CS896 Introduction to Web Science
Fall 2013
Report for Assignment 2

Corren G. McCoy

September 19, 2013

Contents

List of Figures

List of Tables

1 Question 1

1.1 Problem

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

1.2 Response

We used Twitter's Search API (`https://dev.twitter.com/docs/api/1.1`) to search for recent tweets which contained any of the following keywords: "Putin", "Syria", "Assad", "Obama," or "chemical weapons". Our Python program, *ExtractLinks*, as shown in Appendix A, stores the keywords in a list object with each element passed individually to the API as part of the query string. The text of each tweet returned from the search was parsed to remove any URLs which were then written to a text file (i.e., tweetFileCandidates.txt). Using the keywords indicated, we built a collection of approximately 8400 candidate URLs from which we sought to obtain the desired 1000 unique links. The unique links were identified using line-by-line processing of the tweetFileCandidates list. Each shortened URL was opened in order to verify the reference to an existing web site. Any URL with a response status of 200 was checked against the a master domain list. The full URL was obtained from the response header. To ensure uniqueness only one URL from any domain was allowed. This was facilitated by storing the domain as the key and the full URL as the value in a dictionary object which inherently ensures a unique domain for each key. Processing of the URLs in the tweetFileCandidates was terminated once 1000 domain URLs were obtained. This final list was then printed to a text (i.e., tweetFile1000.txt). A sample of the URLs harvested from Twitter is shown in Table 1.

`http://investmentwatchblog.com/`
`http://whithererehwon.wordpress.com/`
`http://maxcouti.blogspot.com/`
`http://kevskewl.wordpress.com/`
`http://www.marketing-projects.biz/`
`http://mundonovovelho.blogspot.com.br/`

Table 1: Sample Twitter URLs

2 Question 2

2.1 Problem

Download the TimeMaps for each of the target URIs. Create a histogram of URIs versus number of Mementos (as computed from the TimeMaps).

2.2 Response

Using our list of 1000 Twitter URIS, we used the ODU Memento Aggregator to generate the associated TimeMaps using our Python program *getTimeMaps* which is described in Appendix B. A sample from the TimeMap output is shown in Table 2. The Rel-Tag[1] of each TimeMap was examined to determine whether the information presented was tagged as a memento. We specifically looked for valid combinations of either "first memento", "last memento", "memento first", "memento last", or "first last memento" using regular expressions in Python. Each occurence of a memento tag increased the counter for each URI. The URI and the associated memento count were saved to a comma-delimited text file (i.e., histogram.txt) that was used as the data source for our histogram. We used the graphing

functions in R to produce the histogram shown in Figure 1. The histogram is representative of the long tail[2] with over 96% of the URIs having less than 2,500 mementos with a much sparser distribution in the range of 2,500 to approximately 30,000.

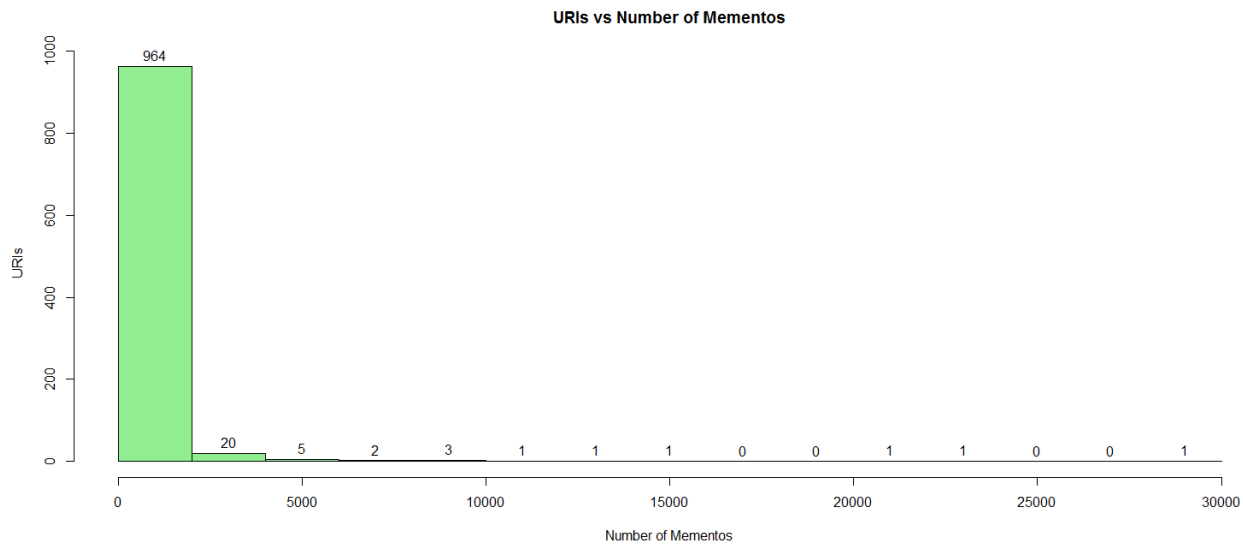| |
| --- |
| `<http://http://mementoproxy.cs.odu.edu/aggr/timemap/link/http://investmentwatchblog.com/>;rel="` |
| `<http://mementoproxy.cs.odu.edu/aggr/timegate/http://investmentwatchblog.com/>;rel="timegate",<` |
| `<http://web.archive.org/web/20100212220649/http://investmentwatchblog.com/>;rel="firstmemento";` |
| `<http://web.archive.org/web/20100409065504/http://investmentwatchblog.com/>;rel="memento";datet` |
| `<http://web.archive.org/web/20100415152016/http://investmentwatchblog.com/?>;rel="memento";date` |

Table 2: Sample TimeMap Output

Figure 1: URIs vs. Mementos

3 Question 3

3.1 Problem

Estimate the age of each of the 1000 URIs using the "Carbon Date" tool. For URIs that have at least one memento and an estimated creation date, create a graph with age (in days) on one axis and number of mementos on the other

3.2 Response

---

[1]http://www.microformats.org/wiki/Rel-Tag
[2]http://en.wikipedia.org/wiki/The_Long_Tail

Appendix A

Python Source for extractLinks.py

```python
# TwitterSearch package obtained from https://github.com/ckoepp/TwitterSearch

from TwitterSearch import *
import codecs
import urllib
import urllib2
from urlparse import urlparse


# This function iterates over the twitter entity object to
# find a particular element
def posted_url(dictionary):
    # Recursively iterates over entities to find expanded URLs
    for key, value in dictionary.iteritems():
        if isinstance(value, dict):
            posted_url(value)
        else:
            return(value)

# Save the final list of 1000 unique links
def saveToFile(pLinks):
    linkFile = codecs.open('C:/Python27/myFiles/linkFile.txt','w','utf-8')
    for key in pLinks:
        linkFile.write(key+'\n');

# This function searches Twitter for a particular keyword
def searchTwitter(pThisTerm):
 try:
    # create a TwitterSearchOrder object
    tso = TwitterSearchOrder()
    # Define all the keywords which must be passed URL encoded.
    # Only return/filter tweets that contain links
    tso.setKeywords(["filter%3Alinks",pThisTerm])
    # we want to see English tweets only'
    tso.setLanguage('en')
    # maximum number of tweets to return
```

6

```
        tso.setCount(100)
        # include the entity information
        tso.setIncludeEntities(True)
        #tso.setResultType('recent')

        # create a TwitterSearch object with my secret tokens (@CorrenMcCoy)
        ts = TwitterSearch(
            consumer_key = 'LrA1DdH1QJ5cfS8gGaWpOA',
            consumer_secret = '9AX14EQBLjRjJM4ZHt2kNf0I4G77sKsYX1bEXQCW8',
            access_token = '1862092890-FrKbhD7ngeJtTZFZwf2SMjOPwgsCToq2A451iWi',
            access_token_secret = 'AdMQmyfaxollI596G82FBipfSMhagv6hjlNKoLYjeg8'
         )

        # Iterate over the tweet entities which are in a nested dictionary
        tweetFile = codecs.open('C:/Python27/myFiles/tweetFile.txt','a','utf-8')
        for tweet in ts.searchTweetsIterable(tso):
            #print( '@%s tweeted: %s' % ( tweet['user']['screen_name'], tweet['text'] ) )
            if tweet['user']['url'] is not None:
                print (tweet['user']['url'])
                tweetFile.write(tweet['user']['url']+'\n');

        tweetFile.close()

    # Error handling. Close file and terminate
    except TwitterSearchException as e:
        tweetFile.close()
        print(e)
        exit()


def findUniqueLinks():
    tweetFile = open('C:/Python27/myFiles/tweetFile.txt')
    # initialize the list to hold all the fully expanded URIs
    # that respond with 200
    tweetURLs=[]
    # create an empty dictionary
    domains={}
    for line in iter(tweetFile):
        uri = line.rstrip('\n')
        #package the request
        request=urllib2.Request(uri)
        request.add_header('User-agent','Mozilla 5.10')
        try:
            response=urllib2.urlopen(request)
            if response.code == 200:
                tweetURLs.append(response.url)
                # make sure there's only one URI with the same domain.
                # http://stackoverflow.com/questions/10362453/unique-by-domains-urls-list
                # Return the last URL for each domain the URL list. By definition, the key
                # in a dictionary is unique.
```

7

```python
            domains=dict((urlparse(u).netloc, u) for u in tweetURLs).values()

            # Stop traversing the file when we reach the 1000th domain
            if len(domains)== 1000:
                # Write out the URLS so we can use them for other tasks
                saveToFile(domains)
                return
        print response.code,response.url, "Domains=", len(domains)
        response.close()
    except urllib2.HTTPError,e:
        print "Error", line, e
        continue
    except urllib2.URLError, e:
        print "Error", line, e
        continue
    except IOError, e:
        print "Error", line, e
        continue
    # how many links do we have?
    print "Unique links = ", len(domains)
    tweetFile.close()



##############################################################################
#  This is main procedure in this package
##############################################################################
def extractLinks():
    # Use these keywords to build a list of Tweets from which we will extract
    # the desired 1000 unique links.
    keywordList={"Putin","Syria","Assad","Obama","chemical%20weapons"}
    for thisTerm in keywordList:
        print "Tweets for keyword>>>",thisTerm
        searchTwitter(thisTerm)
    print "Twitter search complete>>>>>>"

    # Read the file of links.
    findUniqueLinks()
    print "1000 links saved to file>>>>>"
```

Appendix B

Python Source for getTimeMaps.py

```python
#!/usr/bin/python -B
import urllib2
import urllib
import codecs
import re

# Save the final count to a comma delimited file
def saveToFile(pUri, pCount):
    # append the data
    histogram = codecs.open('C:/Python27/myFiles/histogram.txt','a','utf-8')
    histogram.write(pUri + ',' + str(pCount) + '\n')
    histogram.close()


def countMementos(pUri, pTimeMap):
    # Re-use from regular expressions from Scott's timeMap.py
    tokenizer = re.compile('(<[^>]+>|[a-zA-Z]+="[^"]*"|[;,])\\s*')
    mementoCount=0
    # The timeMap is passed as a string. Read/split each line
    for line in pTimeMap.splitlines():
        # Parse the line into tokens to find what's in the rel= tag
        tokens=tokenizer.findall(line)
        for x in tokens:
            # Find the token using Scott's logic from timeMap.py
            if x[:4] == 'rel=':
                rel=x[5:-1]
                # if "memento" is anywhere in the rel tag, let's count it
                if rel.find('memento') <> -1:
                    mementoCount = mementoCount + 1

    # This will be a number >= 0
    saveToFile(pUri, mementoCount)


##############################################################################
#  This is main function in this package
##############################################################################
```

```python
'''
Invoke the ODU Memento Aggregator for each of our 1000 unique
Twitter links.  Search the resulting timeMaps, if any, for valid mementos
which are identified as any valid combination of:
    rel="memento"
    rel="first memento"
    rel="last memento"
    rel="memento first"
    rel="memento last"
    rel="first last memento"
'''

def findMementos():
    # Base setting for the aggregator
    uriBase="http://mementoproxy.cs.odu.edu/aggr/timemap/link/"
    notInArchive=0

    # Write the header of our histogram data file
    histogram = codecs.open('C:/Python27/myFiles/histogram.txt','w','utf-8')
    histogram.write("uri" + "," + "mementos" + "\n")
    histogram.close()

    # Open the file containing our 1000 links from Twitter
    tweetFile = open('C:/Python27/myFiles/tweetFile1000.txt')
    lineNo=0
    for line in iter(tweetFile):
        lineNo=lineNo+1  # for checking progress of iteration over URIs
        uri=line.rstrip('\n')
        # Package the request. Append the target uri to complete.
        uri_t=uriBase+uri
        print lineNo, uri_t
        # Search for any timeMaps
        try:
            request=urllib2.Request(uri_t)
            response=urllib2.urlopen(request)
            if response.code==200:
                timeMap=response.read()
                # Parse out the string we're looking for.
                # Record the URI and the count in a file.
                # The file will be used later to create a Histogram in R
                countMementos(uri, timeMap)
            response.close()
        except urllib2.HTTPError,e:
            print "Error", e
            # A 404 response from aggregator indicates "Resource not in archive"
            saveToFile(uri, notInArchive)
            continue
        except urllib2.URLError, e:
             print "Error", e
             continue
```

```
    except IOError, e:
        print "Error", e
        continue
# Done processing. Close the file.
tweetFile.close()
print ">>> Session Complete. . ."
```

Appendix C

R Source for Histogram

```
w1 <- read.csv(file="histogram.txt",sep=",",head=TRUE)
names(w1)
m<-mean(w1$mementos)
std<-sqrt(var(w1$mementos))
hist(w1$mementos,
col="lightgreen",
main="URIs vs Number of Mementos",
xlab="Number of Mementos",
ylab="URIs",
labels=TRUE)
curve(dnorm(x, mean=m, sd=std), add=TRUE)
```