Assignment 5

Fall 2014
CS595 Web Science
Dr. Michael Nelson

Mathew Chaney

October 14, 2014

Contents

List of Figures

Listings

1 Question 1

1.1 Question

Explore the friendship paradox for your Twitter account. Since Twitter has directional links (i.e., "followers" and "following"), we'll be investigating if the people you follow (Twitter calls these people "friends") follow more people than you. If you are following < 50 people, use my twitter account "phonedude_mln" instead of your own.

Create a graph of the number of friends (y-axis) and the friends sorted by number of friends (x-axis). (The friends don't need to be labeled on the x-axis as "Bob", "Mary", etc. -- just 1, 2, 3 ...) In other words, if you have 100 friends your x-axis will be 1..101 (100 + you), and the y-axis value will be number of friends that each of those friends has. The friend with the lowest number of friends will be first and the friend with the highest number of friends will be last.

Do include yourself in the graph and label yourself accordingly. Compute the mean, standard deviation, and median of the number of friends that your friends have.

The appropriate part of the Twitter API to use is:

https://dev.twitter.com/rest/reference/get/friends/list

1.2 Answer

Using Dr. Michael Nelson's Twitter account and the Twitter API[1], specifically the GET friends/list[2] request, all of Dr. Nelson's Twitter friends were obtained and saved to the file called `friends`. This method also uses the API's paginating scheme: when there are a large number of results for a query, the API will send a cursor index to show that there are more results to process and that more requests are needed. The code to do this is in Listing 1.

```
37 def get_friends(screen_name):
38     print("Getting {}'s friends".format(screen_name))
39     friends = []
40     next_cursor = -1
41     limit, reset = get_limit()
42     while True:
43         if limit == 0:
44         limit, reset = wait_for_reset(reset)
45         response = requests.get(SEARCH_URI, params={'screen_name':screen_name,
46                     'cursor':next_cursor, 'count':200}, auth=OAUTH)
47         if not response:
48             print("Bad response: {}".format(response.reason))
49             return []
50         limit = limit - 1
51         data = json.loads(response.text)
52         print("Got {} friends, limit: {}".format(len(data['users']), limit))
53         next_cursor = data['next_cursor']
54         friends.extend(data['users'])
55         if next_cursor == 0:
56             return [friend['screen_name'] for friend in friends]
```

Listing 1: Getting the friends list

To reduce the impact of high HTTP traffic, the Twitter API rate-limits most requests – the one needed to obtain a user's friends list has a limit of fifteen message per fifteen minutes. Any requests received from a user or service that has reached the limit will be denied. To ensure no HTTP requests are sent after the limit has been reached the script will sleep until the limit resets. This is accomplished using Python's `time` package[3] and the methods shown in Listing 2.

```
24 def get_limit():
25     response = requests.get(LIMIT_URI, params={'resources':'friends'}, auth=OAUTH)
26     data = json.loads(response.text)
27     return data['resources']['friends']['/friends/list']['remaining'],
28            data['resources']['friends']['/friends/list']['reset']
29
30 def wait_for_reset(reset):
31     naptime = reset - time.time() + 5
32     print("Limit reached, sleeping for {}".format(naptime))
33     time.sleep(naptime)
34     print("Time to get up and go to work") # https://www.youtube.com/watch?v=ChP6SfmZVSE
35     return get_limit()
```

Listing 2: Waiting for API request limit reset

The `get_limit` method uses the API to find the number of available requests remaining for the GET friends/list method and also the time at which the limit will reset, received as seconds since the Unix epoch[4]. This method, combined with the `wait_for_reset` method, allowed the script to restart after an interruption and only require waiting for the appropriate amount of time. The sleep time was extended by 5 seconds to allow for a small buffer in case of mathematical errors.

The friends of Dr. Nelson's friends were then obtained with the same `get_friends` method from Listing 1 and stored in a file called `friend_counts`, each on a single line preceeded by their friend count. All of these operations were controlled by a main method, which is shown in Listing 3.

```
58 if __name__ == '__main__':
59     # get list of MLN's friends from previously run get_friends method
60     with open('friends') as infile:
61         friends = [line.strip() for line in infile]
62
63     # initialize map of MLN's friends' friend counts
64     friend_counts = {friend:0 for friend in friends}
65
66     # get MLN's friends' friend counts which have already been processed
67     with open('friend_counts') as infile:
68         for line in infile:
69             count, friend  = line.strip().split(' ')
70             friend_counts[friend] = count
71
72     # process the rest
73     for friend, count in friend_counts.iteritems():
74         if count > 0:
75             friends.remove(friend)
76     with open('friend_counts', 'a') as outfile:
77         for friend in friends:
78             friend_list = get_friends(friend)
79             if friend_list is None:
80                 continue
81             outfile.write('{} {}\n'.format(len(friend_list), friend))
```

Listing 3: Main method

The `friend_counts` file was ordered in place with the Unix command in Listing 4.

```
[mchaney@mchaney-l a5]$ cat friend_counts | sort -g -o friend_counts
```

Listing 4: Sort command

This file was then processed by the R script shown in Listing 5 to produce the graph in Figure 1

```
#! /usr/bin/Rscript

# read data
data <- read.table('friend_counts')
x <- seq(1, length(data$V1))
y <- data$V1

# calculate statistical values
mln_idx <- grep("phonedude_mln", data$V2)
med_val <- median(data$V1)
med_idx <- which(abs(y - med_val) == min(abs(y - med_val)))
mean_val <- mean(data$V1)
mean_idx <- which(abs(y - mean_val) == min(abs(y - mean_val)))
std_dev <- sd(data$V1)

# draw the graph
pdf("friend_plot.pdf")
plot(x, y, type="l", log="y", pch=19, main="Dr. Nelson's Friends' Friends",
    ylab="Number of Friends", xlab="Index of Friend")

# illustrate points of interest
abline(h=data$V1[mln_idx], col="red")
abline(h=data$V1[med_idx], col="blue")
abline(h=data$V1[mean_idx], col="darkolivegreen3")
abline(h=mean_val + std_dev, col="purple")

legend(x=100, y=5, c(paste("Nelson: ", data$V1[mln_idx]),
    paste("median: ", med_val),
    paste("mean: ", format(round(mean_val, 4), nsmall = 4)),
    paste("+std dev: ", format(round(mean_val + std_dev, 4), nsmall = 4))),
    cex=0.8, col=c("red", "blue", "darkolivegreen3", "purple"), lty=c(1, 1))
dev.off()
```

Listing 5: Graph Creation Script

The median, mean and standard deviation were all calculated and plotted on the graph. Only a single line was drawn for the standard deviation because the lower-end value was negative, and thus off the graph.

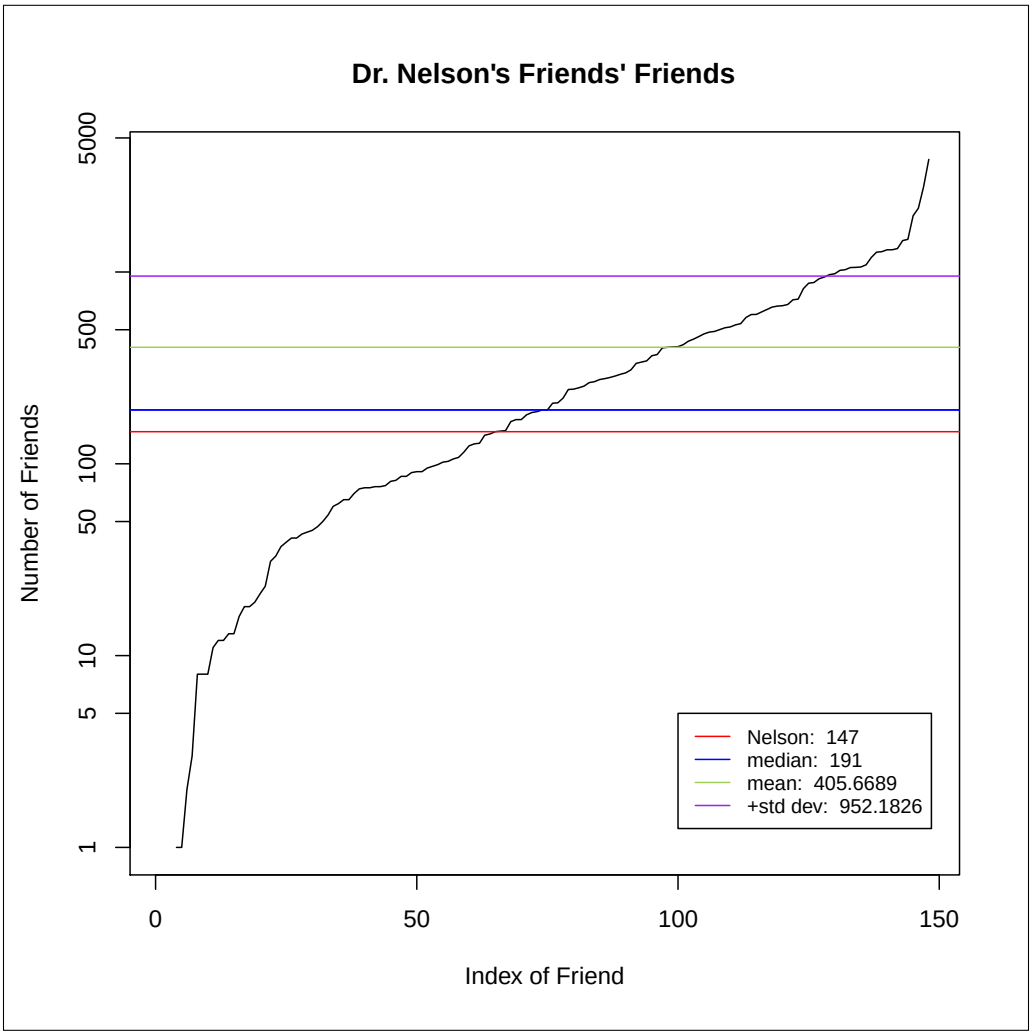

Figure 1: The Friendship Graph

2 References

[1] Twitter, Inc. Twitter API: Overview. https://dev.twitter.com/overview/api/, 2014.

[2] Twitter Inc. Twitter API: GET friends/list. https://dev.twitter.com/rest/reference/get/friends/list/, 2014.

[3] The Python Software Foundation. Python time module. https://docs.python.org/2/library/time.html, 2014.

[4] Community Wiki, Stack Overflow. Why is 1/1/1970 the "epoch time"? http://stackoverflow.com/questions/1090869/why-is-1-1-1970-the-epoch-time, Last edited June 23, 2011.