CS 432: Homework Number Two

Due on Febuary 14, 2018 at 4:20 PM

Alexander Nwala

Tim Bruce

I would like to begin by pointing out that my comments are poorly formatted in ths LaTeX document. I am working on figuring out what is causing them to be inconsistent.

Problem 1

Write a Python program that extracts 1000 unique (collect more e.g., 1300 just in case) links from Twitter. Omit links from the Twitter domain (twitter.com).
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 streaming or search feature to find URIs. If you find something inappropriate for any reason you see fit, just discard it and get some more links. We just want 1000 links that were shared via Twitter.
Hold on to this collection and upload it to github – we'll use it later throughout the semester.
Solution
After some deliberation, I decided to go about this problem in a slightly different manner that what may have been expected in order to make it more general-use. I broke the program into four parts.

1. Stream Tweets with inputtable keyword for designated amount of time.
2. Remove the links from the tweets, follow the redirects, and remove duplicates. (This step also sorts them in alphabetical order)
3. Get the number of mementos for histogram analysis.
4. Create the histogram.

The nice thing about this method, is that after each step all the data is saved, so if not enough data is collected the first time around, more data can be added.

Part A

The streaming is done almost identically to the tutorial that was provided.[**?**] The only part that is heavily modified is the onData function in the listener class which I would like to direct your attenition to.

Listing 1: onData function in the listener class.

```
def on_data(self, data):
        json_new_data = json.loads(data)  # Format the data as JSON
        json_data = {}  # Place to store all of the tweets.
        json_data['text'] = json_new_data['text']  # Store the tweet text.
        json_data['links'] = json_new_data['entities']['urls']  # Store the URIs.
        json_data['time'] = json_new_data['created_at']  # Store the creation date.
        self.tweets['tweets'].append(json_data)  # Add the tweet to the list of tweets.
        f = open('tweets.json', 'w')  # Save the tweets for future use.
        json.dump(self.tweets, f)
        f.close()

        self.counter += 1
        if self.endTime > t.datetime.now():  # If the timer has not expired,
            return True  # continue.
        else:  # If the timer has expired,
            return False  # stop.
```

Note that the function has been modified to take the string 'data' as a parameter. This string is actually the string output of a JSON file, which is read into the Python json data type, which can be used like a

dictionary. Using this method, it can append to this JSON file at the end.

Also at the end of this code is the timer that stops the function if enough time has elapsed. This uses the current time at start and compares it to the time when the function is called. It should be noted that it will collect the first Tweet after the timer has technically expired.

The tweets are finally stored with the following format:

Listing 2: JSON Format for Storing Tweets.


```
{"tweets":
  [
  {"text": "nagaaral sana ako pero nakatitig pa rin ako sa labas kasi puro smog feeling
    ko nasa baguio ako or someth pero pOLLU\u2026 https://t.co/zQcFlbDHTx",
  "links":
    [
    {"url": "https://t.co/zQcFlbDHTx",
    "expanded_url": "https://twitter.com/i/web/status/1095831103737221120",
    "display_url": "twitter.com/i/web/status/1\u2026",
    "indices": [117, 140]
    }
    ],
  "time": "Wed Feb 13 23:44:29 +0000 2019"
  },
  {"text": "\ud83d\ude22. NASA bids adieu to Mars rover that kept going and going and
    going https://t.co/nncdsQJ3rx",
  "links":
    [
    {"url": "https://t.co/nncdsQJ3rx",
    "expanded_url": "https://reut.rs/2GF76ai",
    "display_url": "reut.rs/2GF76ai",
    "indices": [70, 93]
    }
    ],
  "time": "Wed Feb 13 23:44:30 +0000 2019"
  }
  ]
}
```


NOTE: Most tweets with the tag 'nasa' are talking about two things:

1. Ariana Grande appears to recently have come out with a song called 'NASA' which people are going nuts about.

2. The official end of attempts to contact the Opportunity Rover on Mars after an astounding fifteen year mission. RIP OPPY :'(

Part B

To parse the Tweets, the program removes the 'expanded_url' tagged data from the JSON code above. The program then uses a method from Assignment 1 to follow the links using the urllib library in Python.[**?**] Python's urllib includes a function 'urllib.geturl()' which gets the url of the website after the link has been followed past all of the redirects it may have.

Next, the program removes all of the duplicates by hashing all of the links.

Listing 3: Duplicate Removal Function.

Problem 1 continued on next page...

```
1  def removeDuplicates(urls): # This function removes duplicate URIs.
2      x = dict()              # Dictionary for storing hashed URIs.
3      for url in urls:
4          h = hash(url)       # Basic Python hash function appears to be appropriate for
       this use case.
5          if h not in x:      # If hash does not yet exist.
6              x[h] = [url, 1] # Add the hash and a counter for duplicate URIs.
7          else:               # Else, hash does exist.
8              if x[h][0] != url:# Check to see if hash has duplicated a key incorrectly.
9                  raise Exception("Something is wrong with the hash")
10             i = x[h]
11             x[h] = [i[0], i[1] + 1]# Add to the counter.
12     links = []
13     for site in x:          # Add one of each hashed link to the output list.
14         links.append(x[site][0])
15     links.sort()            # Sort the output into alphabetical order.
16     return links
```

Note that on line 4 the function uses the normal hash function included in Python. In the past, I have found that this function is robust and hastle free, so I elected to use it here. The function saves the link and the number of times the link has appeared in a list in the dictionary. At the end, when all of the links are removed, it sorts the links alphabetically.

Finally, the links are saved in a text file with each link taking up a line.

Listing 5: Final Links Listed in Order.

```
1 https://apod.nasa.gov/apod/image/1902/marsshadow_opportunity_960.jpg
2 https://apod.nasa.gov/image/1902/marsshadow_opportunity_960.jpg
3 https://apple.news/AGVsWYnAJSM6tTW4ccYNZGg
4 https://apple.news/AJYIENZ2HSoqCFQQGCkXoqg
5 https://apple.news/AQJW_hgugSTKqivlo9Tdf2Q
6 https://apple.news/Aym9uePiuSp-VnNkLvdycgg
7 https://archaeologynewsnetwork.blogspot.com/2019/02/nasa-finds-possible-second-impact.
      html?utm_source=dlvr.it&utm_medium=twitter
8 https://arstechnica.com/science/2019/02/nasa-emphasizing-speed-in-its-return-to-the-moon
      /
```

NOTE: I didn't know that people used Apple News at all. I found it interesting when I saw these four links in the maddness that is the hashedlinks.txt file.

This concludes the first problem on the homework assignment, but the program for parts one and two is actually just one big program because I got annoyed with juggling a bunch of .py files. The structure will carry through from this problem.

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://memgator.cs.odu.edu/timemap/link/http://www.cs.odu.edu/
or:
URI-T = http://memgator.cs.odu.edu/timemap/json/http://www.cs.odu.edu/

(depending on which format you'd prefer to parse)
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. The x-axis will have the number of mementos, and the y-axis will have the frequency of occurence.

Solution

Part A

It was fairly straightforward to get the JSON from the URI menitioned in the question. After that, it was just a matter of reading how many mementos the URI had, and sorting them somehow.

The method for sorting the mementos for the histogram incorporates the use of a dictionary. This dictionary stores the number of mementos a URI has as the key. The data that the key points to is just an integer that is the number of URIs that had that many mementos.

The program then writes the dictionary to the mementos.json file using the json.dump() function. Producing a result something like below.

Listing 6: Mementos stored in JSON.

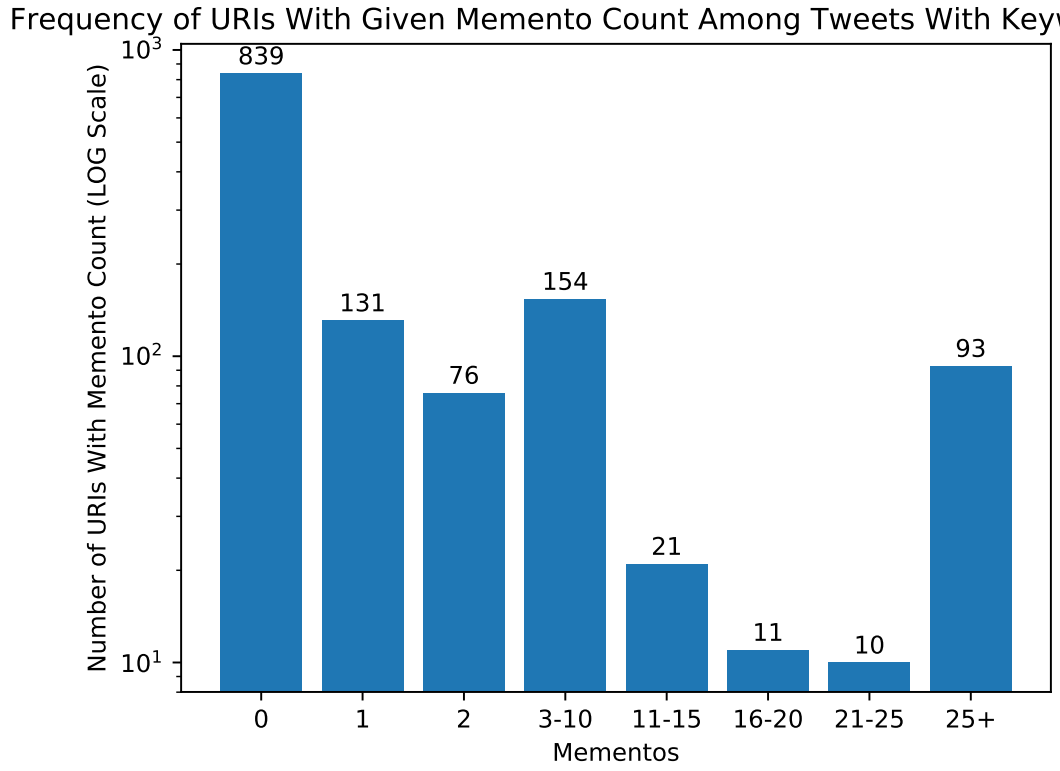

```
{"38": 2, "2574": 1, "62": 1, "8": 13, "1": 131, "810": 1, "191": 1, "0": 839, "58": 1,
    "3": 40, "8836": 1, "2": 76, "4": 23, "64": 1, "9": 13, "13": 9, "147": 1, "538": 1,
     "42": 3, "199": 2, "33": 4, "108": 1, "15680": 1, "10": 7, "37": 2, "21": 2,
    "4138": 1, "93": 2, "79": 1, "66": 1, "76": 1, "3741": 1, "46": 1, "112": 1, "5907":
     1, "4134": 1, "30": 2, "5": 16, "6244": 1, "2666": 1, "336": 1, "2657": 1, "19540":
     1, "7": 26, "35": 1, "494": 1, "6": 16, "24": 2, "28": 2, "99": 1, "434": 1,
    "1785": 1, "111": 1, "22": 1, "129": 1, "29": 3, "47": 1, "526": 1, "244": 1, "75":
    1, "25": 3, "23": 2, "17": 4, "18": 3, "896": 1, "1216": 1, "4237": 1, "732": 1,
    "114": 2, "24788": 1, "16": 2, "11": 4, "1275": 1, "36": 1, "82": 1, "20": 1, "12":
    2, "7274": 1, "35979": 1, "19": 1, "15": 5, "14": 1, "35085": 1, "40": 1, "509": 1,
    "53": 1, "500": 1, "320": 1, "728": 1, "122": 1, "26": 2, "146": 1, "22812": 1,
    "34": 1, "930": 1, "104": 1, "7971": 1, "40511": 1, "27": 2, "45": 1, "132": 2,
    "50": 1}
```


Part B

As you can see above, a lot of numbers of mementos are only had by one page. This means that to create a histogram, we will need to group some of them together.

To create a histogram, I used matplotlib, and made extensive use of matplotlibs documentation.[**?**] This made it easy to take a simple list of data and display it as a chart. There is not a huge amount of explanation to go along with this part but I do need to credit Joshua Gahan, because I based the numeric categories of my chart off of the one he posted to the class's slack page.

Image 1: Histogram Output From Program.


Frequency of URIs With Given Memento Count Among Tweets With Keyword 'nasa'
Number of URIs With Memento Count (LOG Scale)
10^3
10^2
10^1
839
131
76
154
21
11
10
93
0
1
2
3-10
11-15
16-20
21-25
25+
Mementos