Khaja Masroor Ahmed

Assignment 1

CS 751: Introduction to Digital Libraries
Dr. Michael Nelson
Spring d2015

February 14, 2015

Contents

1

Question 1

1.1 Question

Write a program that extracts 10000 tweets with links from Twitter.

- Save the tweet URIs, and the mapping to the link(s) each tweet contains.
- For each t.co link, use "curl -I -L" to record the HTTP headers all the way to a terminal HTTP status (i.e. chase down all the redirects)
- How many unique final URI's? How many duplicate URI's?
- Build a histogram of how many redirects (every URI will have at least 1)
- Build a histogram of HTTP status codes encountered (youll have at least 20000: 10000 301s, and 10000+ more)

1.2 Solution

For solving the above problem I have used Python as the programming language. Following are the steps I've taken to solve the given problem:

- I first registered a twitter application to generate the consumer key and consumer secret key for using the Twitter API.
- The key's generated in the above step are used for authenticating the application for requesting the tweets.
- I encountered multiple packages while researching for packages for fetching tweet data and decided upon TwitterSearch package.
- I was facing an issue with the TwitterSearch package where I was receiving limited results. In-spite of making modifications to the keyword for search I wasn't able to exceed more than 800 results.
- I eventually moved to tweepy package with which I was receiving approximately 2000 results and it would stop again. Along with the keyword argument I also passed another argument 'since = 2014' to the API search as the tweets fetched would stop at a certain date. This additional argument solved my problem. I was able to retrieve more than 10000 tweets.

- From the tweet data received from the API, I fetched the tweet identifier, created date, user identifier, tweet text and of course the URI.
- The above mentioned data was processed to json format and written to a file ‘output.txt’.
- For retrieving the final URI, history of status code’s encountered, I found requests package to be the most powerful and appropriate for use in my context.
- I used the requests package in ‘requestStatus.py’ for fetching the above mentioned information and output this information to ‘status.txt’ file.
- Now with the raw information available, I processed the data further based on the requirement as specified in the question.
- Based on the tweet data, I found that the number of unique URI’s that were encountered were 6431, duplicate links were 3999. The total number of greater than 10000 as some tweets had more than one URI. This information was calculated using script ‘getUniqueURI.py’ and the result was stored in ‘unique.txt’.
- Also, I calculated the number of re-directs for each URI using the script ‘getURIRedirect.py’ and the result was stored in ’uriRedirect.txt’. This information was processed through R for generating a histogram as shown in the figure below.

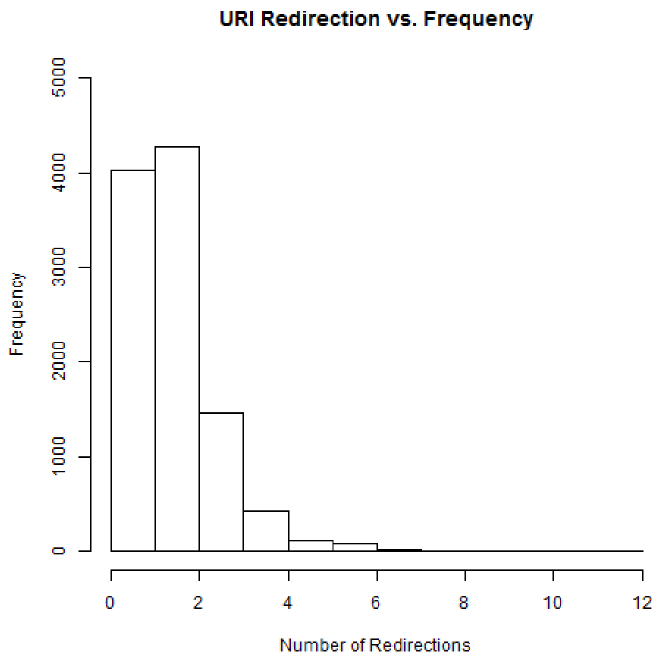

Fig. 1.1. URI Redirection vs. Frequency

- The raw information from status.txt was also used to calculate the frequency of the status codes encountered for these URI's and the figure below represents the histogram for this data.

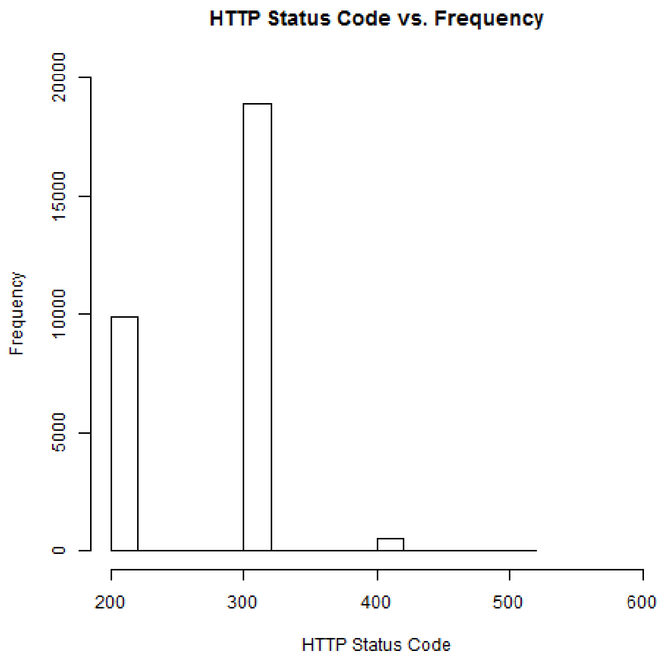

Fig. 1.2. HTTP Status Code vs. Frequency

1.3 Code Listing

```
#CS751 Spring 2015 Offered by Dr. Michael Nelson -
    Assignment 1
#Filename: getTweets.py
#Author: Khaja Masroor Ahmed
#UIN: 00999044
#CS Email: kahmed@cs.odu.edu

import tweepy
import json
import time
import datetime

consumer_key = "iuKUndPfIF5aWnNl0Ayq9Ztgt"
consumer_secret = "
    QuNsF4gL2LssmbcdtKpyLZGiQctz98T4hXWcAKrBYGh72ZTFC8"

access_token = "549294315-
    P89swbZzgiP2n9bq6fW2T2jm5etru6Wr6TN08Lg3"
access_token_secret = "
    NMzDaS5doFtHxXxebE68AunmRHTsFfLxwAkk3LsDN75JH"

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
f = open('output.txt','w')

tweetCount = 1
print('Started execution')
cursor = tweepy.Cursor(api.search, q ='http:', since_id =
    2014).items()
while True:
        try:
                tweet = cursor.next()
                for tweet in cursor:
                        tweetData = {}
                        tweetData['tweetId'] = tweet.id
                        tweetData['createdAt'] = tweet._json
                            ['created_at']
                        tweetData['userId'] = tweet.user.
                            _json['id']
                        tweetData['tweetCount'] = tweetCount
                        tweetData['text'] = tweet.text
                        urlData = []

                        for url in tweet._json['entities']['
                            urls']:
```

```
                urlData.append(url['url'])
            tweetData['url'] = urlData
            tweetData = json.dumps(tweetData)

            if tweetCount == 11999:
                print('Complete')
                print(datetime.datetime.now())
                f.close()
                break
            f.write(tweetData + "\n")
            tweetCount = tweetCount + 1
    except tweepy.TweepError as e:
        print(e)
        time.sleep(900)
        continue
    except StopIteration:
        break
print('Program Executed!')
```

Listing 1.1. Python program for retrieving tweets.

```
#CS751 Spring 2015 Offered by Dr. Michael Nelson -
    Assignment 1
#Filename: requestStatus.py
#Author: Khaja Masroor Ahmed
#UIN: 00999044
#CS Email: kahmed@cs.odu.edu

import requests
import json
import string

f = open('output.txt', 'r+')
w = open('status.txt','w')
err = open('error.txt','w')
count = 1
for line in f:
        data = json.loads(line)
        i = 0
        try:
                tweetData = {}
                tweetData['index'] = count
                tweetData['tweetId'] = data['tweetId']
                tweetData['createdDate'] = data['createdAt']
                tweetURLData = []
                for url in data['url']:

                        r = requests.head(data['url'][i],
                            allow_redirects=True, timeout =
                            15)
                        j = 0

                        URLData = {}

                        URLData['finalURL'] = r.url
                        statusCode = []
                        statusCode.append(r.status_code)
                        for response in r.history:
                                statusCode.append(response.
                                    status_code)
                        URLData['statusCode'] = statusCode
                        URLData['tweetURL'] = data['url'][i]
                        tweetURLData.append(URLData)

                        i = i + 1
                tweetData['tweetURLData'] = tweetURLData

                tweetData = json.dumps(tweetData)
                print count
```

```
                    w.write(tweetData + "\n")
            except:
                    err.write(str(count) + '\n')
                    continue
            count = count + 1
f.close()
w.close()
err.close()
```

Listing 1.2. Python program to request the header information for each URI.

```
#CS751 Spring 2015 Offered by Dr. Michael Nelson - Assignment 1
#Filename: getUniqueURI.py
#Author: Khaja Masroor Ahmed
#UIN: 00999044
#CS Email: kahmed@cs.odu.edu

import json
import sets

f = open('status.txt','r+')
w = open('unique.txt','w')
uniqueURLs = sets.Set()
count = 1
for line in f:
        data = json.loads(line)
        for urlData in data['tweetURLData']:
                uniqueURLs.add(urlData['finalURL'])
                count = count + 1
w.write('Unique: ' + (str)(len(uniqueURLs)) + '\n')
w.write('Duplicates: ' + (str)(count - len(uniqueURLs)) + '\n')
w.write('Total: ' + (str)(count) + '\n')
f.close()
w.close()
```

Listing 1.3. Python program for retrieving the number of unique, duplicate URI's.

```
#CS751 Spring 2015 Offered by Dr. Michael Nelson -
    Assignment 1
#Filename: getURIRedirect.py
#Author: Khaja Masroor Ahmed
#UIN: 00999044
#CS Email: kahmed@cs.odu.edu

import json

f = open('status.txt','r+')
w = open('uriRedirect.txt','w')
w2 = open('uriRedirectCount.txt','w')
uriRedirect = {}
for line in f:
        data = json.loads(line)
        count = 0
        for tweetData in data['tweetURLData']:
                for status in tweetData['statusCode']:
                        if(str(status).startswith('3')):
                                count = count + 1
                w.write(str(count) + '\t' + tweetData['
                    finalURL'] + '\n')
                if count in uriRedirect:
                        uriRedirect[count] = uriRedirect[
                            count] + 1
                else:
                        uriRedirect[count] = 1
w2.write(str(uriRedirect) + '\n')
w.close()
w2.close()
```

Listing 1.4. Python program for calculating the number of redirects for each URI.

```
#CS751 Spring 2015 Offered by Dr. Michael Nelson - Assignment 1
#Filename: getStatusCodeFrequency.py
#Author: Khaja Masroor Ahmed
#UIN: 00999044
#CS Email: kahmed@cs.odu.edu

import json
import string

f = open('status.txt', 'r+')
w = open('statusCode.txt', 'w')
statusCode = {}
for line in f:
        data = json.loads(line)
        if data['index'] == 10000:
                print 'Program Executed'
                break
        for tweetData in data['tweetURLData']:
                for status in tweetData['statusCode']:
                        w.write(str(status) + '\n')
                        if status in statusCode:
                                statusCode[status] = statusCode[status] + 1
                        else:
                                statusCode[status] = 1
```

Listing 1.5. Python program for retrieving the different status codes encountered by each URI.

```r
uriDataset <- read.csv("C:/Users/kahmed/Desktop/uriRedirect.
    txt",stringsAsFactors=F,header=FALSE,sep="\t")
data = uriDataset[,1]
png("C:/Users/kahmed/Desktop/uriRedirect.png")
hist(data,main="URI Redirection vs. Frequency",freq=T,xlab="
    Number of Redirections",ylab="Frequency",xlim=c(0,12),
    ylim=c(0,5000))
```

Listing 1.6. R program for generating the histogram for URI Redirection vs. Frequency

```r
statusCode <- read.csv("C:/Users/kahmed/Desktop/statusCode.
    txt",stringsAsFactors=F,header=FALSE)
data = statusCode[,1]
png("C:/Users/kahmed/Desktop/statusCode.png")
hist(data,main="HTTP Status Code vs. Frequency",freq=T,xlab=
    "HTTP Status Code",ylab="Frequency",xlim=c(200,600),ylim
    =c(0,20000))
```

Listing 1.7. R program for generating the histogram for HTTP Status Code vs. Frequency

Question 2

2.1 Question

- Use “Carbon Date” to estimate the age of each link(s) in a tweet.
  - See: http://ws-dl.blogspot.com/2013/04/2013-04-19-carbon-dating-web.html
- For each t.co link, use “curl -I -L” to record the HTTP headers all the way to a terminal HTTP status (i.e. chase down all the redirects)
  - Many (most?) deltas will be 0, but there should be many ¿ 0.
- For these deltas, compute: median, mean, std dev, std err.
- Use wget to download the text for all the links. Hold on to those, well come back to them later. See:
  - http://superuser.com/questions/55040/save-a-single-web-page-with-background-images-with-wget
  - http://stackoverflow.com/questions/6348289/download-a-working-local-copy-of-a-webpage

2.2 Solution

- For carbon dating the tweet URI’s, I first cloned the github repository.
- As per the instructions of the author of the library, I then created a bitly account and inserted the accesstoken in the configuration file.
- Some of the URI’s were taking longer than a minute and some a few hours to retrieve the estimated creation date, I split the input files into 1000 URI’s each and created five instances of the local.py file with the necessary updates.
- By splitting these URI’s into batches I was able to complete fetching data from each of the URI’s in two days.
- Also, some URI’s were stuck for over a few hours and got repeatedly stuck at the same location. I deleted these URI’s and continued carbon dating the rest in the list.

- In the file 'local.py' I created a JSON structure with the tweet identifier, creation date of tweet and the estimated creation date of the URI.
- I used the 'getTweetAge.py' for fetching the age of each of these URI's. The figure below indicates the histogram of the $\text{Age}_{\text{tweet}}$ - Age_{URI} vs. Frequency.

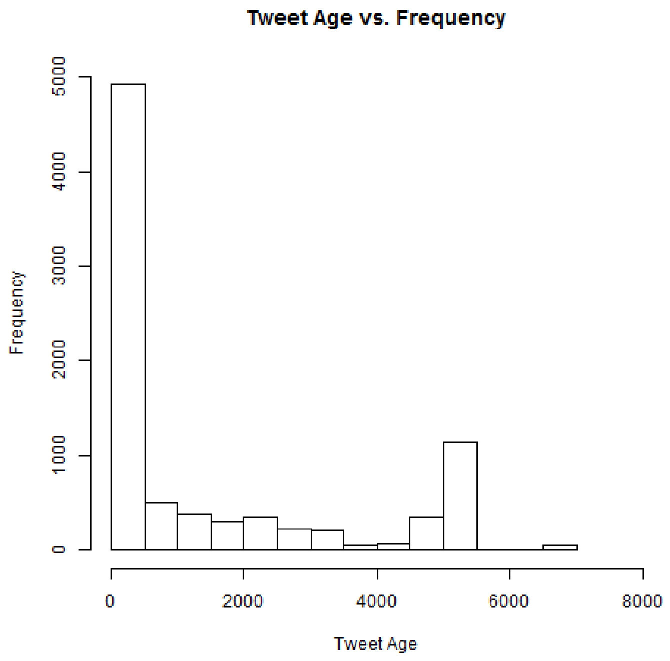

Fig. 2.1. Age vs. Frequency

- Using the age of each of these URI's I found the mean as 1409.735 days, median as 235 days, standard deviation as 1948.022 days, standard error as 21.10941 days.

2.3 Code Listing

```python
from checkForModules import checkForModules
import json
from ordereddict import OrderedDict
import urlparse
import re

from getBitly import getBitlyCreationDate
from getArchives import getArchivesCreationDate
from getGoogle import getGoogleCreationDate
from getBacklinks import *
from getLowest import getLowest

from getLastModified import getLastModifiedDate
from getTopsyScrapper import getTopsyCreationDate
from htmlMessages import *
from pprint import pprint

from threading import Thread
import Queue
import datetime

import os,sys, traceback




def cd(url, backlinksFlag = False):

    #print 'Getting Creation dates for: ' + url


    #scheme missing?
    parsedUrl = urlparse.urlparse(url)
    if( len(parsedUrl.scheme)<1 ):
        url = 'http://'+url


    threads = []
    outputArray =['','','','','','']
    now0 = datetime.datetime.now()


    lastmodifiedThread = Thread(target=getLastModifiedDate,
        args=(url, outputArray, 0))
    bitlyThread = Thread(target=getBitlyCreationDate, args=(
        url, outputArray, 1))
    googleThread = Thread(target=getGoogleCreationDate, args
        =(url, outputArray, 2))
    archivesThread = Thread(target=getArchivesCreationDate,
        args=(url, outputArray, 3))

    if( backlinksFlag ):
        backlinkThread = Thread(target=
            getBacklinksFirstAppearanceDates, args=(url,
            outputArray, 4))

    topsyThread = Thread(target=getTopsyCreationDate, args=(
        url, outputArray, 5))


    # Add threads to thread list
    threads.append(lastmodifiedThread)
    threads.append(bitlyThread)
    threads.append(googleThread)
    threads.append(archivesThread)

    if( backlinksFlag ):
        threads.append(backlinkThread)

    threads.append(topsyThread)


    # Start new Threads
    lastmodifiedThread.start()
    bitlyThread.start()
    googleThread.start()
    archivesThread.start()

    if( backlinksFlag ):
        backlinkThread.start()

    topsyThread.start()


    # Wait for all threads to complete
    for t in threads:
        t.join()

    # For threads
    lastmodified = outputArray[0]
    bitly = outputArray[1]
    google = outputArray[2]
    archives = outputArray[3]

    if( backlinksFlag ):
        backlink = outputArray[4]
    else:
        backlink = ''

    topsy = outputArray[5]

    #note that archives["Earliest"] = archives[0][1]
    try:
        lowest = getLowest([lastmodified, bitly, google,
            archives[0][1], backlink, topsy]) #for thread
    except:
       print sys.exc_type, sys.exc_value , sys.exc_traceback



    result = []

    result.append(("URI", url))
    result.append(("Estimated Creation Date", lowest))
    result.append(("Last Modified", lastmodified))
    result.append(("Bitly.com", bitly))
    result.append(("Topsy.com", topsy))
    result.append(("Backlinks", backlink))
    result.append(("Google.com", google))
    result.append(("Archives", archives))
    values = OrderedDict(result)
    r = json.dumps(values, sort_keys=False, indent=2,
        separators=(',', ': '))

    now1 = datetime.datetime.now() - now0

    print 'runtime in seconds:  ' + str(now1.seconds) + '\n
        ' + r + '\n'

    return lowest

f = open('status.txt', 'r+')
w = open('carbonDate.txt', 'w')

for line in f:
        data = json.loads(line)

        for tweetData in data['tweetURLData']:
                print data['index']
                print datetime.datetime.now()
                carbonDate = {}
                carbonDate['oldestDate'] = cd(tweetData['
                    finalURL'])
                carbonDate['createdDate'] = data['
                    createdDate']
                carbonDate['tweetId'] = data['tweetId']
                carbonDate = json.dumps(carbonDate)
                w.write(carbonDate + "\n")
                print carbonDate
```

Listing 2.1. Python program for carbon dating the URI's.

```python
import json
import datetime

f = open('carbonData.txt','r+')
w = open('age.txt','w')
count = 0
for line in f:
        data = json.loads(line)
        if data['oldestDate'] != '':
                count = count + 1
                oldestDate = datetime.datetime.strptime(data['oldestDate'], '%Y-%m-%dT%H:%M:%S')
                createdDate = datetime.datetime.strptime(data['createdDate'], "%a %b %d %H:%M:%S +0000 %Y")
                age = abs((oldestDate.date() - createdDate.date()).days)
                w.write(str(age) +'\n')
print('Execution Complete')
```

Listing 2.2. Python program for calculating the $\text{Age}_{\text{tweet}}$ - Age_{URI}.

```
age <- read.csv("C:/Users/kahmed/Desktop/age.txt",
     stringsAsFactors=F, header=FALSE)
data = age[,1]

png("C:/Users/kahmed/Desktop/tweetAge.png")
hist(data,main="Tweet Age vs. Frequency",freq=T,xlab="Tweet
     Age",ylab="Frequency",xlim=c(1,8000),ylim=c(0,5000))

se <- function(x) sqrt(var(x)/length(x))
write("Mean", file = "C:/Users/kahmed/Desktop/meanMedMode.
     txt")
write(mean(data), file = "C:/Users/kahmed/Desktop/
     meanMedMode.txt",
        append = TRUE, sep = "\t")
write("Median", file = "C:/Users/kahmed/Desktop/meanMedMode.
     txt",
        append = TRUE, sep = "\t")
write(median(data, na.rm = FALSE), file = "C:/Users/kahmed/
     Desktop/meanMedMode.txt",
        append = TRUE, sep = "\t")
write("Standard Deviation", file = "C:/Users/kahmed/Desktop/
     meanMedMode.txt",
        append = TRUE, sep = "\t")
write(sd(data, na.rm = FALSE), file = "C:/Users/kahmed/
     Desktop/meanMedMode.txt",
        append = TRUE, sep = "\t")
se <- function(data) sd(data)/sqrt(length(data))
write("Standard Error", file = "C:/Users/kahmed/Desktop/
     meanMedMode.txt",
        append = TRUE, sep = "\t")
write(se(data), file = "C:/Users/kahmed/Desktop/meanMedMode.
     txt",
        append = TRUE, sep = "\t")
```

Listing 2.3. R program for generating the histogram for Age vs. Frequency and calculating mean, median, mode and standard error