

Assignment 1

Introduction to Web Science

Dr. Michael Nelson

Fall 2014

Sybil Melton

September 11, 2014

Contents

1	Question	1
1.1	The Answer	1
1.1.1	Authentication	1
1.1.2	Searching	1
1.1.3	Parsing Data	1
1.1.4	Uniqueness	2
1.1.5	Completeness	2
1.1.6	Running	3
2	Question	3
2.1	The Answer	3
2.1.1	Downloading TimeMaps	3
2.1.2	Results	4
3	Question	5
3.1	The Answer	5
3.1.1	CarbonDate	5

List of Figures

1	URI vs Mementos	4
2	URI's with 0 -10 Mementos	5
3	URI's with 10 - 100 Mementos	5
4	URI's with 100 - 1000 Mementos	5
5	Age vs Mementos	6

Listings

1	Oath Function	1
2	Search Function	1
3	Parse Function	1
4	Check Function	2
5	Count Function	2
6	Program Run Commands	3
7	TimeMap Program	3
8	CarbonDate Program	5

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.

```
1 def get_oauth():
2     oauth = OAuth1(CONSUMER_KEY,
3                   client_secret=CONSUMER_SECRET,
4                   resource_owner_key=OAUTH_TOKEN,
5                   resource_owner_secret=OAUTH_TOKEN_SECRET)
6     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.

```
1 def search(query_arg):
2     params = urllib.urlencode({'q': query_arg, 'count': 100}) #https://docs.python.org/2/library/urllib.html
3     url = "https://api.twitter.com/1.1/search/tweets.json"
4     full_url = url+"?" + params
5     r = requests.get(url=full_url, auth=oauth)
6     data = json.loads(r.text) #http://www.pythonforbeginners.com/python-on-the-web/parsingjson/
7     slist = list(data[u'statuses'])
8     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.

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

```

9     else:
10         continue
11 for i in range(len(slist)):
12     if not slist[i][u'entities'][u'urls']:
13         continue
14     else:
15         u = slist[i][u'entities'][u'urls'][0][u'expanded_url']
16         u = u.encode('ascii')
17         ulist.append(u)
18 for i in range(len(ulist)):
19     ufile = open('urls.txt', 'a',0)
20     buffer = BytesIO()
21     c = pycurl.Curl()
22     c.setopt(c.URL, ulist[i])
23     c.setopt(c.WRITEDATA, buffer)
24     c.setopt(c.FOLLOWLOCATION, True)
25     try: #http://www.angryobjects.com/2011/10/15/http-with-python-pycurl-by-example/
26         c.perform()
27         if c.getinfo(c.RESPONSE_CODE) == 200:
28             o = urlparse(c.getinfo(c.EFFECTIVE_URL))
29             loc = o.netloc + o.path
30             ch = check(loc)
31             if not ch:
32                 ufile.write(loc + '\n')
33                 ufile.close()
34             else:
35                 continue
36         c.close()
37     except pycurl.error, error:
38         errno, errstr = error
39         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.

```

1 def check(url): #http://stackoverflow.com/questions/16432203/python-checking-if-string-is-in-a
   -text-file @Ashwini Chaudhary
2     with open("urls.txt") as f:
3         found = False
4         for line in f: #iterate over the file one line at a time(memory efficient)
5             if re.search(url, line): #if string found is in current line then print it
6                 found = True
7     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.

```

1 def check(url): #http://stackoverflow.com/questions/16432203/python-checking-if-string-is-in-a
   -text-file @Ashwini Chaudhary
2     with open("urls.txt") as f:
3         found = False
4         for line in f: #iterate over the file one line at a time(memory efficient)
5             if re.search(url, line): #if string found is in current line then print it
6                 found = True
7     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 ??.

```
1 search('#newmusicuesday')
2 search('lightning')
3 search('wolfknives')
4 search('festival')
5 count_URL()
6
7 tlist = ('ghost', 'wolf', 'drawing')
8 for t in tlist:
9     search(t)
10    print "done"
11    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 histogram. The program is in Listing 7.

```
1 import pycurl
2 from urlparse import urlparse
3
4 def count_lines():
5     with open('out.html') as f:
6         count = sum(1 for _ in f)
7     return count
8
9 def getTimeMap():
10    with open("urls.txt") as f:
11        for line in f:
12            urli = "http://mementoweb.org/timemap/link/http://"
13            full_url = urli+line
14            #print full_url
15            with open('out.html', 'wb', 0) as g: # http://stackoverflow.com/questions/3167494/how-often-does-python-flush-to-a-file
16                c = pycurl.Curl()
17                c.setopt(c.URL, full_url)
18                c.setopt(c.WRITEDATA, g)
19                c.perform()
20                count = count_lines() -3 #get the number of lines in the timemap
21                temp = 0
22                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.

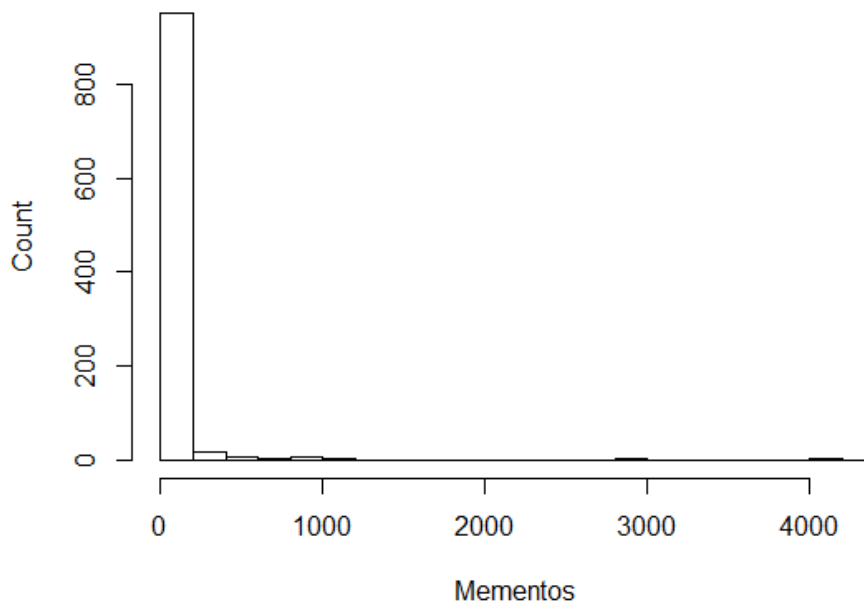


Figure 1: URI vs Mementos

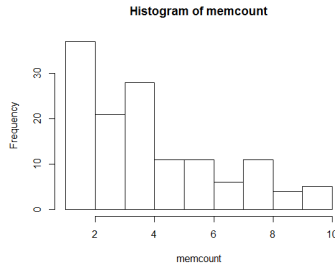


Figure 2: URI's with 0 -10 Mementos



Figure 3: URI's with 10 - 100 Mementos

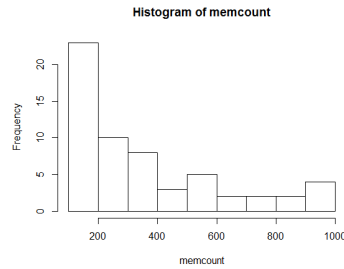


Figure 4: URI's with 100 - 1000 Mementos

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             url = "http://192.168.1.100:8080/cd?url=http://"
16             full_url = url+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("\n:")
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 with a small mementos and were created quite a long time ago.

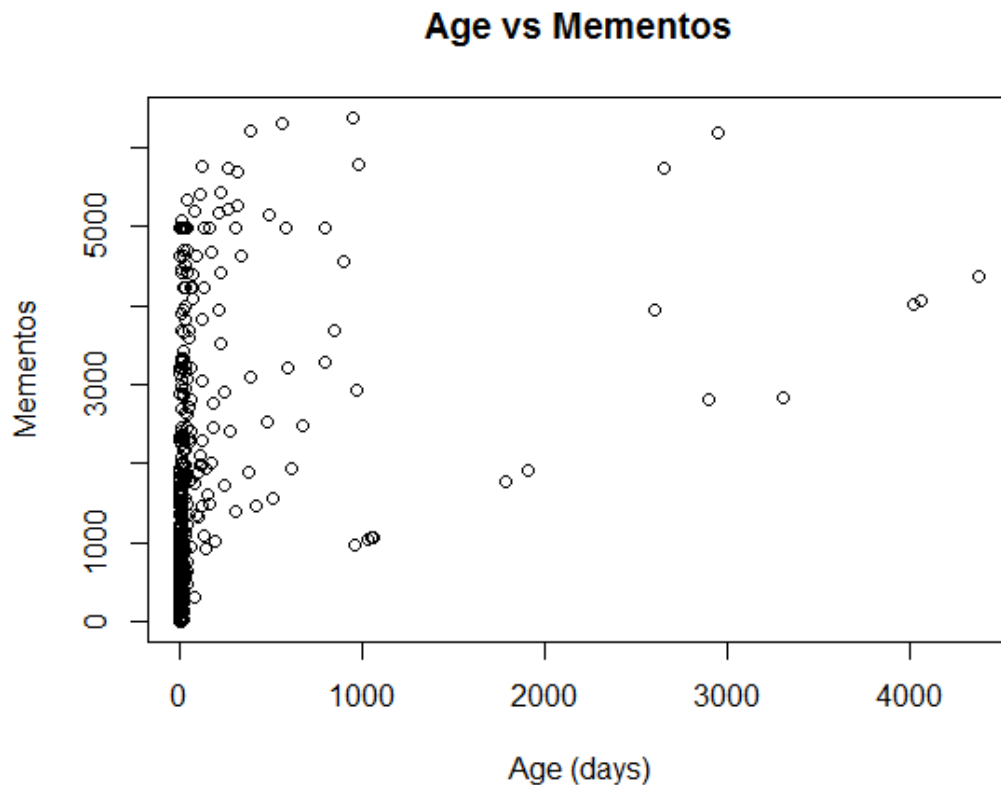


Figure 5: Age vs Mementos

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>