'][0].decode(
        encoding='UTF-8')
    resource_owner_secret = credentials[b'oauth_token_secret'
        ][0].decode(encoding='UTF-8')

    # Authorize
    authorize_url = AUTHORIZE_URL + resource_owner_key
    print('Please go here and authorize: ' + authorize_url)

    verifier = input('Please input the verifier: ')
    oauth = OAuth1(CONSUMER_KEY,
                   client_secret=CONSUMER_SECRET,
                   resource_owner_key=resource_owner_key,
                   resource_owner_secret=resource_owner_secret,
                   verifier=verifier)

    # Finally, Obtain the Access Token
    r = requests.post(url=ACCESS_TOKEN_URL, auth=oauth)
    credentials = parse_qs(r.content)
    token = credentials[b'oauth_token'][0].decode(encoding='UTF
        -8')
    secret = credentials[b'oauth_token_secret'][0].decode(
        encoding='UTF-8')

    return token, secret


def get_oauth():
    """
        Code inspired by:
        http://thomassileo.com/blog/2013/01/25/using-twitter-
            rest-api-v1-dot-1-with-python/
    """
    oauth = OAuth1(CONSUMER_KEY,
                client_secret=CONSUMER_SECRET,
                resource_owner_key=OAUTH_TOKEN,
                resource_owner_secret=OAUTH_TOKEN_SECRET)
    return oauth


def call_api(ident, oauth, count, screenName):
    #url = \
    #     "https://api.twitter.com/1.1/statuses/home_timeline.json?max_id=" + \
    #     str(ident) + "&count=" + str(count)
    url = \
        "https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=" + \
        screenName + "&count=" + str(count) + "&max_id=" + str(ident)

    r = requests.get( url, auth=oauth )

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

    return r

def print_tweets_with_ids(
    startingid, oauth, tweetCountPerCall, numberOfCalls, screenName):
    """
        Make the API call, print the tweets returned.
    """

    ident = startingid

    for i in range(0, numberOfCalls):

        response = call_api(ident, oauth, tweetCountPerCall, screenName)

        tweetListSize = len(response.json())

        print(tweetListSize)

        if tweetListSize == 1:
            raise APIError("Ran out of tweets?")

        for i in range(0, tweetListSize):
            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> <tweetCountPerCall> <apiCalls> <
            screenName>")


if __name__ == "__main__":

    #startingid = "400000000000000000"
    try:
        startingId = sys.argv[1]
        tweetCountPerCall = 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()
        print( "OAUTH_TOKEN: " + token )
        print( "OAUTH_TOKEN_SECRET: " + secret )
        print( )
    else:
        oauth = get_oauth()

        try:
            print_tweets_with_ids(
                startingId, oauth, tweetCountPerCall, apiCalls,
                    screenName)
        except APIError as e:
            sys.stderr.write(e.value)
            sys.exit(254)
```

Listing 1: Python program for acquiring tweets for a given screen name

extractURIsFromTweets.py

```
#!/usr/local/bin/python3

import sys
import re
import urllib.request

# for IRI support, thank you evitan for your 0-point answer that
    was helpful:
# http://stackoverflow.com/questions/4389572/how-to-fetch-a-non-
    ascii-url-with-python-urlopen
import httplib2

URLPATTERN = re.compile("http://[\S]*")

def extractURIsFromLine(line):
    """
        Attempts to extract the URIs from the string given.

        Unfortunately, it requires the global URLPATTERN for
            performance.
    """

    return URLPATTERN.findall(line)



def extractRealSupportedURI(uri):
    """
        Returns "real" URI if it survives redirects and returns
            a 200.

        Returns None otherwise.
    """

    realURI = None

    try:
        # this function follows the URI, resolving all redirects
            ,
        # and detects redirect loops
        # iri2uri is needed for IRIs
        request = urllib.request.urlopen(httplib2.iri2uri(uri))

        if request.getcode() == 200:
            realURI = request.geturl()

    except urllib.error.HTTPError as e:
        # something went wrong, we don't care what
        realURI = None

    except urllib.error.URLError as e:
        # something went wrong, we don't care what
        realURI = None

    except UnicodeError as e:
        # something went very wrong with the IRI decoding
        realURI = None

    return realURI

if __name__ == "__main__":

    filenames = sys.argv[1:]

    for filename in filenames:
        f = open(filename)

        for line in f:
            sys.stderr.write("Working on: " + line + '\n')
            uris = extractURIsFromLine(line)

            for uri in uris:
                sys.stderr.write("Trying URI: [" + uri + ']\n')
                goodURI = extractRealSupportedURI(uri)

                if goodURI:
                    print(goodURI)

        f.close()
```

Listing 2: Python program for extracting URIs from a file of tweets

combineLists.py

```
#!/usr/local/bin/python3

import sys

# this file performs the equivalent of cat <filenames> | sort |
    uniq

inputFilenames = sys.argv[1:]

urilist = []

# combine all of the entries from the given files together
for filename in inputFilenames:
    f = open(filename)
    urilist.extend(f.readlines())
    f.close()

# sort the list
urilist.sort()

# expensive, but agreed upon simplest way to "uniq" a list
urilist = list(set(urilist))

# sort the list again
urilist.sort()

# output!
for entry in urilist:
    print(entry.strip())
```

Listing 3: Python program for combining, sorting, and uniquing several files

extractRandom1000Links.py

```python
#!/usr/local/bin/python3

# I realized that I didn't want to spend time waiting for more
    than 1000 links,
# so this script gives a random sample

import random
import sys

inputfile = sys.argv[1]

f = open(inputfile)
links = f.readlines()
f.close()

count = 1000

selectedLinks = []

while count > 0:
    # grab a random entry from the list
    index = random.randint(0, len(links) - 1)
    newlink = links[index].strip()

    if newlink not in selectedLinks:
        selectedLinks.append(newlink)
        count -= 1

for link in sorted(selectedLinks):
    print(link)
```

Listing 4: Python program for randomly extracting 1000 lines from a given file

2

Question

Original:

```
Download the TimeMaps for each of the target URIs.  We'll use the ODU
Memento Aggregator, so for example:

URI-R = http://www.cs.odu.edu/

URI-T = http://mementoproxy.cs.odu.edu/aggr/timemap/link/http://www.cs.odu.edu/

Create a histogram of URIs vs. number of Mementos (as computed from
the TimeMaps).  For example, 100 URIs with 0 Mementos, 300 URIs
with 1 Memento, 400 URIs with 2 Mementos, etc.
```

Answer

The Data Collection

The script `countMementos.py` acquires the TimeMap for each link from a file containing many links, then parses that TimeMap and counts the number of mementos present. Requests that generate a 404 are recorded as having 0 mementos. The mementos are counted using a regular expression.

Because the first run of this script took almost 10 hours to complete, I used the Unix `split` command to divide the 1000 links into 5 files of 200 links each, like so:

```
split -l 200 ../../q1/uris/urilist-final.txt
```

which produced the files `xaa` through `xae`.

I then ran the script against each file in turn. Running these in parallel drastically reduced the run time. In fact, the original 10 hour run was on Sunday afternoon. The second run was on Monday night during dinner, which took 1 hour.

The script `countMementos.py` is run like so:

```
./countMementos.py workspace/xae > xae.txt
```

The source code for doing the dereferencing and processing of each Time Map is shown in Listing 5.

Once the 5 files were acquired, they were combined together using the Unix `cat` command and stored in the file `mementoCounts.txt` like so:

```
cat xaa.txt xab.txt xac.txt xad.txt xae.txt > mementoCounts.txt
```

The Results

The histogram in Figure 1 on page 14 is difficult to read. This histogram was generated in R using buckets of size 1. This was done to keep the y-axis low, because there are 375 URIs with 0 mementos and 244 URIs with 1 memento, totaling more than half of all data recorded.

Outliers also skew the results. Two outliers have 22,157 and 10543 mementos. Stripping out these outliers gets us to Figure 2 on page 15, which shows a slightly better visualization, but there is still such a jump between the low number of mementos to those at 3000 that it is still difficult to see a pattern.

If we focus on the highest number of records, mementos less than 100, the plot actually begins to look like a histogram in Figure 3 on page 16,

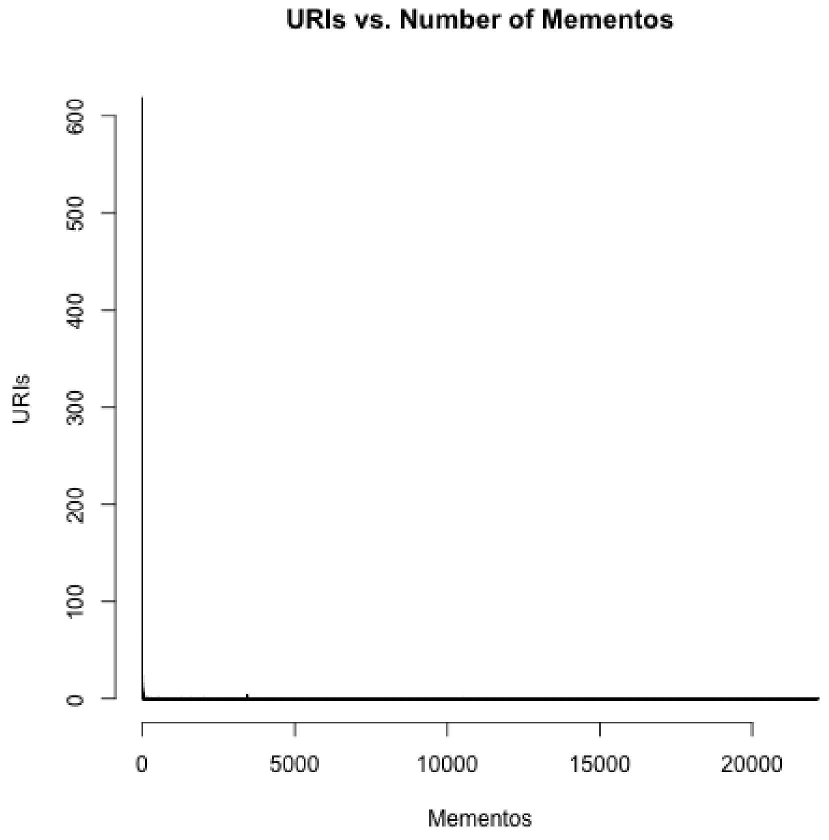

Figure 1: Histogram of URIs vs. number of Mementos for the entire dataset

and we can see the precipitous drop between 0 and 2, and the “Long Tail” decline thereafter.

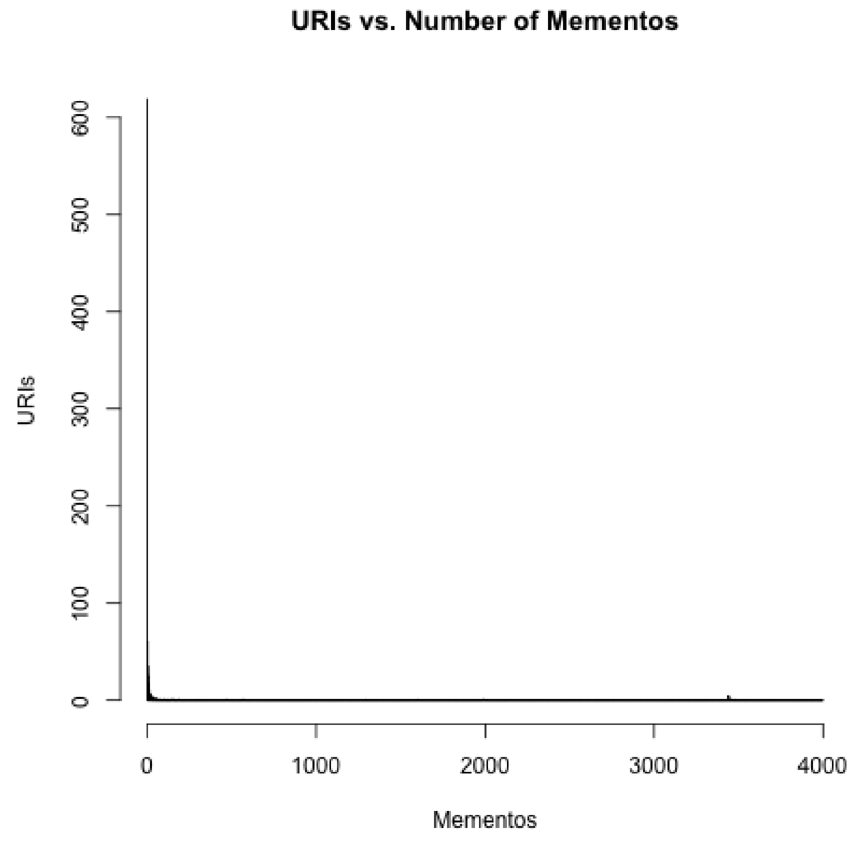

Figure 2: Histogram of URIs vs. number of Mementos for URIs with less than 10,000 Mementos

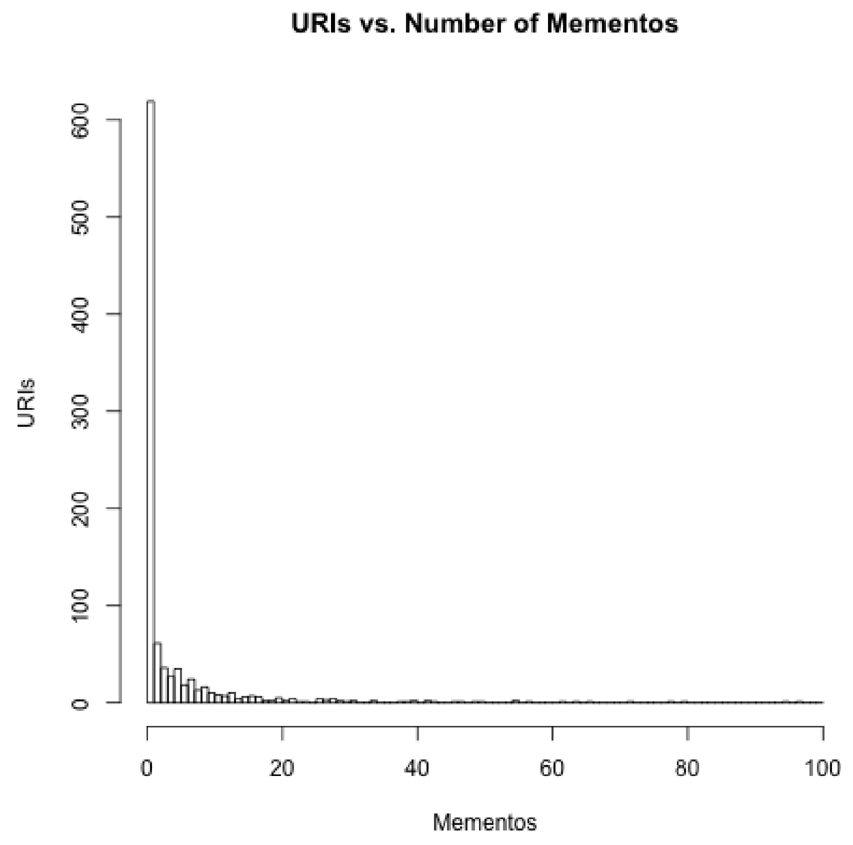

Figure 3: Histogram of URIs vs. number of Mementos for URIs with less than 100 Mementos

```
#!/usr/local/bin/python3

import re
import sys
import urllib.request

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

def getTimeMap(uri):

    urit = "http://mementoproxy.cs.odu.edu/aggr/timemap/link/" +
        uri

    try:
        request = urllib.request.urlopen(urit)

        if request.getcode() == 200:
            timemap = request.readall()
            request.close()
        else:
            timemap = None
            request.close()

    except urllib.error.HTTPError as e:
        timemap = None

    except urllib.error.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)))

    return count


if __name__ == "__main__":
    inputfile = sys.argv[1]

    f = open(inputfile)

    for uri in f:

        mementoCount = countMementos(uri.strip())

        print(str(mementoCount) + "\t" + uri.strip())
        sys.stdout.flush()

    f.close()
```

Listing 5: Python program for processing Time Maps for a given file full of links

3

Question

```
Estimate the age of each of the 1000 URIs using the "Carbon Date" tool:

http://ws-dl.blogspot.com/2013/04/2013-04-19-carbon-dating-web.html

Note: you'll have to download the tool and install; don't try to use the
web service.

For URIs that have > 0 Mementos and an estimated creation date,
create a graph with age (in days) on one axis and number of mementos
on the other.
```

Answer

The Data Collection

Since I did not have Python 2.6 readily available, I modified Carbon Date to work with Python 2.7 and shared it with the author so he could share with others.

From some test runs, it became apparent that the Carbon Date tool takes between 1 and 7 minutes to query all of its services for a given URI. A worst case scenario yields:

$$\frac{7\text{ minutes}}{1\text{ URI}} \times 1000\text{ URIs} = 7000\text{ minutes} \times \frac{1\text{ day}}{1440\text{ minutes}} \approx 5\text{ days}$$

I modified the Carbon Date tool to accept a configuration file as an argument (also sharing with the author), then ran it 5 times across 5 different screen sessions using 5 different configuration files containing 5 different port numbers.

Then I did the same for the Carbon Date tool client shown in Listing 6.

Here's an example run of the Carbon Date client using a subset of the 1000 links in a file named `xae`.

```
./queryCarbonDateFor1000Links.py http://ganesh:8547/cd xae > xae
    .data
```

Getting the dates was not enough, of course, because we need the ages in days. Listing 7 takes in the list from the previous script and generates a tab-delimited file containing the URIs and ages in days. It is run like so:

```
/listAgesFromCdlist.py work/cdlist-final.txt > daycounts-final.
    txt
```

What we really want is to create a graph for all URIs that have >0 Mementos and an estimated creation date. This is where 8 comes in. This script joins the tab-delimited file generated from Question 2 with the tab-delimited file containing our ages in days, eliminates those with 0 Mementos and no carbon dates, then generates another tab-delimited file that can be fed into R for our scatter plot.

This script is run like so:

```
./joinAndProcessAgesWithMementoData.py ../q2/mementoCounts.txt
    daycounts-final.txt > mementosVsAge.txt
```

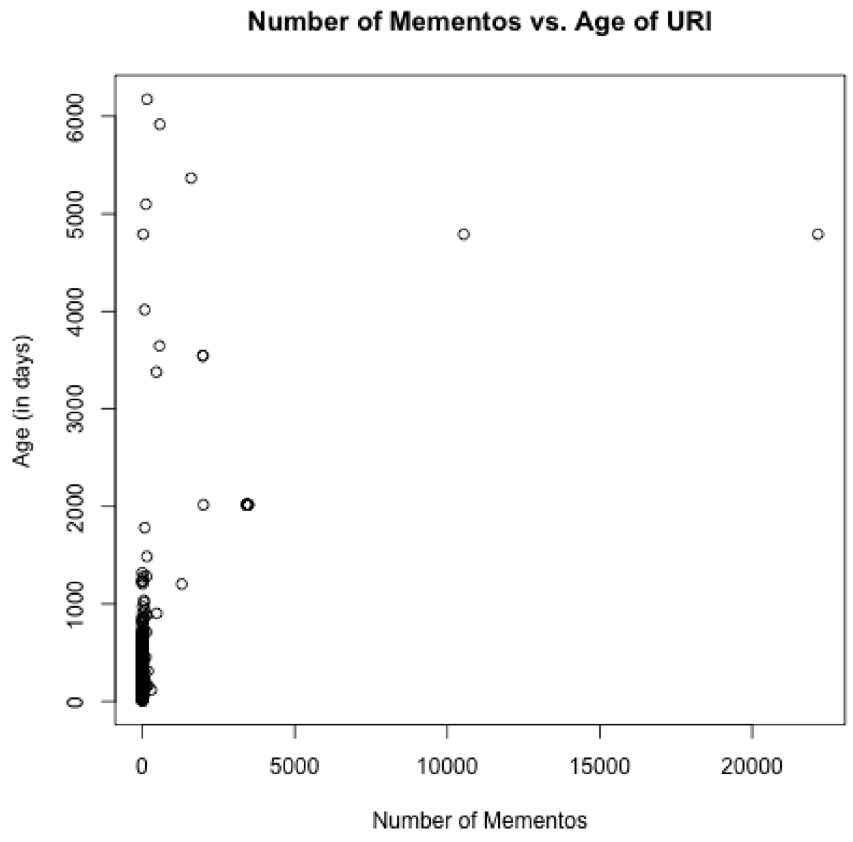

Figure 4: Number of Mementos vs. Age of URI

The Results

Because, as shown in Figure 3, the majority of the memento counts are between 0 and 2, most of the data points in Figure 4 are on the left side, with a large number of mementos being roughly less than 1200 days old. There are also two outliers, our friends with 22,157 and 10543 mementos, who seem to be much older. In a perfect world, the whole graph would follow the pattern of these two outliers, with the number of mementos increasing from the age of first creation.

Then again, because the majority of the URIs are "young", and it takes some time to get archived, this graph may make sense, especially for twitter postings.

```
#!/usr/local/bin/python3

import sys
import urllib.request
import json

def queryCarbonDate(cduri, uri):

    if cduri[-1] != '/':
        cduri += '/'

    sys.stderr.write("Using cduri = " + cduri + "\n")
    sys.stderr.write("Requesting " + cduri + uri + "\n")
    sys.stderr.flush()

    request = urllib.request.urlopen(cduri + uri)
    pagedata = request.readall().decode('utf-8')
    request.close()

    data = json.loads(pagedata)

    return data['Estimated Creation Date']

if __name__ == '__main__':
    cduri = sys.argv[1]
    urifile = sys.argv[2]

    f = open(urifile)

    for line in f:
        cdate = queryCarbonDate(cduri, line.strip())
        print(cdate + "\t" + line.strip())
        sys.stdout.flush()

    f.close()
```
Listing 6: Python program for remotely querying the Carbon Date tool

```
#!/usr/local/bin/python3

import sys
import time
import datetime

cdlistfile = sys.argv[1]

f = open(cdlistfile)

for line in f:
    try:
        line = line.strip()
        (cdate, uri) = line.split('\t')
        ct = time.strptime(cdate, "%Y-%m-%dT%H:%M:%S")
        # Thanks http://stackoverflow.com/questions/1697815/how-
            do-you-convert-a-python-time-struct-time-object-into-
            a-datetime-object
        cdt = datetime.datetime.fromtimestamp(time.mktime(ct))
        now = datetime.datetime.now()
        days = (now - cdt).days
        print(str(days) + '\t' + uri)
    except ValueError:
        # skip over those items without carbon dates
        pass

f.close()
```

Listing 7: Python program for calculating the ages based on dates gathered from the Carbon Date tool

```python
#!/usr/local/bin/python3

import sys

mementoDataFile = sys.argv[1]
ageDataFile = sys.argv[2]

mementoData = {}
ageData = {}

f = open(mementoDataFile)

for line in f:
    line = line.strip()
    (mementoCount, uri) = line.split('\t')

    if int(mementoCount) > 0:
        mementoData[uri] = mementoCount

f.close()

f = open(ageDataFile)

for line in f:
    line = line.strip()
    (age, uri) = line.split('\t')

    ageData[uri] = age

f.close()

for key in mementoData:

    print(key + '\t' + mementoData[key] + '\t' + ageData[key])
```

Listing 8: Python program for calculating the ages based on dates gathered from the Carbon Date tool