

Web Science cs532-s16

ASSIGNMENT 2 REPORT

BY: HUAN HUANG

02/11/2016

Problem 1

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 many different shortened URIs for www.cnn.com (t.co, bit.ly, goo.gl, etc.).

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

For this problem, I created 3 programs that handled the problem at each stage. But before I can start writing the programs, I had to register a new application with my twitter account. Once that is approved, I received my keys and tokens, with those keys and tokens I can access twitter's data through its API. So, the purpose of my first program is to use my keys and tokens to access twitter's data and listen for any tweets that contain the subject or phrase I set in my filter. originally, I intended to use Tweepy for this job, then, I came across the another module called Python Twitter Tools which made this part of assignment much simpler. When I setup the API object for tweet streaming, it automatically assign the different parts of the tweet into entities, all I have to do now is call for the entity urls to bring up the links in every tweet.

```
1 #!/usr/bin/python
2
3 from twitter import *
4
5 savefile = open('initialurls.txt', 'a')
6
7 # load my API credentials
8
9 accesstoken = "110800099-ZiXf6imxd24OVqw5nnm4nkHXmVwhX46QQI0Wb3Zt"
10 accesstokensecret = "FIDFz2OI035FONaiL9uTzfqRauVArVwwVMuxiHDWQsmwo"
11 consumerkey = "1N9TkDHB9g4UIbjVj5j6idqnn"
12 consumersecret = "5Yjw1hRViWd5TloKLmEibPoregSltxO1bY7YJckjnecuSw4n9B"
13
14 # create twitter API object
15
```

```
16 auth = OAuth(accesstoken, accesstokensecret, consumerkey, consumersecret)
17 stream = TwitterStream(auth = auth, secure = True)
18
19 # set filter credentials
20 tweet_iter = stream.statuses.filter(track='sports, game')
21
22 for tweet in tweet_iter:
23
24 # print URLs found inside
25     try:
26         for url in tweet["entities"]["urls"]:
27
28             print "%s" % url["expanded_url"]
29             savefile.write("%s" % url["expanded_url"])
30             savefile.write('\n')
31
32
33 except (KeyError, UnicodeEncodeError) as e:
34     pass
```

I let the program ran for about 30 minutes and collected 3896 links, which I saved in a text file called initialurls.txt(every file I mention will be provided in github).

```
3883 http://www.nick.com/kids-choice-awards/vote/favorite-video-game/
3884 http://goo.gl/PKD66U
3885 https://twitter.com/KingFavre/status/697149646003691520
3886 http://apple.co/1NSZVYM
3887 http://bit.ly/SmurfApp
3888 https://www.google.com/url?rct=j&sa=t&url=http://www.sentinelassam.com/sports/story.php%3F
3889 http://es.pn/1Rmg1vW
3890 http://snpy.tv/1XgH0tB
3891 http://wp.me/p78QgF-6qx
3892 http://youtu.be/hiXVaec90aY?a
3893 http://m.mynews.ly/!EH.Db3QD
3894 https://twitter.com/Karabo_Skhuku/status/697130453170569217
3895 http://hnhh.co/sehzyr
3896 http://goo.gl/fb/pzGWxt
3897
```

Figure 1: Sample of part of the initialurls.txt

Many of these URLs are short links or redirects, to handle that I wrote another program called `urlredirect`. I actually have two version of this program that does almost the same thing. The other version of this program uses `request.get` instead of `HTTPRedirectHandler`. But it operates much slower and sometimes stops at a URL without giving me any error or return control back to me nor was it in an infinite loop. It just stops there and this issue only happens sometimes. I could not figure out why and, therefore, given up on it.

```
1 import urllib2
2 import errno
3 import socket
4 import httplib
5 import ssl
6 import sys
7 import requests
8 from requests.exceptions import HTTPError
9
10 gett = open('initialurls.txt', 'r')
11 savefile = open('urlsRound2.txt', 'a')
12
13 s = socket.socket()
14 s.settimeout(5)
15
16 def getHeaders(urls):
17     try:
18         #conn = urllib2.urlopen(urls)
19         opener = urllib2.build_opener(urllib2.HTTPRedirectHandler)
20         request = opener.open(urls)
21
22         print request.url
23         savefile.write(request.url)
24         savefile.write('\n')
25         #print request.code
26
27     except urllib2.HTTPError, x:
28         print 'Ignoring ', x.code
29     except urllib2.URLError, e:
30         print 'Ignoring ', e.args
31     except ssl.CertificateError, s:
32         print 'Ignoring ', s
33     except socket.error, s:
34         print 'Ignoring %s' % s
35     except httplib.BadStatusLine, b:
36         pass
37
38 for links in gett:
39     urls = links
40     getHeaders(urls);
```

The result is saved in a file called `urlsRound2.txt`.

```

3677 https://twitter.com/Killager/status/697150543521681408
3678 http://www.nick.com/kids-choice-awards/vote/favorite-video-game/
3679 http://www.pnixgames.com/BowlingKing/version.htm
3680 https://twitter.com/KingFavre/status/697149646003691520
3681 https://itunes.apple.com/app/apple-store/id1035238692?mt=8
3682 https://itunes.apple.com/us/app/smurfs-village/id399648212?mt=8
3683 https://www.google.com/url?rct=j&sa=t&url=http://www.sentinelassam.com/sports/story.php%3Fse
3684 http://espn.go.com/video/clip?id=14745366&ex_cid=sportscenterTW&sf20622421=1
3685 https://twitter.com/btsportfootball/status/697151976660807681
3686 http://profootballfanzone.com/2016/02/oakland-raiders-owner-not-concerned-about-lost-home-game/
3687 https://www.youtube.com/watch?v=hiXVaec90aY&feature=youtu.be&a
3688 http://www.appy-gamer.com/Web/ArticleWeb.aspx?regionid=1&articleid=57635843
3689 https://twitter.com/Karabo_Skhuks/status/697130453170569217
3690 http://www.hotnewhiphop.com/honorable-c-note-7-days-feat-peewee-longway-2-chainz-and-zappsola-new-s
3691 http://news.okezone.com/read/2016/02/10/340/1308294/asyik-main-game-di-warnet-palaku-begal-diringku
3692

```

Figure 2: Sample of Part of urlround2.txt

The links I have now are all final links that would return a 200 response code, but there are too many duplicates and I only need 1000 unique links. Therefore, I created another program called uniqueurls.py to give me 1000 unique links. It just use *set* function to remove duplicate strings and remove all lines after line 1000.

```

1
2 gett = open('urlsRound2.txt', 'r')
3 savefile = open('uniqueurls.txt', 'a')
4 savefile2 = open('requiredurls.txt', 'a')
5
6 #set function remove duplicates in a list
7 unique = set(gett)
8 for link in unique:
9     #print link
10    savefile.write(link)
11
12 #get the top 1000 strings
13 with open('uniqueurls.txt', 'r') as uniqueurls:
14    top1000 = [next(uniqueurls) for x in xrange(1000)]
15 for links in top1000:
16    savefile2.write(links)
17
18
19 #i = 0
20
21 #def forloops(unique, i):
22     #for link in unique:
23         #i = i + 1
24         #print link
25         #savefile.write(link)
26         #return i
27
28
29 #while i < 1000:
30     #i = forloops(unique, i)

```

The final result of 1000 unique URLs are saved in a file called requiredurls.txt.

```

988 http://www.forbes.com/sites/anthonydimoro/2016/02/07/2016-nfl-all-forbes-sports-money-team/#3f7ae35
989 http://www.nbarevolution.com/news/bulls-jimmy-butler-salta-lall-star-game-per-infortunio/
990 http://basketballfanzone.org/2016/02/miami-heat-vs-charlotte-hornets-game-recap-whitesides-triple-d
991 https://apps.facebook.com/doubleucasino/?adcode=56ba3e8cd8a2c
992 http://www.springhillhomepage.com/volunteers-recreate-alliance-volleyball-courts-in-ag-center-cms-7
993 http://www.amazon.com/Earnhardt-Nation-Full-Throttle-NASCARs-Family/dp/0062367714
994 https://twitter.com/fecrvgl/status/697085513883389952
995 https://twitter.com/tizzzzzy/status/697139262018424832
996 http://www.azcentral.com/story/news/local/inspire/2016/02/09/special-olympics-sports-program-high-s
997 https://twitter.com/dosminutos/status/697139979387072513
998 https://www.youtube.com/watch?v=cif7Yi7bbDM&feature=youtu.be&a
999 https://twitter.com/WhitworthGerald/status/696464679586107392
1000 http://www.nba.com/warriors/tivo-playlist-volume-vi

```

Figure 3: Sample of part of requiredurls.txt

Problem 2

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/1/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.

* = `https://en.wikipedia.org/wiki/Histogram`

Answer

For this problem, my approach is this, for every link in the 1000 links, attach the url `http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/` in front of it. Then, use `curl` or `urllib2.urlopen` or whatever to open the newly generated links. (Be careful, if you are using `curl`, make sure to put the entire url inside `"`.) In the response, whenever the strings `'rel="memento"'`, `'rel="first memento"'`, `'rel="last memento"'`, `'rel="memento first"'`, `'rel="memento last"'` appear, it means there is a memento. Therefore, I just have to count them and add them up. Also, the response of some sites may contain the `'rel="timemap"'`, it means this site have more mementos in another link, make sure to open them up and count the mementos in there as well.

```

1 import commands
2 import os, sys
3 import re

```

```
4 import requests
5 import urllib2
6
7 readfile = open('requiredurls.txt', 'r')
8 savefile = open('countMemento.txt', 'a')
9 savefile2 = open('forgraph.txt', 'a')
10 #inputUrlsList = readfile.readlines()
11 #readfile.close()
12
13 timeMap = re.compile(r'<[^>]+>;rel\w*?=\w*?"timemap".*?')
14
15
16 def nextpage(foundTimeMap):
17
18     while len(foundTimeMap) == 1 :
19         url = foundTimeMap[0]
20         stripedURL = url.strip('<')
21         urlStriped = stripedURL.split('>')
22         nextPageLink = urlStriped[0]
23
24         command = 'curl --silent "' + nextPageLink + '"'
25         response = os.popen(command).read()
26
27         count = response.count('rel="memento"')
28         count2 = response.count('rel="first memento"')
29         count3 = response.count('rel="last memento"')
30         count4 = response.count('rel="memento first"')
31         count5 = response.count('rel="memento last"')
32         countbig = count + count2 + count3 + count4 + count5
33         foundTimeMap = timeMap.findall(response)
34
35         if countbig != 0:
36             return countbig
37
38     return 0
39
40
41 for link in readfile:
42     command = 'curl --silent "http://mementoproxy.cs.odu.edu/aggr/timemap/link'
43         /1/' + link.strip() + '"'
44     response = os.popen(command).read()
45     #command = 'http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/' + link
46     #response = urllib2.urlopen(url=command, timeout=10)
47     #time_map = response.read()
48
49     count = response.count('rel="memento"')
50     count2 = response.count('rel="first memento"')
51     count3 = response.count('rel="last memento"')
52     count4 = response.count('rel="memento first"')
53     count5 = response.count('rel="memento last"')
54
55     foundTimeMap = timeMap.findall(response)
56     returnedcount = nextpage(foundTimeMap)
```

```

57  finalcount = count + count2 + count3 + count4 + count5 + returnedcount
58
59  output = "{:<10}{%}" .format(finalcount, link)
60  print output
61
62  savefile.write(output)
63  savefile2.write(str(finalcount))
64  savefile2.write('\n')

```

The outputs are saved in two files, one contain the count and corresponding links(which is used later in problem 3), and another one only contain the count number(which is used for the graph).

```

345 0      http://dragplus.com/post/id/33525757
346 0      http://my.socialtoaster.com/splash/4LWpt/
347 0      http://thenextweb.com/dd/2016/02/09/amazon-wants-game-developers-to-use-its-new-free-open-;
348 0      http://www.pushpowerpromo.com/blog/the-indie-music-game-has-changed-here-s-how-you-play?91l
349 0      https://twitter.com/tmntmovie/status/696493168418619392
350 0      http://jp.automaton.am/articles/newsjp/fashion-media-editors-review-video-game-characters-;
351 0      https://nowthisnews.com/nba-star-steph-curry-s-daughter-riley-adorably-steals-post-game-in;
352 49      http://www.amazon.com/Techno-Source-01600-Tetris-Link/dp/B005RBVE8E
353 0      https://twitter.com/2girls1richard/status/697139943450275840
354 14      http://www.coconut-sports.de
355 0      https://vine.co/v/ilp0tj77ITx
356 16      https://www.youtube.com/watch?v=olbfKgzk_F0&feature=youtu.be&a
357 25      http://www.amazon.co.jp/GENTOS-%E3%82%B8%E3%82%A7%E3%83%B3%E3%83%88%E3%82%B9-%E3%80%90%E6%
358 0      http://www.foxsports.com/presspass/latestnews/2016/02/09/emmy-award-winning-broadcaster-fe;
359 0      http://www.twitch.tv/daisect#6182

```

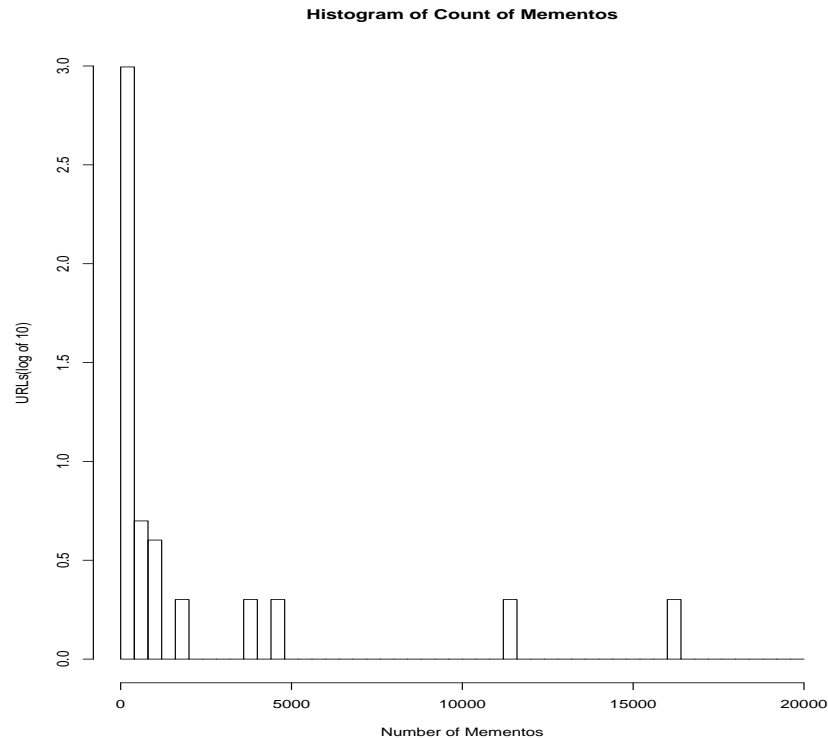
Figure 4: Sample of part of countMemento.txt

Below is the R code and graph of the histogram.

```

1  data2 = read.table(file.choose(), header = F)
2
3  r = hist(data2$V1, ylim = c(0, 1000), breaks = seq(from=0, to=20000, by=400),
4         plot = F)
5
6  r$counts = log10(r$counts+1)
7
8
9  plot(r, main="Histogram of Count of Mementos", xlab = "Number of Mementos",
10     ylab = "URLs(log of 10)")

```

Problem 3

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

<http://ws-dl.blogspot.com/2014/11/2014-11-14-carbon-dating-web-version-20.html>

Note: you'll should download the library and run it locally; 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.

Not all URIs will have Mementos, and not all URIs will have an estimated creation date. State how many fall into either categories.

Answer

For this problem, we have to create a Bitly account first. While waiting for your verification mail to activate your account, go install PhantomJS, download CasperJS, and download CarbonDate tools. CasperJS is just a compressed folder with some files in it, just put it somewhere you can remember, but make sure you create a short link in `/usr/local/bin` that

points to the casperjs file in the bin fold in the CasperJS folder. Now, activate your Bitly account and generate a token. Open the CarbonDate tool folder and open the config file, put your token in the right place and make sure the “CasperJSLocation” is set to where you created your short link towards the casperjs file. Now, we can finally start on the program itself.

To carbon date the 1000 links, I actually went into the local.py file and altered it. Since we are only looking for the estimated creation date, I just have the *cd* function return the variable *lowest*. Once I have the values outside the function, I convert it from string format back to object and use the time of now to subtract it to get the days since the site’s creation. The result is saved in carbonDate.txt file.

```
1 from checkForModules import checkForModules
2 import json
3 from ordereddict import OrderedDict
4 #import simplejson
5 import urlparse
6 import re
7
8 from getBitly import getBitlyCreationDate
9 from getArchives import getArchivesCreationDate
10 from getGoogle import getGoogleCreationDate
11 from getBacklinks import *
12 from getLowest import getLowest
13
14 from getLastModified import getLastModifiedDate
15 #Topsy service is no longer available
16 #from getTopsyScraper import getTopsyCreationDate
17 from htmlMessages import *
18 from pprint import pprint
19
20 from threading import Thread
21 import Queue
22 import datetime
23
24 import os,sys, traceback
25
26 readfile = open('countMemento.txt', 'r')
27
28 savefile = open('carbonDate.txt', 'a')
29
30
31 def cd(url, backlinksFlag = False):
32
33     #print 'Getting Creation dates for: ' + url
34
35
36     #scheme missing?
37     parsedUrl = urlparse.urlparse(url)
38     if( len(parsedUrl.scheme)<1 ):
39         url = 'http://' +url
40
41
```

```
42     threads = []
43     outputArray =[',',',',',',',',',',',']
44     now0 = datetime.datetime.now()
45
46
47     lastmodifiedThread = Thread(target=getLastModifiedDate , args=(url ,
48     outputArray , 0))
49     bitlyThread = Thread(target=getBitlyCreationDate , args=(url , outputArray ,
50     1))
51     googleThread = Thread(target=getGoogleCreationDate , args=(url , outputArray
52     , 2))
53     archivesThread = Thread(target=getArchivesCreationDate , args=(url ,
54     outputArray , 3))
55
56     if( backlinksFlag ):
57         backlinkThread = Thread(target=getBacklinksFirstAppearanceDates , args
58         =(url , outputArray , 4))
59
60     #topsyThread = Thread(target=getTopsyCreationDate , args=(url , outputArray ,
61     5))
62
63
64     # Add threads to thread list
65     threads.append(lastmodifiedThread)
66     threads.append(bitlyThread)
67     threads.append(googleThread)
68     threads.append(archivesThread)
69
70     if( backlinksFlag ):
71         threads.append(backlinkThread)
72
73     #threads.append(topsyThread)
74
75
76     # Start new Threads
77     lastmodifiedThread.start()
78     bitlyThread.start()
79     googleThread.start()
80     archivesThread.start()
81
82     if( backlinksFlag ):
83         backlinkThread.start()
84
85     #topsyThread.start()
86
87
88     # Wait for all threads to complete
89     for t in threads:
90         t.join()
91
92     # For threads
93     lastmodified = outputArray[0]
94     bitly = outputArray[1]
95     google = outputArray[2]
```

```

90     archives = outputArray[3]
91
92     if( backlinksFlag ):
93         backlink = outputArray[4]
94     else:
95         backlink = ''
96
97     #topsy = outputArray[5]
98
99     #note that archives["Earliest"] = archives[0][1]
100    try:
101        #lowest = getLowest([lastmodified, bitly, google, archives[0][1],
102    backlink, topsy]) #for thread
103        lowest = getLowest([lastmodified, bitly, google, archives[0][1],
104    backlink]) #for thread
105    except:
106        print sys.exc_type, sys.exc_value, sys.exc_traceback
107
108    result = []
109
110    result.append(("URI", url))
111    result.append(("Estimated Creation Date", lowest))
112    result.append(("Last Modified", lastmodified))
113    result.append(("Bitly.com", bitly))
114    result.append(("Topsy.com", "Topsy is out of service"))
115    result.append(("Backlinks", backlink))
116    result.append(("Google.com", google))
117    result.append(("Archives", archives))
118    values = OrderedDict(result)
119
120    #r = json.dumps(values, sort_keys=False, indent=2, separators=(',', ': '))
121
122    now1 = datetime.datetime.now() - now0
123
124
125    #print "runtime in seconds: "
126    #print now1.seconds
127    #print r
128    #print 'runtime in seconds: ' + str(now1.seconds) + '\n' + r + '\n'
129    #savefile.write(lowest)
130
131    return lowest
132
133    nowdaate = datetime.datetime.now()
134
135    for line in readfile:
136        link = line.split(" ")
137        url = link[-1]
138        returndate = cd(url)
139
140        if returndate != "":
141            date = datetime.datetime.strptime(returndate, '%Y-%m-%dT%H:%M:%S')

```

```

142         dayys = (nowdaate - date).days
143
144         print dayys
145         print type(dayys)
146         print link[0]
147         print type(link[0])
148         output = "{} {}".format(link[0], dayys)
149         savefile.write(output)
150         savefile.write('\n')
151
152
153
154 #if len(sys.argv) == 1:
155     #print "Usage: ", sys.argv[0] + " url backlinksOnOffFlag ( e.g: " + sys.
        argv[0] + " http://www.cs.odu.edu [--compute-backlinks] )"
156 #elif len(sys.argv) == 2:
157     #fix for none-thread safe strptime
158     #If time.strptime is used before starting the threads, then no exception
        is raised (the issue may thus come from strptime.py not being imported in
        a thread safe manner). -- http://bugs.python.org/issue7980
159     #time.strptime("1995-01-01T12:00:00", '%Y-%m-%dT%H:%M%S ')
160     #cd(sys.argv[1])
161 #elif len(sys.argv) == 3:
162     #datetime.strptime("1995-01-01T12:00:00", '%Y-%m-%dT%H:%M%S ')
163
164     #if (sys.argv[2] == '--compute-backlinks'):
165         #cd(sys.argv[1], True)
166     #else:
167         #cd(sys.argv[1])

```

```

1 1 448
2 6 3
3 4 405
4 5 529
5 16 2792
6 5 287
7 16 2792
8 9 220
9 16 2792
10 1 2
11 1 285
12 1 202
13 575 927
14 16 2792
15 16385 6270
16 13 401
17 16 2792
18 26 2228
19 5 347
20 43 854
21 2 2
22 49 1225
23 16 2792
24 2 366
-- --

```

Figure 5: Sample of part of carbonDate.txt

Now to finish the problem just use the data in carbonDate.txt to create a graph.

