CS896 Introduction to Web Science
Fall 2013
Report for Assignment 5

Corren G. McCoy

October 17, 2013

Contents

List of Figures

List of Tables

1 Question 1

1.1 Problem

The "friendship paradox" (`http://en.wikipedia.org/wiki/Friendship_paradox`) says that your friends have more friends than you do. Determine if the friendship paradox holds for your Facebook account. 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.) 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.

1.2 Response

For this problem, we used the XML file associated with Dr. Nelson's Facebook account. The file was saved in the graphML format. The social network data of interest was found in the *node* tag as shown in the snippet below. We used Python to parse the XML to locate all of the *node* tags. For each node, we then queried each of the enclosed *data* tags until we located one with an attribute of "friend_count." If successful, we extracted the value of friend_count from the character data (i.e., CDATA) of the tag. If the attribute was not present, the friend_count for that *node* was set to zero rather than excluding the *node*. This was done so we could correctly determine the number of friends associated with the owner of the network.

```
<node id="Simeon_Warner_428351">
    <data key="Label">Simeon Warner</data>
    <data key="uid"><![CDATA[428351]]></data>
    <data key="name"><![CDATA[Simeon Warner]]></data>
    <data key="mutual_friend_count"><![CDATA[13]]></data>
    <data key="friend_count"><![CDATA[244]]></data>
</node>
```

The source code for the XML parser is shown in Appendix A. To facilitate a graphical analysis, the output from the parser was saved to text file (i.e., paradox.txt) consisting of a sequential ID number and the friend count. To analyze the network, we created a function in "R" which performed the following tasks:

- Accept a parameter to determine the dataset to represent on the graph;
- Read the network data from the text file created by the XML parser;
- Calculate the mean, median, and standard deviation of the dataset; and
- Draw and annotate the resulting graph.

The source code for the "R" function is shown in Appendix D. From the graph shown in Figure 1, we can see that the "friendship paradox" holds for the Facebook account. For this particular network, the owner has 165 friends, denoted by the purple marker, which is smaller than the median or typical value of 244 friends denoted by the red marker. The high standard deviation of 370 is consistent with the dispersion that we see on the right side of the graph.

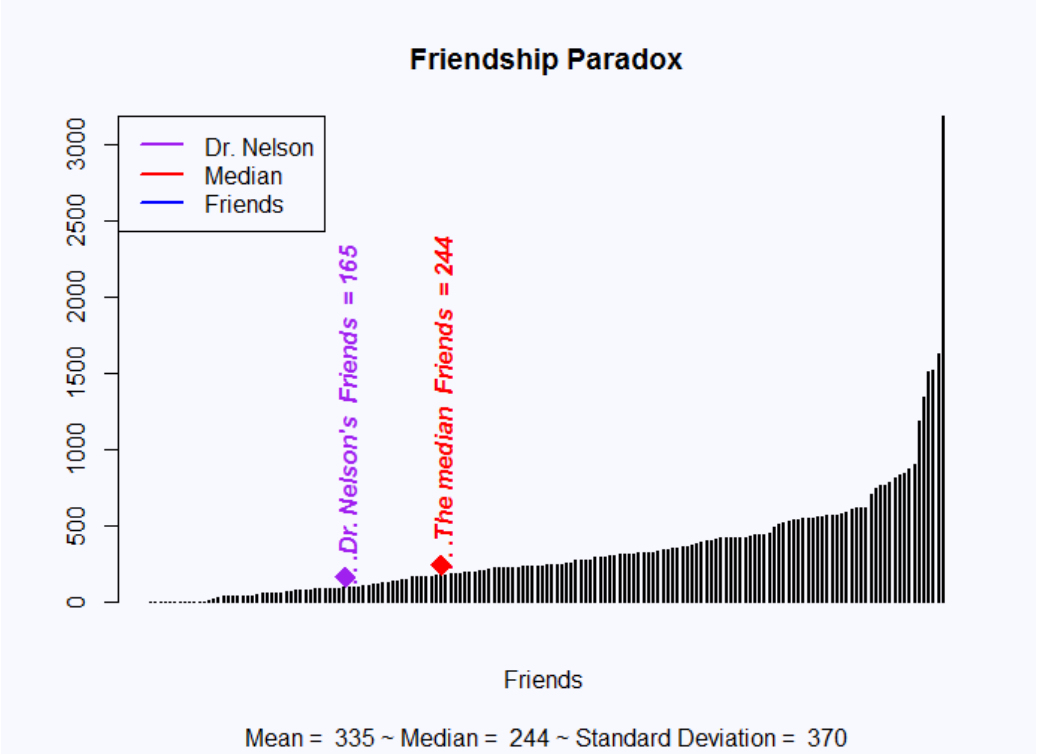

Figure 1: Friendship Paradox - Facebook

2 Question 2

2.1 Problem

Determine if the friendship paradox holds for your Twitter account. Since Twitter is a directed graph, use "followers"as value you measure (i.e., "do your followers have more followers than you?"). Generate the same graph as in question #1, and calculate the same mean, standard deviation, and median values.

2.2 Response

For this problem, we used Dr. Nelson's Twitter account (i.e., @phonedude_mln). His profile summary, shown in Figure 2, provides a means of verifying our results as of the date we performed our calculations. We used Python to access the Twitter API, GET followers/list[1], which returns information about the specified user. The user object returned by the API includes a followers_count which indicates "the number of followers this account currently has." The Tweepy Python package [2] provides a wrapper for all of the Twitter API methods. We used the constructs in Tweepy to traverse the list of followers and subsequently obtain each of their followers_counts. The source code for Python is shown in Appendix B. The output from the Twitter was saved to text file (i.e., followersFile.txt) consisting of a sequential ID number, the followers count, and the user_name.

For graphical analysis, we used the same "R" function, Appendix D, to produce the graph shown in Figure 3. For reusability of code, each data file used by the "R" script must consist of an ID and COUNT column. The "friendship paradox" does not hold for Twitter followers. The 204 followers of @phonedude_mln is slightly above the median of 192.5 for each of his followers. The wide swing in the standard deviation is most likely due to the outliers who have followers in the range of 8 to

10,000. These outliers include @klischk with 10,128, @bfluzin with 10,040, and @albakhakh with 9,385 followers respectively.

Figure 2: Dr. Nelson's Twitter Profile Summary

3 Question 3 Extra Credit

3.1 Problem

Repeat question #1, but with your LinkedIn profile.

3.2 Response

Not attempted.

4 Question 4 Extra Credit

4.1 Problem

Repeat question #2, but change "followers" to "following". In other words, are the people I am following following more people?

4.2 Response

We used the same processing as noted for question 2, except instead of extracting the followers_counts, we extracted the friends_ count instead. The Twitter API indicates that friends_count represents "the number of users this account is following (AKA their "followings")." The Python source code is shown in Appendix C. Once again, we saved the output from Twitter to a text file (i.e., friendsFile.txt) consisting of a sequential ID number, the friend count, and the user_name.

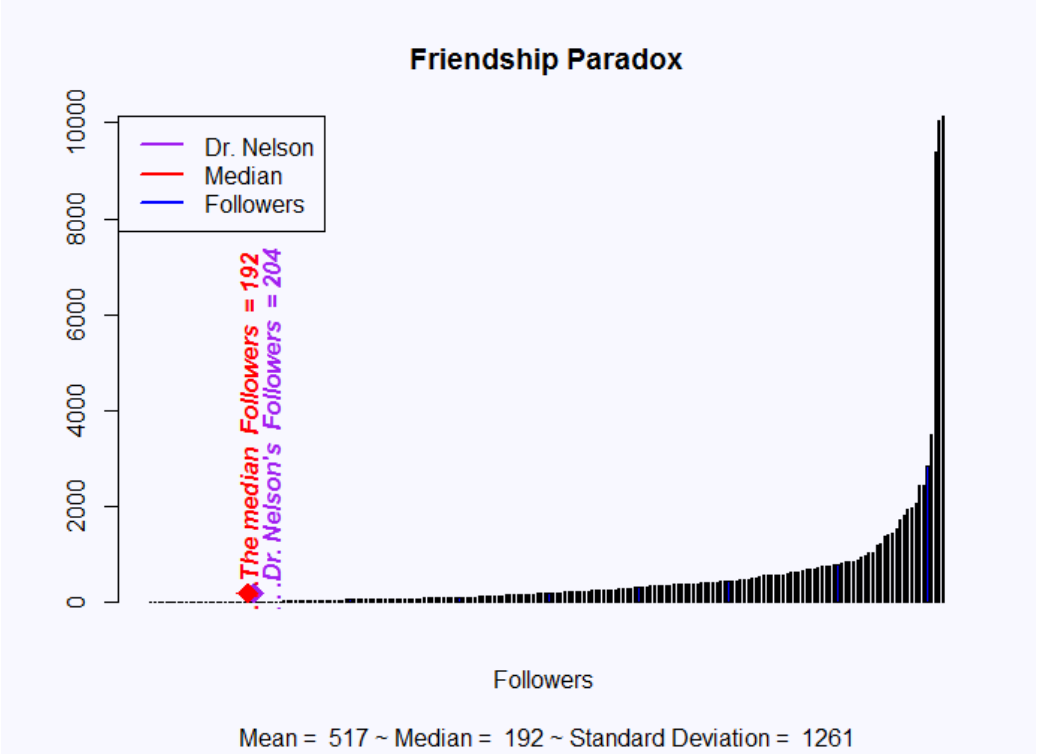

Figure 3: Friendship Paradox - Followers on Twitter

For graphical analysis, we used the same "R" function, Appendix D, to produce the graph shown in Figure 4. The "friendship paradox" holds for the users @phonedude_mln is following. His following list consists of 73 users which is well below the median of 199 users for those he is following. The standard deviation shows TBD. The outlier is @smalljones who is following 3,341 users.

[1] https://dev.twitter.com/docs/api/1.1/get/followers/list
[2] http://code.google.com/p/tweepy/

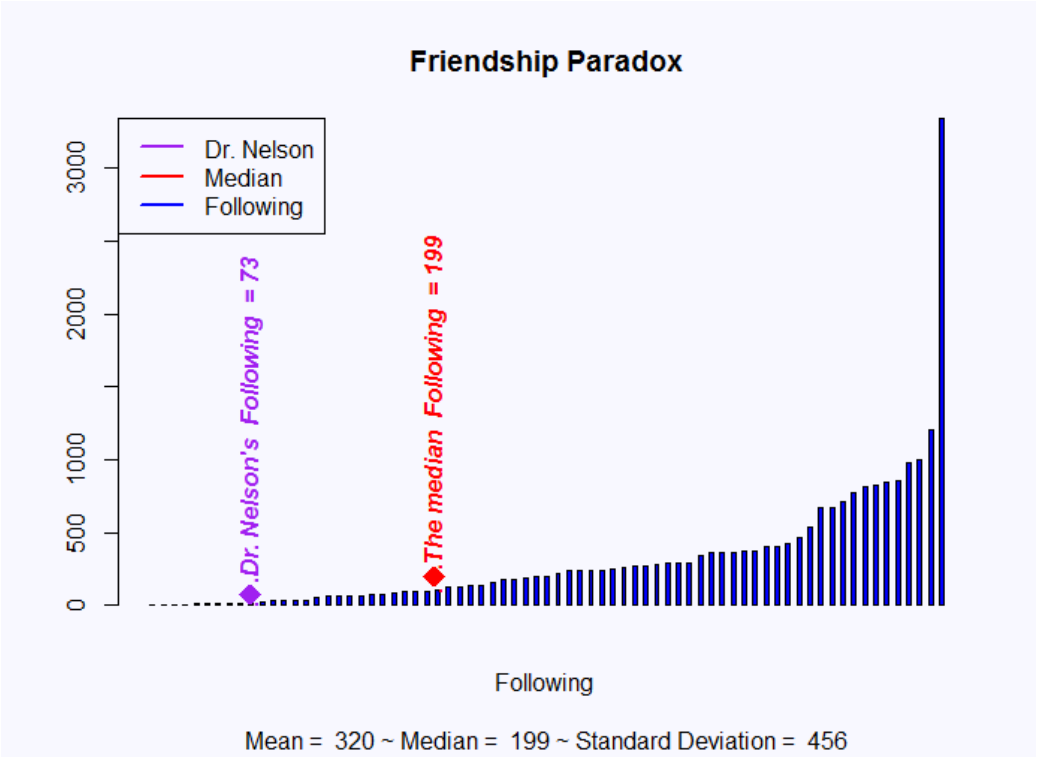

Figure 4: Friendship Paradox - Following on Twitter

Appendix A

Python Source - paradox.py

```
#import easy to use xml parser called minidom:
from xml.dom.minidom import parseString
from xml.dom import minidom
from xml.dom import pulldom



# Dr. Nelson's facebook network
file = open('C:/Python27/myFiles/Assignment 5/mln.graphml', 'r')
# convert to string
data = file.read()
# close the file
file.close()

#parse the xml downloaded from Facebook
dom = parseString(data)

# initialize the output objects
graphData={}
mlnFriendCount = 0

# Initialize the dot file
friends = open('C:/Python27/myFiles/Assignment 5/paradox.txt', 'w')
# Write the header
friends.write("ID" +"," + "FRIEND.COUNT")

#retrieve the first xml tag (<tag>data</tag>) that the parser finds with name "node":
for node in dom.getElementsByTagName("node"):
    try:
        # keep track of MLNs friends
        mlnFriendCount = mlnFriendCount + 1
        nodeData = node.getElementsByTagName('data')
        for subNode in nodeData:
            friendCount = 0
            thisKey = subNode.getAttribute("key")
            if thisKey == "friend_count":
```

```
                subNode=subNode.toxml()
                #data key="friend_count"><![CDATA[421]]></data>
                #strip off the tag (<tag>data</tag>  --->   data):
                cData=subNode.replace('<data key="friend_count">','').replace('</data>','')
                #friendCount is extracted from <![CDATA[39]]>
                friendCount = cData.replace('<![CDATA[','').replace(']]>','')
            else:
                # My friend doesn't have any Facebook friends
                friendCount = 0
            graphData[mlnFriendCount] = friendCount
    except KeyboardInterrupt:
         print ""
         print "processing terminated."
         sys.exit(0)


#graph data is: friend#, friendCount
for key,value in graphData.iteritems():
    friends.write("\n")
    friends.write(str(key) + "," + str(value))
    # debug line
    print key, value
# Close the output file
friends.close()
```

Appendix B

Python Source - getFollowers.py

```
'''
@Author Corren McCoy, 2013
@Purpose
Use the Twitter APIs to extract:
(1) the followers (count) of a certain user including the followers
of those entities
(2) the following (count) of a certain user including the number of
accounts his followers are following.
(3) the friends (count) of a certain user include the number friends
associated with his followers
Note:  The Twitter API indicates that following is otherwise known as their "friends"

Example request:
https://api.twitter.com/1.1/followers/list.json?
+ cursor=-1&screen_name=sitestreams&skip_status=true&include_user_entities=false

This code leverages tweepy (https://code.google.com/p/tweepy/) which is
a Python API for twitter. It provides a nice wrapper for the Twitter APIs
'''
import tweepy
import time

# Initialize the oAuth authentication settings
# Documentation: https://dev.twitter.com/docs/auth/oauth
CONSUMER_KEY = "LrA1DdH1QJ5cfS8gGaWpOA"
CONSUMER_SECRET = "9AX14EQBLjRjJM4ZHt2kNf0I4G77sKsYX1bEXQCW8"
OAUTH_TOKEN = "1862092890-FrKbhD7ngeJtTZFZwf2SMjOPwgsCToq2A451iWi"
OAUTH_SECRET = "AdMQmyfaxollI596G82FBipfSMhagv6hjlNKoLYjeg8"

# Dr. Nelson's twitter account
USERNAME='phonedude_mln'
# variables used to manage the API rate limit
FIFTEEN_MIN=900 # seconds
FIVE_MIN=300

# first set up authenticated API instance
```

```
auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(OAUTH_TOKEN, OAUTH_SECRET)
#create an instance of a tweepy StreamListener to handle the incoming data
api = tweepy.API(auth, api_root='/1.1')

# open the output file that will contain the data for graphing in "R"
followersFile = open('C:/Python27/myFiles/Assignment 5/followersFile.txt','w')
# write the header line for identification
followersFile.write("ID" +"," +"COUNT" + "," + "FOLLOWER.USERNAME" )

try:
    counter = 0
    for follower in tweepy.Cursor(api.followers, screen_name=USERNAME).items():
            counter = counter + 1
            print follower.screen_name, follower.followers_count
            followersFile.write('\n')
            followersFile.write(str(counter) +',' + str(follower.followers_count)
            +',' + follower.screen_name)

            # Returns the remaining number of API requests available to the requesting user
            # before the API limit of 180 requests every 15 minutes
            # Calls to rate_limit_status do not count against the rate limit.
            remaining_hits = api.rate_limit_status('application')['resources']['application']
            ['/application/rate_limit_status']['remaining']

            if (remaining_hits < 5):# or counter % 160 == 0:
                print '\n'
                print 'You have', remaining_hits, 'API calls remaining in this window.
                Started sleeping at', time.ctime()
                time.sleep(FIFTEEN_MIN)
            else:
                pass
    followersFile.close()
    print '\n'
    print 'You have', remaining_hits, 'API calls remaining in this window.', time.ctime()

except tweepy.error.TweepError as e:
    print e.reason
    followersFile.close()
```

Appendix C

Python Source - getFriends.py

```
'''
@Author Corren McCoy, 2013
@Purpose
Use the Twitter APIs to extract:
(1) the friends (count) of a certain user including the friends of those entities
(2) the following (count) of a certain user including the number of accounts
his followers are following.

Example request:
https://api.twitter.com/1.1/followers/list.json?cursor=-1
+ &screen_name=sitestreams&skip_status=true&include_user_entities=false

This code leverages tweepy (https://code.google.com/p/tweepy/) which is a
Python API for twitter. It provides a nice wrapper for the Twitter APIs
'''
import tweepy
import time

# Initialize the oAuth authentication settings
# Documentation: https://dev.twitter.com/docs/auth/oauth
CONSUMER_KEY = "LrA1DdH1QJ5cfS8gGaWpOA"
CONSUMER_SECRET = "9AX14EQBLjRjJM4ZHt2kNf0I4G77sKsYX1bEXQCW8"
OAUTH_TOKEN = "1862092890-FrKbhD7ngeJtTZFZwf2SMjOPwgsCToq2A451iWi"
OAUTH_SECRET = "AdMQmyfaxollI596G82FBipfSMhagv6hjlNKoLYjeg8"

# Dr. Nelson's twitter account
USERNAME='phonedude_mln'
# variables used to manage the API rate limit
FIFTEEN_MIN=900 # seconds
FIVE_MIN=300

## first set up authenticated API instance
auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(OAUTH_TOKEN, OAUTH_SECRET)
#create an instance of a tweepy StreamListener to handle the incoming data
api = tweepy.API(auth, api_root='/1.1')
```

```
# open the output file that will contain the data for graphing in "R"
friendFile = open('C:/Python27/myFiles/Assignment 5/friendsFile.txt','w')
# write the header line for identification
friendFile.write("ID" + "," + "COUNT" +"," + "FOLLOWING.USERNAME")

try:
    counter = 0
    for friend in tweepy.Cursor(api.friends, screen_name=USERNAME).items():
            counter = counter + 1
            print friend.screen_name, friend.friends_count
            friendFile.write('\n')
            friendFile.write(str(counter) +',' + str(friend.friends_count)+ ',' +
            friend.screen_name)

            # Returns the remaining number of API requests available to the requesting user
            # before the API limit is reached for the current hour.
            # Calls to rate_limit_status do not count against the rate limit.
            remaining_hits = api.rate_limit_status('application')['resources']['application']
            ['/application/rate_limit_status']['remaining']
            #remaining_hits=api.rate_limit_status['remaining_hits']

            if (remaining_hits < 5): #or counter % 160 == 0:
                print '\n'
                print 'You have', remaining_hits, 'API calls remaining in this window.
                Started sleeping at', time.ctime()
                time.sleep(FIFTEEN_MIN)
            else:
                pass
    friendFile.close()
    print '\n'
    print 'You have', remaining_hits, 'API calls remaining in this window.', time.ctime()

except tweepy.error.TweepError as e:
    print e.reason
    friendFile.close()
```

Appendix D

R Source - paradox.r

```
paradox<- function(graphType) {
  # Graphical analysis of the friendship paradox
  # input is either:
  # 1 - the friend_count associated with a Facebook account
  # 2 - the followers count associated with a Twitter account
  # 3 - the following (friend) count associated with a Twitter

setwd("C:/Users/Corren/Documents/My R Scripts")
# Ignore the header line
switch(graphType,
      G1 = {data<-read.csv("paradox.txt", head=TRUE)
            what<-"Friends"},
      G2 = {data<-read.csv("followersFile.txt", head=TRUE)
            what<-"Followers"},
      G3 = {data<-read.csv("friendsFile.txt", head=TRUE)
            what<-"Following"},
      stop("valid graph types are: (1) Facebook (2) Followers (3) Following")

)

# coerce to numeric. column 2 is the FRIEND.COUNT
data[,2] <- as.numeric(data[,2])

# sort by the number of friends, followers, following
sortedData<- data[order(data[,2]),]
# mean
meanData<-round(mean(data[,2], na.rm=TRUE), digits=0)
print(paste("Mean = ", meanData))

# standard deviation
sdData<-round(sd(data$COUNT, na.rm=TRUE), digits=0)
print(paste("Standard Deviation = ", sdData))

# how many records in the file (i.e., friends, followers, following)
myDataCount <- nrow(data)
```

```
# median
medianData<-round(median(data[,2], na.rm=TRUE), digits=0)
print(paste("Median = ", medianData))

# Trim off excess margin space (bottom, left, top, right)
#par(mar=c(0.2, 0.2, 0.2, 0.2), bg="grey")
par(bg="ghostwhite")

labelMean<-paste("Mean = ", meanData )
labelMedian<-paste("Median = ", medianData )
labelSD<-paste("Standard Deviation = ", sdData )

# Put calculations in the footer of the graph
footer<- paste(paste(labelMean, labelMedian, sep=" ~ "), labelSD, sep= " ~ ")

barplot(sortedData$COUNT, width=3,space=1.5,
      main="Friendship Paradox",
      sub=footer,
      ylab="",
      xlab=paste(what,"\n"),
      col="blue"
     )

#abline (v=myDataCount, col="purple", lwd="2")
textloc <- myDataCount
textlabel <- paste(paste(paste(". . .Dr. Nelson's ", what), " ="), myDataCount, sep=" ")
# use the value of the "count" to determine where to print the text
text(textloc, textloc+(max(sortedData$COUNT) / 3), textlabel, col = "purple", srt=90, font=4)
points(textloc, textloc, pch=18, cex=2,col="purple")


# add a vertical line for my count
#abline (v=medianData, col="red", lwd="2")
textloc <- medianData
textlabel <- paste(paste(paste(". . .The median ", what)," ="), medianData, sep=" ")
# use the value of median to determine where to print the text
text(textloc, textloc + (max(sortedData$COUNT) / 3), textlabel, col = "red", srt=90, font=4)
points(textloc, textloc, pch=18, cex=2,col="red")

legend("topleft",
legend=c('Dr. Nelson', 'Median', what),
col=c('purple', 'red', 'blue'), lwd=2, lty=c(1,1,1))

}
```