2019

@yourlocation: A Spatial Analysis of Geotagged Tweets in the US

Ocean McKinney

@yourlocation:

A Spatial Analysis of Geotagged Tweets in the US

A Thesis Presented

by

A. Ocean McKinney

To the Keck Science Department

Of Claremont McKenna, Pitzer, and Scripps Colleges

In partial fulfillment of

The degree of Bachelor of Arts

Senior Thesis in Physics

Fall 2019

Table of Contents

Acknowledgements

I would like to thank Dr. Adam Landsberg and Dr. Mike Izbicki for their guidance and resources for this project. Thanks also to my network research team, Melody Sue, Katherine Tully, and Kyle Schuster. I would also like to thank my partner for her patience and support throughout my research.

Abstract

This project examines the spatial network properties observable from geo-located tweet data. Conventional exploration examines characteristics of a variety of network attributes, but few employ spatial edge correlations in their analysis. Recent studies have demonstrated the improvements that these correlations contribute to drawing conclusions about network structure. This thesis expands upon social network research utilizing spatial edge correlations and presents processing and formatting techniques for JSON (JavaScript Object Notation) data.

Chapter 0: Introduction

0.1 Networks

A network, also known as a graph, is a space occupied by points which may or may not be connected to each other in a particular order. A node (also called a vertex, site, member, or actor, depending on the field of study) is modeled by a single point in a network. It can represent a person, a neuron, a landmark, and many other things that can be connected to another node in space. The connections between nodes are known as edges (also called bonds, links, or ties). Edges may represent a cable between two devices, a friendship, a wave, or many other things that could connect two or more nodes together. It's worth noting that edges cannot be formed in the absence of nodes. In most cases, at least two nodes are required to be present for an edge to be formed. When an edge is found connecting one node to itself, it's called a *self-edge*. Self-edges have no direction and usually appear as a loop in a modeled network [7].

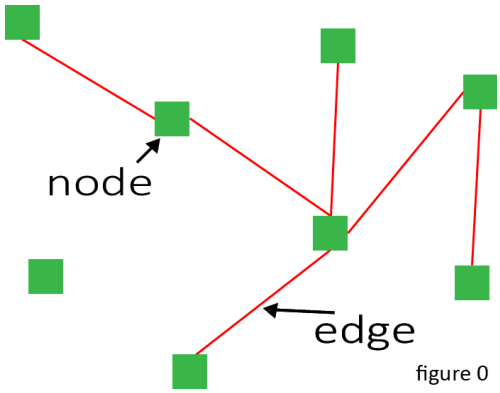


figure 0

Edges in a network may be unidirectional, indicating a one way transfer of something. Think of a network of gutters along the roads. Water goes in, but doesn't come out. This is known as a *directed network.* Alternatively, edges may be bidirectional, allowing free transfer between nodes. Networks consisting of these edges are known as *undirected networks* [7]. Relationship networks generally work this way, allowing bidirectional communication between pairs of friends.

Edges can also carry a weight. In a *weighted network*, edges may be associated with different values. The value could represent the frequency a path is used between two nodes or the strength of the connection. In an *unweighted network*, all edges are equal in value [7]. The twitter data analyzed in this project was parsed into an undirected, unweighted network to reduce the sample size.

0.2 Network Properties

Certain trends and tendencies may be analyzed to make predictions as to how a network might behave. These properties describe how efficient a network is at keeping all of the members of the network connected. Some of the most commonly used metrics include: average degree, characteristic path length, and clustering coefficients.

0.2.1 Degree

One of the simplest things that can be measured from a graph is the *degree* of a node. The degree is the number of edges an individual node has connecting it to other nodes within the same network [7]. Nodes with a high degree (also called popular nodes) often act as hubs that provide a lot of structure for the network. In Figure 2, node 3 acts as a hub, being directly connected to the most nodes.

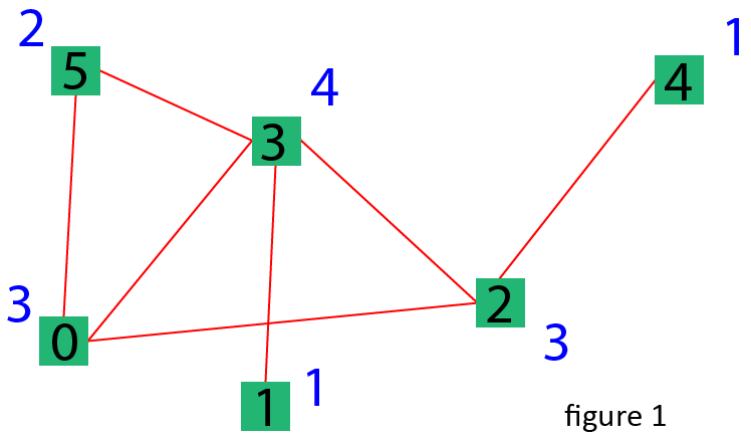


figure 1

Calculating the average degree of a network, by dividing the total number of edges by the number of nodes in the network, yields a good estimate for the number of edges attached to a particular node chosen at random. In Figure 2, the average degree would be (3+1+3+4+1+2)/6 or 2.33.

0.2.2 Path Length

Another useful measure worth noting is the *path length* or number of sites en route to reaching a target site. An example of this would be the number of steps required for a postcard to travel from you to your friend. A path length is called *geodesic* if it represents the shortest distance between two sites [8].

Applied to the entire network, calculating the average geodesic path length from a site to all of the others is called the closeness centrality. This describes how close or central a site is relative to other sites in the network [7].

0.2.3 Transitivity

Transitivity is one of the most important properties in social network analysis. It describes the relationship between connected nodes also known as *neighbors.* Similar to the transitive property in mathematics, transitivity describes a rule between neighbors: if a > b and b > c, then a > c. In terms of application, this metric refers to friends of friends also being considered friends [7].

0.2.3.1 Clustering Coefficient

A measure of transitivity describing the connectedness of a network is known as the *clustering coefficient.* This can be thought of as the number of closed triangles divided by the number of paths of length 2 in a network [7]. Perfect transitivity implies C =1, meaning all neighbors are connected to each other. For simplicity, this metric can also be defined as the average of all local clustering coefficients in a network. C_i in the following equation relies on the local clustering coefficient defined in the next section.

$$C = \frac{\sum_{1}^{n} C_i}{n}$$

0.2.3.2 Local Clustering Coefficient

The clustering coefficient describing the property of a single vertex is called the *local clustering coefficient*. In the below equation, the local clustering coefficient can be represented relative to the degree (*D*) and number of links between neighbors (*L*) of a vertex (*i*) [7].

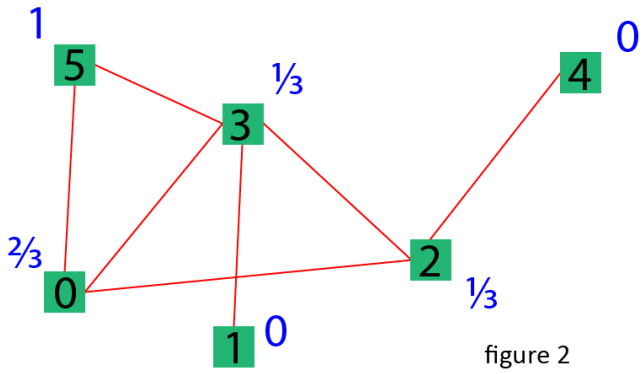


figure 2

$$C_i = \frac{2 \cdot L_i}{D_i(D_i - 1)}$$

In figure 2, the clustering coefficient of node 2 is calculated as 2*(1)/(3*(3-1)) = 2/6 = ⅓. The global clustering coefficient would be the average of all of the blue values, 7/18 or 0.389.

0.3 Network Representations

There are many different ways to represent a network for the purpose of analysis. Networks with high edge volume, like figure 5, tend to work best with an *adjacency matrix*. This looks like a spreadsheet of 1's and 0's (figure 3) for an unweighted network representing whether or not an edge exists between two nodes. Sparser networks can be examined through *adjacency lists,* like figure 4, with each row representing a connection between a pair of nodes [7].

	0	1	2	3	4	5
0	0	0	1	1	0	1
1	0	1	0	1	1	0
2	1	0	0	1	0	0
3	1	1	1	0	0	1
4	0	0	1	0	0	0
5	1	0	0	1	0	0

figure 3

node	connections
0	2,3,5
1	1,3
2	0,3,4
3	0,1,2,5
4	1
5	0,3

figure 4

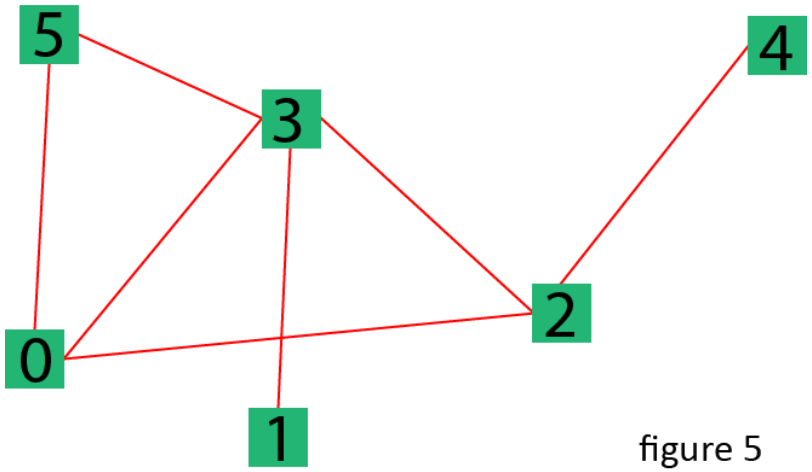


figure 5

Chapter 1: Spatial Networks

Most observable networks occupy defined spaces in the physical world. We call networks 'spatial' when they contain nodes and edges representing physical locations. This usually takes the form of a two or three-dimensional graph. Restaurant franchise locations each have addresses corresponding to geographical coordinates while brain data can be mapped as a 3d mesh organized by neural regions. The spatial information of a network allows for a deeper analysis of the relationship between nodes. For instance, if two nodes are connected to each other, it would be useful to know how physically far apart they are when assessing the meaning of their relationship. The distance between the connected nodes could represent the difference in the type of connection that exists between two friends that live in the same neighborhood and penpals living in different countries.

1.1 Locality

Spatial networks tend to be composed of nodes that are varying distances apart from each other. A node's neighbor can be close to (local) or far away from (global) a node in space. The *locality* describes whether a node is local or global [7]. This property is observed most commonly in social networks where you may be friends with someone who is physically close to or farther away from you. Locality is defined by a cutoff distance, usually defined by the researcher, which sets a maximum range for a neighbor to be considered local. Connected nodes outside of a cutoff distance are known as global neighbors.

1.2 Geo-Tagged Twitter Network

The data for this research comes from research conducted by Mike Izbicki & his colleagues in 2019. The full set of data contains extended information for individual tweets, consisting of: user ID, tweet ID, number of followers, number following, number of user lifetime tweets, declared user location, geo-tagged user location, tweet text, mentioned user ID, and verified status, to name a few [6].

Twitter presents a unique opportunity for data research due to the platform's widespread use and extensive data collection accessibility. Other social network platforms don't permit spatial data collection or don't record their data at all. As a spatial network, geo-tagged Twitter data can be analyzed through the vast defining properties of user relationships. Although user mention relationships are examined in this research, the same data could be used to analyze retweets relationships or the content of user tweets.

1.3 Background

As data continues to expand and networks become more prevalent, the tools we use to assess these structures must also become more robust. Standardized metrics are calculated to analyze and group networks using some of the documented characteristics described in Chapter 0. Although most network research draws conclusions based on the organization of nodes in relationship to each other, few studies factor the precise location of individual nodes in the network. Geo-tagged Twitter data allows researchers to explore the possibilities of spatialized metrics. I specifically chose to look at the relationship of mentions to fabricate a network due to their high level of engagement. Users receive a

notification when they are mentioned, or @ed, in a tweet [3]. In other social interactions on Twitter, a user may send a tweet out to their followers, but due to the nature of Twitter's complex algorithms and variations in the amount of time a user spends on the platform, a user's followers may not see their tweet. A similar shortcoming of retweets, when a user essentially quotes what another user has previously tweeted, is that the quoted 'tweeter' may not be aware of all of the specific users that have retweeted their tweet.

Examining users and their mentions, permits a more accurate representation of a connection between users than retweets or followers because of the way the function was designed as well. To @ someone on twitter is an abstraction of yelling out someone's name before saying something in a crowd. Other users are able to observe the post as public information and the mentioned user is alerted to the call.

As more companies begin to collect data, whether it be to analyze the effectiveness of store locations or to observe trends of their users, spatial metrics will become vital to understanding the nature of the relationships that exist in forming networks. I hope that through this research, the value of employing spatial metrics on modern data is emphasized.

Chapter 2: Preprocessing

As this research involves the analysis of a large set of data, few materials were necessary to obtain results. The network comes from a set of twitter user data of over 2 billion tweets provided by Professor Michael Izbicki on the Lambda servers at CMC [6]. In order to access and parse the tweets into meaningful data, Putty, a free SSH platform was used to log on to the servers while Python code was run to process the data. Python was the language of choice for data manipulation due to its ease of use on a linux based system and its extensive online documentation. Other programs utilized include Microsoft Excel and operating system command terminals.

2.1 Original Data Structure

The original tweet data was stored on a linux server in 667 zipped files, organized by date, each containing 24 JSON files organized by the hour they were retrieved. The majority of information from each tweet was unnecessary for the creation of a spatialized network. For the purpose of this study, a select number of specific fields were selected to reduce the sample size, increase the speed of calculations, and later ensure the legitimacy of tweets. The chosen fields of interest include: user ID, number of followers, number of lifetime tweets, mentioned user ID(s), replied user IDs, tweet coordinates, and user declared location. Below is an example of a single line of JSON formatted data. This particular tweet was captured on March 24, 2018. A few of the fields have been obscured for privacy.

2.2 Parsing

Although all of the accessible data provided some form of geo location information for each of the tweets, not all of the geographic information was credible. Each tweet had geo information recorded either as a point-specific geographical location, indicated by longitude and latitude coordinates, or a bounding box, denoted by a set of coordinates defining an area ranging in size from a landmark to a city [1]. To reduce the size of the network while improving the accuracy of node location, the sample size was restricted to include tweets with a specific coordinate point between the boundaries of the contiguous United States. The bounding values were obtained from openstreetmap.org. Additionally, the tweet language and location declared by each user were extracted to serve as a check on the coordinate data. Finally, the data was filtered to extract only lines that included a mentioned user, “mention”, and/or a response to another user, “reply”, to form the edges between nodes. In the case where a user mentioned multiple accounts in a single tweet, only the first listed mention was recorded to reduce processing time. In total, the initial parsing yielded about 8.5 million lines of data represented in .csv format as sampled below.* The script for this step can be found in Appendix A.

	A	B	C	D	E	F	G
1	userid	coords	decloc	followers	statuses	reply	mention
2	3.3E+09	[-118.2867	Los Angeles,	4406	355		1.4E+08
3	2.3E+07	[-111.8219	Los Angeles,	2388	8446		8.6E+17
4	2.5E+08	[-83.92494	Atlanta, GA	316	3027		2.7E+07
5	1.4E+08	[-76.21064	somewhere	378	7212		1.7E+07
6	3E+09	[-80.19609	Miami, FL	840	1908		5E+07
7	1.7E+07	[-122.6249	Vancouver, C	1335	760		1.8E+09

*The “reply” column contained far fewer values than the mention column (a ratio of about 1:14) indicating that twitter users engage in generally @ing individuals far more frequently than replying directly to a user’s tweet. The “userid”, “coords”, and “mention” columns have been intentionally truncated for user privacy. Also note that the “decloc” field (short for declared location) represents information input by the user.

2.3 Formatting

Once the useable nodes were gathered, there was still the matter of assigning coordinates to the mentioned and replied nodes. The data structure after parsing contained a user ID with coordinate information and mention ID without coordinate data. A separate script was used to collect mentioned and replied ID’s that also occurred as a main user and to assign them point coordinates from the “coords” list. Lines without coordinate data for a mentioned or replied user (linked) were then dropped, reducing the sample size to about 1 million connections. The script for this step can be found in Appendix B. In this state, the data listed each user ID, with its respective mentioned ID, each corresponding to its own coordinate data. For simplicity and comparisons, users connected to replies were output to a separate file from users connected to mentions. Below is an example of the mention output format.

	A	B	C	D	E	F	G
1	userid	coords	decloc	followers	statuses	mention	mentcoords
2	3E+09	[-80.19609	Miami, FL	840	1908	50391248	[-80.2720184,
3	2E+07	[-122.6249	Vancouve	1335	760	1.8E+09	[-122.79679,
4	9E+07	[-118.1350	Fullerton,	201	9403	1.92E+08	[-118.136944
5	2E+07	[-87.8031,	Chicago	104	2934	2.28E+08	[-87.779622,
6	3E+08	[-80.22125	Jensen Be	71	669	2.9E+08	[-80.2212908,

2.3.1 Repeated Data

There are no limitations on the number of times a user may mention another user on Twitter. As a result, Twitter data has the potential to be analyzed as a weighted or unweighted graph. To simplify more intense calculation and threshold the impact of high-engagement users, duplicate edges were deleted from the data using the script from Appendix C. This step reduced the sample of mention connections from about 1 million to 650 thousand and reduced the sample of reply connections from about 100 thousand to 70 thousand. The similar scale of reduction can be interpreted as users engaging with each other for similar lengths of time regardless of whether through a single tweet thread or through entirely separate tweets.

2.3.2 Self @'s

While Twitter users are able to @ other users, they may also @ their own user ID. These self @'s occured in 2% of the reply connections as a result of users responding to one of their own previous tweets and in 3% of the mention connections for reasons yet to be verified. One conjecture is that @ing oneself serves to organize particular tweets for ease of access later on. Searching a username on Twitter returns each instance the username was publicly mentioned [3]. This phenomenon doesn't quite capture the characteristic of a *self-edge* described in Chapter 0 as the message is still accessible to other users. To reduce calculations and further clean the data, part 0 of the script in Appendix E creates a list of the indices where self @'s occur, counts them, and returns a new list to be processed by the second part of the script.

2.3.3 Undirecting

As described in Chapter 0, an undirected network represents edges as a bidirectional relationship. For the Twitter data, this means if user i mentions user j, user i has also been mentioned by user j. Representing tweets as an undirected graph increases the authenticity of the links being described while further reducing sample size. Part 1 of the script from Appendix E uses the coordinate data from part 1 to create a new series of coordinate lists composed of user coordinates that describe the 'i, j relationship.' This step reduced the total sample to about 40 thousand connected users, a substantial, but expected reduction from the previous step. The new lists created were then passed to step 3.

2.3.4 Adjacency Coordinate List

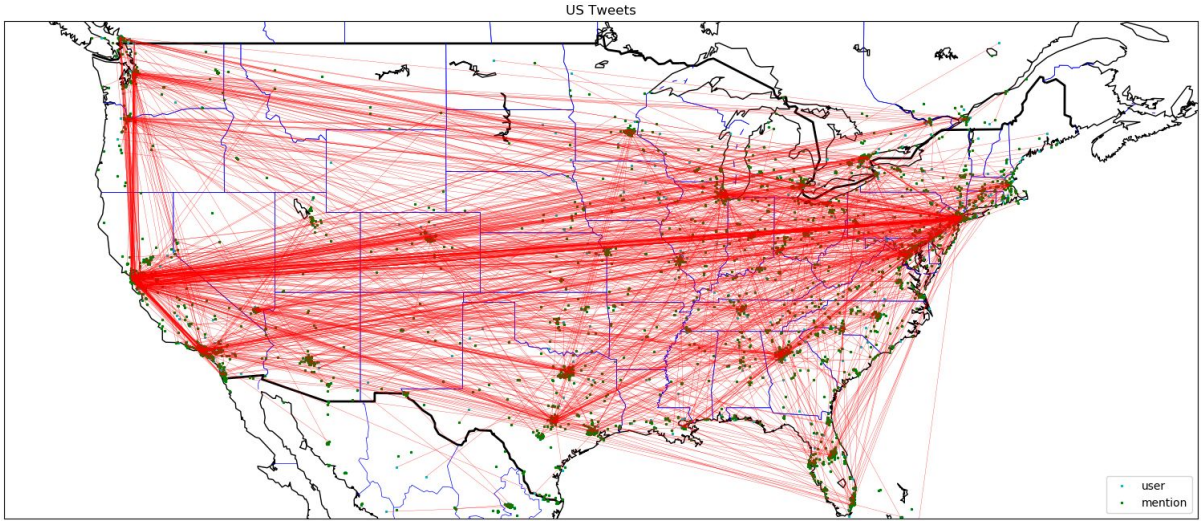
To increase processing speeds during metric calculations, part 2 of the script from Appendix E creates two new lists representing a single 'user' on the same line as all of its respective bidirectional mentions. For this step, I chose to combine the reply and mention data as there were not enough reply relationships remaining to conduct meaningful analysis on them alone. The 'mention' list was appended by a "/" whenever a user re-appeared as an iteration of the user list created in step 2. An example of the final output of these lists is below.

	A	B
4745	-122.32686626, 47.6078199	[-90.06757, 29.95449]
4746	-122.38323932, 37.60040959	[-122.3870945, 37.616713]
4747	-100.31560421, 25.71855064	[-106.48694444, 31.73944444]/[-107.39, 24.7994]/[-116.9632, 32.5037]

The “/” was selected as a place marker to denote the separation between coordinates because of its absence from the original data and to ensure ease of visibility and identification for metric calculations. As an example, the Average degree script from Appendix F is able to quickly count the number of “/”s that appear plus 1 to determine the degree of each node in the network.

2.3.5 Visualization

To help visualize the tweet data, the coordinates were plotted using the basemap toolkit from the matplotlib library [4]. Plotting of the parsed coordinate data ensures that



the network follows expected trends. As you can see from the network above the high concentration of red lines depicts large, tech savvy cities as more densely populated with nodes and edges. The script for obtaining this plot can be found in Appendix D.

Chapter 3: Self Edge Correlation

3.1 SEC Definition

The self edge correlation, or SEC describes the clustering of a node's neighbors. The following formula calculates the probability of two randomly selected neighbors of a node of interest being physically near each other [2].

$$\rho_s(n) \equiv \frac{1+(\Gamma(n))}{|\Gamma(n)|}$$

$|\Gamma(n)|$ represents the number of global neighbors of a node (n) while $(\Gamma(n))$ is the average number of local nodes to a randomly selected neighbor of n.

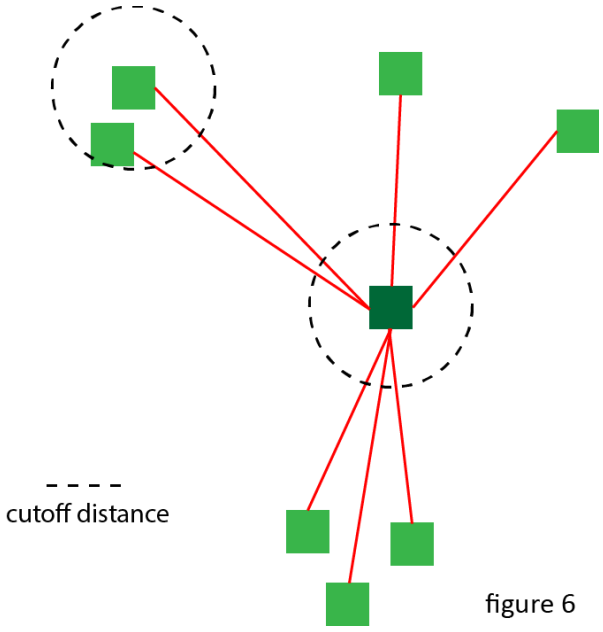


figure 6

The inverse of the SEC for a node, $1/\rho_s(n)$, describes the number of effective 'clusters' that exist global to node *n.* Computation of these metrics on large, edge dense, networks can be extremely processor intense, making the *average SEC* a more reasonable property to employ in spatial network analysis [2]. This process involves calculating the individual SEC's of each node and averaging the resulting values. The SEC calculation in Appendix H utilizes the above formula to calculate the SEC for global neighbors of the network defined by a cutoff distance described in the next section.

3.2 Cutoff Distance Calculation

Defining the cutoff distance for a network is generally arbitrary, though some information can assist in determining its value [2]. I decided to start with a cutoff distance that would be representative of the most common distance the average american was willing to travel, about 16 miles [5]. Though the research determining this average distance was conducted in 2005, it yielded acceptable SEC values with an average of .452. The inverse of this value describes the global neighbor tendency to cluster in 2.2 regions of a particular node. I decided to experiment with a few different cutoff values to replicate the evolution of average work distances in more recent times, but the average SEC for smaller and larger distance cutoffs indicated lower SEC. This could be due to a number of reasons that will be discussed in the next chapter.

Discussion

While working with data that requires multiple steps before reaching a point of analysis, it's useful to have test data to check a scripts' reliability. A few times during this project, my scripts were returning values that appeared to be credible, but upon deeper inspection proved unrepresentative of the intended analysis. Many of the issues were the result of syntax errors or index misalignments when retrieving an item from a list. Sometimes it's impossible to check for all exceptions, especially with large sets of data, but having a team or computer scientist helps to keep things running smoothly.

This project focused heavily on the formatting of data for spatial network analysis and aimed to describe the possibilities of using modern social network data. Though the SEC values were lower than expected, this could be attributed to the sample limitations. Users must opt-in to sharing their locations when sending tweets and there is no guarantee that the user they mention also elects to keep their location public [6]. The complexity of the SEC equation converted to script form in Appendix H might also contain errors as it was the most complicated computation-wise.

Conclusion

The data extraction process for this project took about 2 weeks of processing time to transform billions of tweets from raw JSON format to the ~50k tweets of cleaned data. Script optimization and corrections further reduced the time that could be dedicated to a more thorough analysis of the data. Future work could benefit from experimenting with cutoff distances to produce more accurate clustering behaviors as well as increasing the sample size to include less precise location data.

Network research tools have evolved to take advantage of our increasingly data-driven world. Spatial metrics of US twitter data can more accurately describe the relationships that exist between users based on their physical locations. This information could be used to implement targeted marketing strategies or more accurately locate people of interest. As SEC describe the relationships between the neighbors of a node, one could use this metric to make predictions about the node's future location or the potential locations that new neighbors are likely to appear.

References

1. Developer.twitter.com, 2019. *Introduction to Tweet JSON*--Twitter Developers. Twitter, Inc.

2. Friedman, E., Landsberg, A., Owen, J., Hsieh, W., Kam, L. & Mukherjee P. (2015) *Edge Correlations in Spatial Networks*. IMA Journal of Complex Networks, 1-14.

3. Help.twitter.com, 2019. *About Replies and Mentions.* Twitter, Inc.

4. Jeffrey Whitaker, *Introduction--Basemap Matplotlib Toolkit 1.2.1 documentation* (2016) The matplotlib development team. matplotlib.org/basemap/

5. Langer, G., *A Look Under the Hood of a Nation on Wheels*. (2005) ABC News/Time magazine/Washington Post poll.

6. M. Izbicki, V. Papalexakis, and V. Tsotras (2019) *Geolocating Tweets in any language at any location*. Proceedings of the 28th ACM International Conference on Information and Knowledge Management - CIKM 19.

7. Newman, M.E.J. *Networks: An Introduction.* Oxford: Oxford UP, 2010. Print.

8. Skiena, S. “Shortest Paths.” 6.1 in *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica.* Reading, MA: Addison-Wesley, pp.225-253, 1990.

Appendix

A. Retrieval Script

```
import datetime
import os
from zipfile import ZipFile
from zipfile import BadZipfile
import json
import csv

datadir='/data/tweets'

lon = [-124.848974, -66.885444]
lat = [24.396308, 49.384358]
tnc = 0 #number of nodes

csv_data = open('output.csv', 'w')
wr = csv.writer(csv_data, quoting=csv.QUOTE_ALL)
wr.writerow(['user id', 'coords', 'declared location', 'followers', 'statuses', 'reply', 'mention'])

for filename in os.listdir(datadir):
        print (datetime.datetime.now(),filename)
        try:
             with ZipFile(datadir+'/'+filename,'r') as archive:
                 print(datetime.datetime.now(),filename)
                 for subfilename in archive.namelist():
                     print(datetime.datetime.now(),filename,subfilename)
                     with archive.open(subfilename) as f:
                         ncr = 0 # reply counter
                         ncm = 0 # mention counter
                         for line in f:
                             data = json.loads(line)
                             if 'geo' in data:
                                 if data['geo'] != None and lon[0] <= data['coordinates'][u'coordinates'][0] <= lon[1]:
                                     if lat[0] <= data['coordinates'][u'coordinates'][1] <= lat[1]:

                                         if data['in_reply_to_status_id'] != None: # collects replies

                                             subdata = {k: data[k] for k in ('lang','in_reply_to_user_id',)}
                                             subdata['dec_location'] = data['user']['location']
                                             subdata['coordinates'] = data['coordinates'][u'coordinates']
                                             subdata['id'] = data['user']['id']
                                             subdata['followers_count'] = data['user']['followers_count']
                                             subdata['status_count'] = data['user']['statuses_count']
                                             if data['entities']['user_mentions'] != []:
                                                 subdata['user_mentions'] = data['entities']['user_mentions'][0]['id']
                                             else: subdata['user_mentions'] = None

                                             ncr += 1
                                             dlist = []

                                             if subdata['dec_location'] != None: # handles unicoded strings
                                                    subdata['dec_location'] = subdata['dec_location'].encode('ascii', 'ignore')
                                             dlist.append(subdata['id'])
                                             dlist.append(subdata['coordinates'])
                                             dlist.append(subdata['dec_location'])
                                             dlist.append(subdata['followers_count'])
                                             dlist.append(subdata['status_count'])
                                             dlist.append(subdata['in_reply_to_user_id'])
                                             dlist.append(subdata['user_mentions'])

                                             wr.writerow(dlist)
```

A. (continued)

```
                            elif data['entities']['user_mentions'] != []: # collects mentions

                                subdata = {k: data[k] for k in ('lang',)}
                                subdata['dec_location'] = data['user']['location']
                                subdata['coordinates'] = data['coordinates'][u'coordinates']
                                subdata['id'] = data['user']['id']
                                subdata['followers_count'] = data['user']['followers_count']
                                subdata['status_count'] = data['user']['statuses_count']
                                subdata['user_mentions'] = data['entities']['user_mentions'][0]['id']
                                subdata['in_reply_to_user_id'] = None

                                tnc += 1
                                ncm += 1
                                dlist = []

                                if subdata['dec_location'] != None: # handles unicoded strings
                                   subdata['dec_location'] = subdata['dec_location'].encode('ascii', 'ignore')
                                dlist.append(subdata['id'])
                                dlist.append(subdata['coordinates'])
                                dlist.append(subdata['dec_location'])
                                dlist.append(subdata['followers_count'])
                                dlist.append(subdata['status_count'])
                                dlist.append(subdata['in_reply_to_user_id'])
                                dlist.append(subdata['user_mentions'])

                                wr.writerow(dlist)
                        else:
                            continue
                    print ncr, '/', tnc, 'replies entered and ', ncm, '/', tnc, 'mentions entered'
except BadZipfile:
    print('  skipping..')
except ValueError:
    print('  skipping..')
```

B. Assigning Locations

```
# assigns coordinates to mention and reply ID's
import pandas

data = pandas.read_csv('output.csv')

# creates coordinate map
useable = dict(zip(reversed(data.userid),reversed(data.coords)))


data['replycoords'] = data['reply'].map(useable)
data['mentcoords'] = data['mention'].map(useable)

print 'assigning complete'

# removes empty row 1
data1 = data.drop('reply',1)
data1 = data1.drop('replycoords',1)
data1 = data1.dropna(subset = ['mentcoords'])

data2 = data.drop('mention',1)
data2 = data2.drop('mentcoords',1)
data2 = data2.dropna(subset = ['replycoords'])

print 'writing to files...'

export_csv = data1.to_csv (
        r'C:/Users/ocean/Documents/thesis/finaloutfastments.csv',
        index=None, header=True)
export_csv = data2.to_csv (
        r'C:/Users/ocean/Documents/thesis/finaloutfastreplys.csv',
        index=None, header=True)
```

C. Removing Duplicates

```
# Removes duplicates (repeated mentions) from data
import pandas

rdata = pandas.read_csv('finaloutfastreplys.csv')
mdata = pandas.read_csv('finaloutfastments.csv')

rdata.drop_duplicates(subset = ['coords', 'replycoords'],
                      keep='last', inplace = True)
mdata.drop_duplicates(subset=['coords','mentcoords'],
                      keep='last', inplace = True)

export_csv = rdata.to_csv (
        r'C:/Users/ocean/Documents/thesis/parsedreplyoutput.csv',
        index=None, header=True)
export_csv = mdata.to_csv (
        r'C:/Users/ocean/Documents/thesis/parsedmentoutput.csv',
        index=None, header=True)
```

D. Plotting

```
# Plots users to mentions on a map of the US
import os
import pandas
os.environ['PROJ_LIB'] = r'C:\Users\ocean\Anaconda2\Library\share'
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap


m = Basemap(projection='mill',
            llcrnrlat = 25,
            llcrnrlon = -130,
            urcrnrlat = 50,
            urcrnrlon = -60,
            resolution = 'l') #creates map

m.drawcoastlines() #adds edges
m.drawcountries(linewidth=2) #adds countrylines
m.drawstates(color='b') #adds statelines

#m.bluemarble() #adds flavor
data = pandas.read_csv('parsedmentionoutput.csv')

usercoords = data.coords.tolist() #reads coords column as list
mentcoords = data.mentcoords.tolist() #reads ment coords column as list
def convert(string): #splits up coordinate data into lat + long
    li = list(string.split(','))
    return li
xs = []
ys = []
# turns every element in usercoords into its long float value
userlong = [float(convert(item[1:])[0]) for item in usercoords]
userlat = [float(convert(item[:-1])[1]) for item in usercoords]
xpt,ypt = m(userlong,userlat)
m.plot(xpt,ypt, 'c*', markersize=2, label = 'user')
xs.append(xpt)
ys.append(ypt)

mentlong = [float(convert(item[1:])[0]) for item in mentcoords]
mentlat = [float(convert(item[:-1])[1]) for item in mentcoords]
xpt,ypt = m(mentlong,mentlat)
m.plot(xpt,ypt, 'g*', markersize=2, label = 'mention')
xs.append(xpt)
ys.append(ypt)

m.plot(xs,ys, color='r', linewidth=.2) # draws edges between connected nodes


plt.legend(loc=4)



plt.title('US Tweets')
plt.show()
```

E. Formatting

```
# Creates new set of data organized by user, attached to all replies/mentions

# part0

import pandas
import csv


rdata = pandas.read_csv('parsedreplyoutput.csv')
mdata = pandas.read_csv('parsedmentoutput.csv')

# extracts lists of user, reply, and reply coordinates
ruser = rdata.userid.tolist()
rcoords = rdata.coords.tolist()
reply = rdata.reply.tolist()
replycoords = rdata.replycoords.tolist()

# extracts lists of user, mention, and mention coordinates
muser = mdata.userid.tolist()
mcoords = mdata.coords.tolist()
mention = mdata.mention.tolist()
mentcoords = mdata.mentcoords.tolist()

 # collects a list of self @'s
rnonat = []
mnonat = []
for i, val in enumerate(rcoords):
    if val == replycoords[i]:
        rnonat.append(i)
print 'self @\'s eliminated', len(rnonat), '/ 2021'
for i, val in enumerate(mcoords):
    if val == mentcoords[i]:
        mnonat.append(i)
print 'self @\'s eliminated', len(mnonat), '/ 33792'
# removes self @'s and saves data as new lists
newrdata = rdata.drop(rdata.index[rnonat])
newmdata = mdata.drop(mdata.index[mnonat])
# extracts new user coord list from "newdata"
prcoords = newrdata.coords.tolist()
pmcoords = newmdata.coords.tolist()
# extracts new mention coord list from "newdata"
preplycoords = newrdata.replycoords.tolist()
pmentcoords = newmdata.mentcoords.tolist()
```

E. (continued)

```
# part 1
# collects lists of bidirectional user coordinates and reply coordinates

birusercoords = [] # step 2a user list
bireplycoords = [] # step 2a mention list

rpair = zip(preplycoords, prcoords)
for i,j in zip(prcoords,preplycoords):
    check = i,j
    if check in rpair:
        birusercoords.append(i)
        bireplycoords.append(j)

print 'there are', len(birusercoords)*2, 'bidirectional replies'
# collects lists of bidirectional user coordinates and mention coordinates
bimusercoords = [] # step 2b user list
bimentcoords = [] # step 2b mention list
# join two lists for cross examination
mpair = zip(pmentcoords, pmcoords)
for i,j in zip(pmcoords,pmentcoords):
    check = i,j
    if check in mpair:
        bimusercoords.append(i)
        bimentcoords.append(j)

print 'there are', len(bimusercoords)*2, 'bidirectional mentions'

# part 2
# reshapes data as coordinate-adjacency list

bicoord1 = birusercoords + bimusercoords # combines replies and mentions
bicoord2 = bireplycoords + bimentcoords
pcoords1 = [] # final 'user' list
pcoords2 = [] # final 'mention' list

# groups users with all corresponding mentions, separated by '/'
for i, val in enumerate(bicoord1):
    if val not in pcoords1:
        pcoords1.append(val)
        pcoords2.append(bicoord2[i])
    else: # joins mentions to common user
        pcoords2[pcoords1.index(val)] = '/'.join(
                (pcoords2[pcoords1.index(val)], bicoord2[i]))

rows = zip(pcoords1,pcoords2)

with open('nodetoallmentions.csv', 'wb') as output:
    writer = csv.writer(output)
    writer.writerow(['user','mention'])
    for row in rows:
        writer.writerow(row)
```

F. Average Degree

```
# calculates the average degree
import pandas

data = pandas.read_csv('nodetoallmentionsd.csv')

user = data.user.tolist()
mentions = data.mention.tolist()

deg = []

for i in range(len(user)):
    if "/" not in mentions[i]:
        deg.append(1)
    else:
        if mentions[i].count('/') < 200: # limits extreme users
            deg.append(mentions[i].count("/") +1)

# print(deg) # returns list of all degrees of network


avg_deg = sum(deg) / float(len(deg))

print 'Average degree =', avg_deg
```

G. Clustering Coefficient

```
# Calculates the global clustering coefficient
import pandas

data = pandas.read_csv('nodetoallmentions.csv')

user = data.user.tolist()
mentions = data.mention.tolist()
clust = []
for i in range(len(user)):
    if "/" in mentions[i]:
        pairs = 0
        # separates each item in mention column for iteration
        for val in mentions[i].split('/'):
            if val in user: # checks if neighbor has an edge
                # checks if neighbor is connected to other neighbors
                for v in mentions[user.index(val)].split('/'):
                    if v in mentions[i].split('/'):
                        pairs += 1
        # clustering equation
        cci = 2*pairs / ((mentions[i].count("/") + 1) * mentions[i].count("/"))
        clust.append(cci)
    else:
        clust.append(0)

avg = sum(clust) / (float(len(user))+float(len(mentions))+mentions.count("/"))
print 'clustering coefficient =', avg
```

H. SEC Script

```
import pandas
from geopy.distance import geodesic


data = pandas.read_csv('nodetoallmentions.csv')

user = data.user.tolist()
mentions = data.mention.tolist()

neighbors = []
userlist = []
mentlist = []
dcutt = 25.7495 # 16 mile cutoff distance
sec = [] # list of sec values for each node
c = 0 # separate counter for new gn list indices
for i in range(len(user)):
    if "/" in mentions[i]:
        neighbors.append(mentions[i].count("/") +1) # creates list of degrees
        mtemp = mentions[i].replace('[','')
        mtemp = mtemp.replace(']','')
        utemp = user[i].replace('[','')
        utemp = utemp.replace(']','')
        mentlist.append(
                [[float(y) for y in x.split(',')] for x in mtemp.split('/')])
        userlist.append(
                [float(y) for y in utemp.split(',')])
        gn = 0 # global neighbors counter
        local = 0 # local node counter for sec calculation
        for j in mentlist[c]:
# list of stored distances to prevent double count of close global neighbors
            dist = []
            # checks that neighbor is outside commute distance 16m
            if geodesic(userlist[c][::-1], j[::-1]).km > dcutt:
                gn += 1
                for k in mentlist[c]:
                    if j != k and geodesic(
                            userlist[c][::-1],k[::-1]).km > dcutt:
                        # compares 2 global neighbors
                        if geodesic(j[::-1], k[::-1]).km < dcutt:
                            if geodesic(j[::-1], k[::-1]).km not in dist:
                                local += 1
                                dist.append(geodesic(j[::-1], k[::-1]))
        if gn !=0: # only calculates SEC of nodes with global neighbors
            sec.append((1+local/gn)/gn)
        else:
            neighbors.append(mentions[i].count('/')+1)
        c += 1

    else:
        neighbors.append(1)

avg_sec = sum(sec)/float(len(sec))

print 'avg sec = ', avg_sec
```