Assignment 2

Introduction to Web Science
Dr. Michael Nelson
Fall 2014

Sybil Melton

September 27, 2014

Contents

List of Figures

Listings

1 Question

Write a Python program that extracts 1000 unique links from Twitter.

1.1 The Answer

1.1.1 Authentication

I used the recommended reading to set up the Twitter oauth. Once it was set up, I only needed to keep using the "get_oauth" function.[1] The function is in Listing 1.

```
def get_oauth():
  oauth = OAuth1(CONSUMER_KEY,
        client_secret=CONSUMER_SECRET,
        resource_owner_key=OAUTH_TOKEN,
        resource_owner_secret=OAUTH_TOKEN_SECRET)
  return oauth
```

Listing 1: Oath Function

1.1.2 Searching

The Twitter API documentation was the best reference to get the correct syntax to search tweets. Using python urllib, I hard-coded a count of 100 into my queries to get the maximum number of results returned.[2][3][5] I am new to Twitter, so I don't follow any famous people who post a lot of URL's, but the search functionality seemed to work ok. The returned JSON data was encoded into Python in order to make it searchable.[4] The function is in Listing 2.

```
def search(query_arg):
  params = urllib.urlencode({'q': query_arg, 'count': 100}) #https://docs.python.org/2/library
      /urllib.html
  urli = "https://api.twitter.com/1.1/search/tweets.json"
  full_url = urli+"?"+params
  r = requests.get(url=full_url, auth=oauth)
  data = json.loads(r.text) #http://www.pythonforbeginners.com/python-on-the-web/parsingjson/
  slist = list(data[u'statuses'])
  parse_Lists(slist)
```

Listing 2: Search Function

1.1.3 Parsing Data

Twitter stores URL's in an arrays within the Entities Object. I retrieved the expanded URL, because it is the "real" URL, not the shorted version in the tweet.[6]. All the URI's were stored in a file called "urls.txt." PyCurl was a useful python module, because it includes the capability to write to a byte stream or file location. Additionally, it will follow redirects and the information returned was the final URI.[8] Once the response was returned, urlparse allowed me to take the location and path from the URL and write it to file.[7] The function is in Listing 3.

```
def parse_Lists(slist):
  ulist = list()
  u = ""
  for i in range(len(slist)):
    if u'url' in slist[i][u'user'][u'entities'].keys():
      u = slist[i][u'user'][u'entities'][u'url'][u'urls'][0][u'expanded_url']
      u = u.encode('ascii')
      ulist.append(u)
```

```
    else:
      continue
  for i in range(len(slist)):
    if not slist[i][u'entities'][u'urls']:
      continue
    else:
      u = slist[i][u'entities'][u'urls'][0][u'expanded_url']
      u = u.encode('ascii')
      ulist.append(u)
  for i in range(len(ulist)):
    ufile = open('urls.txt', 'a',0)
    buffer = BytesIO()
    c = pycurl.Curl()
    c.setopt(c.URL, ulist[i])
    c.setopt(c.WRITEDATA, buffer)
    c.setopt(c.FOLLOWLOCATION, True)
    try: #http://www.angryobjects.com/2011/10/15/http-with-python-pycurl-by-example/
      c.perform()
      if c.getinfo(c.RESPONSE_CODE) == 200:
        o = urlparse(c.getinfo(c.EFFECTIVE_URL))
        loc = o.netloc + o.path
        ch = check(loc)
        if not ch:
          ufile.write(loc + '\n')
          ufile.close()
        else:
          continue
      c.close()
    except pycurl.error, error:
      errno, errstr = error
      print 'An error occurred: ', errstr
```

Listing 3: Parse Function

1.1.4 Uniqueness

In order to verify URI uniqueness, I used an re search function. The function is in Listing 4.

```
def check(url): #http://stackoverflow.com/questions/16432203/python-checking-if-string-is-in-a-text-file @Ashwini Chaudhary
  with open("urls.txt") as f:
    found = False
    for line in f: #iterate over the file one line at a time(memory efficient)
      if re.search(url, line):    #if string found is in current line then print it
        found = True
  return found
```

Listing 4: Check Function

1.1.5 Completeness

A function was used to count the number of URI's in the file after each search, until I had more than 1000. Upon reaching this, I looked through the file manually to ensure each was unique.[9] The function is in Listing 6.

```
def check(url): #http://stackoverflow.com/questions/16432203/python-checking-if-string-is-in-a-text-file @Ashwini Chaudhary
  with open("urls.txt") as f:
    found = False
    for line in f: #iterate over the file one line at a time(memory efficient)
      if re.search(url, line):    #if string found is in current line then print it
        found = True
  return found
```

Listing 5: Count Function

1.1.6 Running

The functions were imported and run via the Python console. First, I verified functionality by running the search function and verifying the results were written to file. Then I created small lists The commands used are in Listing ??.

```
search('#newmusictuesday')
search('lightning')
search('wolfknives')
search('festival')
count_URL()

tlist = ('ghost', 'wolf', 'drawing')
for t in tlist:
  search(t)
  print "done"
  count_URL()
```

Listing 6: Program Run Commands

2 Question

Download the TimeMaps for each of the target URIs. 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.

2.1 The Answer

2.1.1 Downloading TimeMaps

I chose to use the mementoweb.org Aggregator. I used PyCurl again, to retrieve the data and wrote it to a file. The pages were sent in multiple's of 1000, so a counter was used to keep track of the number of records retrieved. After the subtraction of the non-record lines, if the number of lines was a multiple of 1000, the next page was attempted to be retrieved. There was a problem at first with the entire buffer not being written to the file, but that was corrected with the '0' option, to write everything directly to the file.[10] The number of lines were added up and the results were written to "hist.txt,"" with each URI to make the histograph. The program is in Listing 7.

```
import pycurl
from urlparse import urlparse

def count_lines():
  with open('out.html') as f:
    count = sum(1 for _ in f)
  return count

def getTimeMap():
  with open("urls.txt") as f:
    for line in f:
      urli = "http://mementoweb.org/timemap/link/http://"
      full_url = urli+line
      #print full_url
      with open('out.html', 'wb', 0) as g: # http://stackoverflow.com/questions/3167494/how-
          often-does-python-flush-to-a-file
        c = pycurl.Curl()
        c.setopt(c.URL, full_url)
        c.setopt(c.WRITEDATA, g)
        c.perform()
        count = count_lines() -3 #get the number of lines in the timemap
        temp = 0
        i = 3
```

```
23          while count % 1000 == 0:
24            temp = temp + count
25            urlt = '%s%d%s' %("http://mementoweb.org/timemap/link/", temp+1, "/http://"+line)
26            #print urlt
27            c = pycurl.Curl()
28            c.setopt(c.URL, urlt)
29            c.setopt(c.WRITEDATA, g)
30            c.perform()
31            i += 2
32            count = count_lines() -i
33            count = count + temp
34          if count == -2:
35            count = 0
36          outfile = open('hist.txt', 'a', 0)
37          outfile.write('%d' % count+"\t"+line)
38          print line
39          c.close()
40          outfile.close()
41
42  getTimeMap()
```

Listing 7: TimeMap Program

2.1.2 Results

I found extremely one-sided results. There are 680 URI's with 0 or 1 memento only. Over 90% had less than 1000 mementos, as Figure 1 illustrates. Subsets of the data were graphed, to show the distributions at different intervals. Each shows a tailed graph, so there are significantly lower number of URI's with a large number of mementos. Figure 2 shows the URI's between zero and ten mementos. This was the second highest distribution. Figure 3 is the URI's between ten and 100. Figure 4 continues with 100 to 1000.

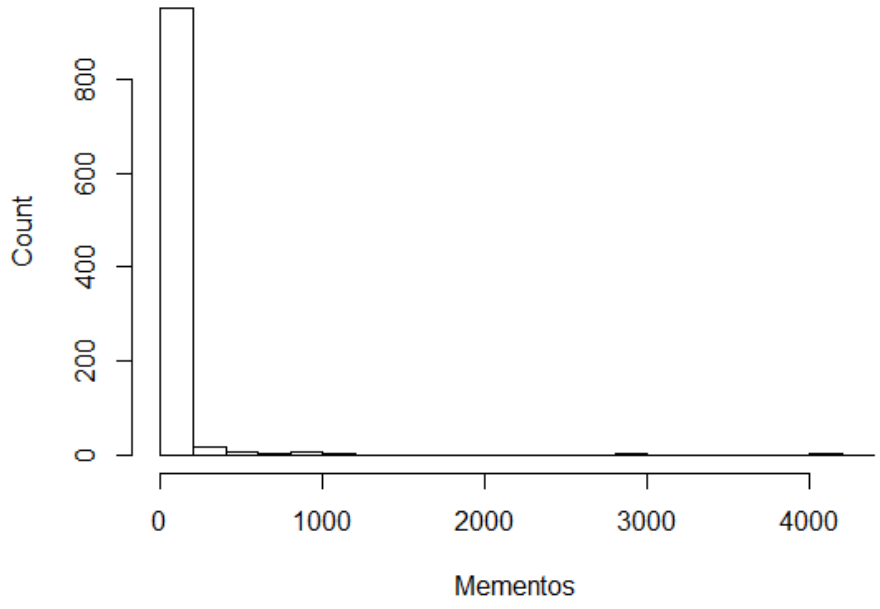

Figure 1: URI vs Mementos

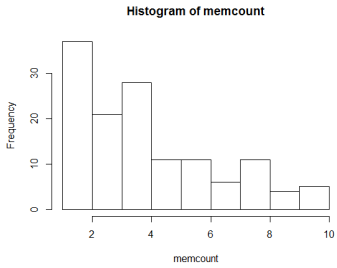

Figure 2: URI's with 0 -10 Mementos

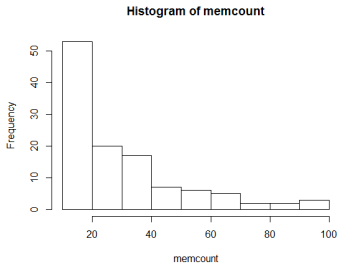

Figure 3: URI's with 10 - 100 Mementos

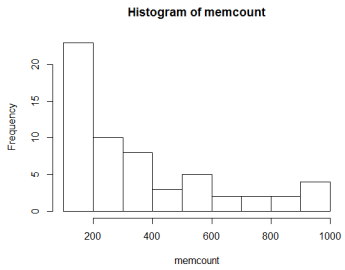

Figure 4: URI's with 100 - 1000 Mementos

3 Question

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

3.1 The Answer

3.1.1 CarbonDate

I installed the CarbonDate server on a local virtual machine. After processing the text result, the estimated created date was changed to datetime format in order to calculate the age in days.[11] The age was saved into a text file with the URI. The program is in Listing 8.

```
1
2  import pycurl
3  from datetime import date
4  import datetime
5  import urlparse
6
7  try:
8      from io import BytesIO
9  except ImportError:
10     from StringIO import StringIO as BytesIO
11
12 def getCarbonDate():
13   with open("urls.txt") as f:
14     for line in f:
15       urli = "http://192.168.1.100:8080/cd?url=http://"
16       full_url = urli+line
17       #print full_url
18       buffer = BytesIO()
19       c = pycurl.Curl()
20       c.setopt(c.URL, full_url)
21       c.setopt(c.WRITEDATA, buffer)
```

```
22        c.perform()
23        contents = buffer.getvalue()
24        b = contents.split("\n")
25        d = b[2].strip(" ")
26        d = d.split("\":")
27        if d[1] != u' "",':
28          created = datetime.datetime.strptime(d[1], ' "%Y-%m-%dT%H:%M:%S",') #https://docs.
                python.org/2/library/datetime.html
29          created = created.date
30          print created()
31          today = date.today()
32          print today
33          time_to_created = abs(created() - today)
34          print time_to_created
35          outfile = open("dates.txt", 'a', 0)
36          outfile.write('%d' % time_to_created.days+"\t"+line)
37        else:
38          outfile = open("dates.txt", 'a', 0)
39          outfile.write("UNK\t"+line)
40        c.close()
41        outfile.close()
42
43  getCarbonDate()
```

Listing 8: CarbonDate Program

The age data was merged with the memento data and a scatterplot was created. My collection of URI's does not show a strong correlation between number of mementos and age. As the graph in figure 5 shows, there is a higher concentration when the number of mementos are small and age is short. But there are still are a large number of URI's witih a small mementos and were created quite a long time ago.

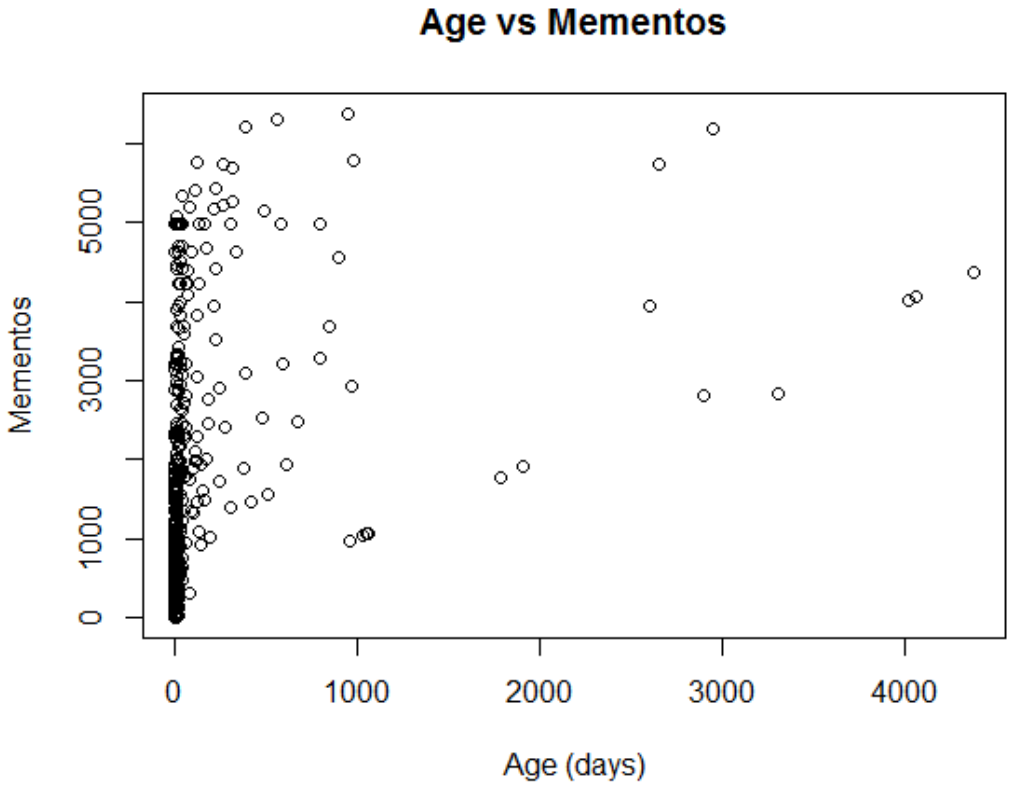

Figure 5: Age vs Mementos

References

[1] Using Twitter REST API v1.1 with Python, Retrieved September 20, 2014 from http://thomassileo.com/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/

[2] GET search/tweets, Retrieved September 19, 2014 from https://dev.twitter.com/rest/reference/get/search/tweets

[3] Open arbitrary resources by URL, Retrieved September 19, 2014 from https://docs.python.org/2/library/urllib.html

[4] Parsing JSON in Python, Retrieved September 19, 2014 from http://www.pythonforbeginners.com/python-on-the-web/parsingjson/

[5] The Search API, Retrieved September 19, 2014 from https://dev.twitter.com/rest/public/search

[6] Entities, Retrieved September 19, 2014 from https://dev.twitter.com/overview/api/entities

[7] objectified HTTP with Python – PycURL by Example, Retrieved September 19, 2014 from http://www.angryobjects.com/2011/10/15/http-with-python-pycurl-by-example/

[8] PycURL Quick Start, Retrieved September 19, 2014 from http://pycurl.sourceforge.net/doc/quickstart.html

[9] Ashwini Chaudhary, Python: Checking if string is in a text file, Retrieved September 19, 2014 from http://stackoverflow.com/questions/16432203/python-checking-if-string-is-in-a-text-file

[10] Corey Goldberg, How often does python flush to a file?, Retrieved September 20, 2014 from http://stackoverflow.com/questions/3167494/how-often-does-python-flush-to-a-file

[11] Basic date and time types, Retrieved September 21, 2014 from https://docs.python.org/2/library/datetime.html