CSE 472: Social Media Mining – Project 1

Twitter Data Crawling using Twitter API and Python

Step 1: Crawling Twitter data.

I used the API method to crawl data from Twitter. I created a config file which contains my api key, api key secret, access token and access token secret, which are fetched by the twitter program. Then, I used Tweepy for the API authentication.

I decided to crawl data regarding the different attitudes towards to Covid-19, i.e., Pro-vaccine and Anti-vaccine. For this, I fetched 200 tweets for each attitude in a cursor using the search_tweets method by setting specific keywords and filtering the language as English and excluding retweets to eliminate repetitions.

```
[12]  keywords = '#WearAMask -filter:retweets'
      limit = 200
      ✓ 0.4s

      tweets = tweepy.Cursor(api.search_tweets, q=keywords, lang='en',
          count=100, tweet_mode='extended').items(limit)
[13]  ✓ 0.6s
```

Next, I stored the username, full tweet text and entities of the tweet in a dataframe, which was used to store the data in a .JSON file.

```
[14]  #creating dataframe
      columns = ['User','Tweet','Entities']
      data = []
      for tweet in tweets:
          data.append([tweet.user.screen_name,tweet.full_text,tweet.entities])
      df = pd.DataFrame(data,columns=columns)
      ✓ 0.8s
```

The entities attribute of the tweet has been stored because they explicitly store all the hashtags mentioned in that particular tweet in a dictionary, like so:

```
                                                Entities
0    {'hashtags': [{'text': 'CovidisAirborne', 'ind...
1    {'hashtags': [{'text': 'RBandME', 'indices': [...
2    {'hashtags': [{'text': 'Covidisntover', 'indic...
3    {'hashtags': [{'text': 'WearAMask', 'indices':...
4    {'hashtags': [{'text': 'ThankYouHCW', 'indices...
..                                                 ...
195  {'hashtags': [{'text': 'WearAMask', 'indices':...
196  {'hashtags': [{'text': 'WearAMask', 'indices':...
197  {'hashtags': [{'text': 'covid', 'indices': [0,...
198  {'hashtags': [{'text': 'doctorcelestemd', 'ind...
199  {'hashtags': [{'text': 'GetBoosted', 'indices'...

[200 rows x 3 columns]
```

This helped me prepare a list of all the hashtags present in all the tweets combined. It must be noted that different users may type the same hashtag in different ways due to different case. For instance, #WearAMask and #wearamask are the same hashtag for twitter's API but when stored in a list, they would be counted as different elements. Hence, I converted all the hashtags to their lower case. Then, I created a different list which only contained the unique hashtags (only 1 instance of each hashtag).

```
hashtagslist = []
for dict in df['Entities']:
    for ht in dict['hashtags']:
        hashtagslist.append(ht['text'])
```
[17] ✓ 0.3s

```
for i in range(len(hashtagslist)):
    hashtagslist[i] = hashtagslist[i].lower()
```
[78] ✓ 0.8s

```
unique_hashtags = list(set(hashtagslist))
```
[79] ✓ 0.5s

Then, I create a dictionary of tweets which will be used later for the creating the co-occurrence matrix.

```
tweets_dictionary = {}
iterative = 0
for tweet in df['Tweet']:
    iterative += 1
    for uht in unique_hashtags:
        if '#'+(uht) in tweet:
            if str(iterative) in tweets_dictionary.keys():
                tweets_dictionary[str(iterative)].append(uht)
            else:
                tweets_dictionary[str(iterative)] = [uht]
```
[41] ✓ 0.4s

I created the matrix, which turned out to be a sparse matrix, (on second thoughts, this is how it should have been because not every user will be using the same set of hashtags in their tweets)

```
df1 = pd.DataFrame(columns = unique_hashtags, index = unique_hashtags)
df1[:] = int(0)
for value in tweets_dictionary.values():
    for uht1 in unique_hashtags:
        for uht2 in unique_hashtags:
            if uht1 in value and uht2 in value:
                df1[uht1][uht2] += 1
                df1[uht2][uht1] += 1
```
✓ 0.5s

Then, I created 2 lists, one for the edges and one for the nodes. The edges list stores data in the form of [node1, node2, co-occurrence weight]. The weights were divided by 200 (number of tweets) to represent them as a fraction of the primary keyword used for searching. The nodes list stores data in the form of [node, number of occurrences].
I also remove the nodes with 0 occurrences, and those edges which are between the same nodes.

```
edge_list = []
for index, row in df1.iterrows():
    i=0
    for col in row:
        weight = float(col)/400
        edge_list.append((index,df1.columns[i],weight))
        i+=1

updated_edge_list = [x for x in edge_list if not x[2] == 0.0]

node_list = []
for i in unique_hashtags:
    for e in updated_edge_list:
        if i == e[0] and i == e[1]:
            node_list.append((i, e[2]*5))
for i in node_list:
    if i[1] == 0.0:
        node_list.remove(i)
# for i in node_list:
#     if i[1] < 5.0:
#         node_list.remove(i)

for i in updated_edge_list:
    if i[0] == i[1]:
        updated_edge_list.remove(i)
```

[45] ✓ 0.1s

So now, we have our cleaned co-occurrence matrix. All that's left is to plot a graph.
First, I create a plain graph using networkx and added the nodes and edges to it.

```
plt.subplots(figsize=(10,10))

G = nx.Graph()
#G = nx.barabasi_albert_graph(1000,2,20532)
for i in sorted(node_list):
    G.add_node(i[0],size=i[1])
G.add_weighted_edges_from(updated_edge_list)
```

✓ 0.2s

My aim was to present the nodes with more occurrences with a larger size as compared to the ones with less occurrences, and the edge between nodes which co-occur more frequently to be thicker than other edges. For this, I had to maintain 2 more lists with the sizes of nodes and width of edges.

```
node_order = nx.nodes(G)

updated_node_order = []
for i in node_order:
    for x in node_list:
        if x[0] == i:
            updated_node_order.append(x)

test = nx.get_edge_attributes(G,'weight')
updated_edges_2 = []
for i in nx.edges(G):
    for x in test:
        if i[0] == x[0] and i[1] == x[1]:
            updated_edges_2.append(test[x])
```

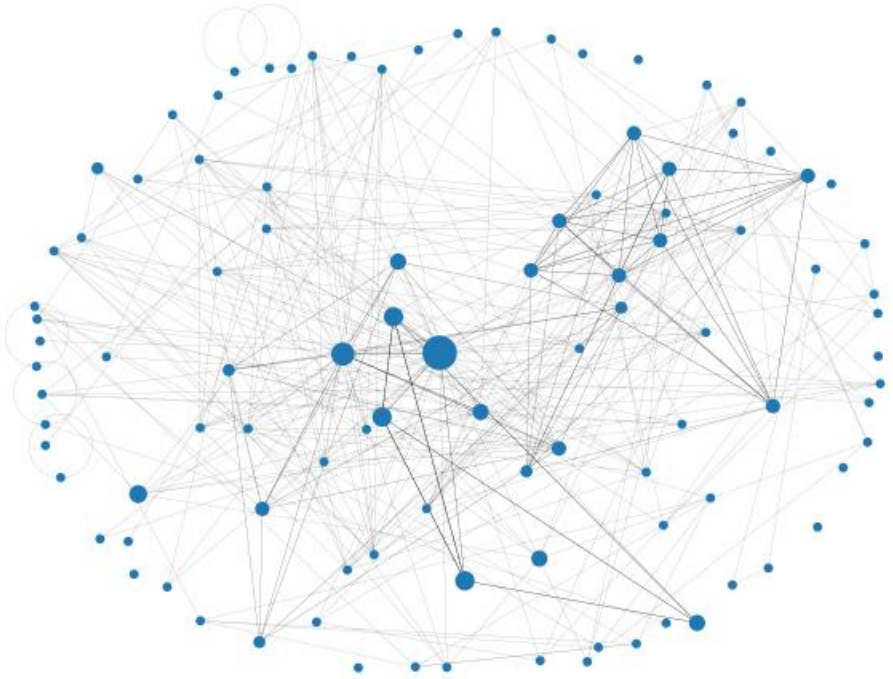
✓ 0.1s

Then, I upscale the sizes of the nodes and edges by multiplying them with a scalar value and determine the positions of the nodes by using spring_layout of network and plot the graph.

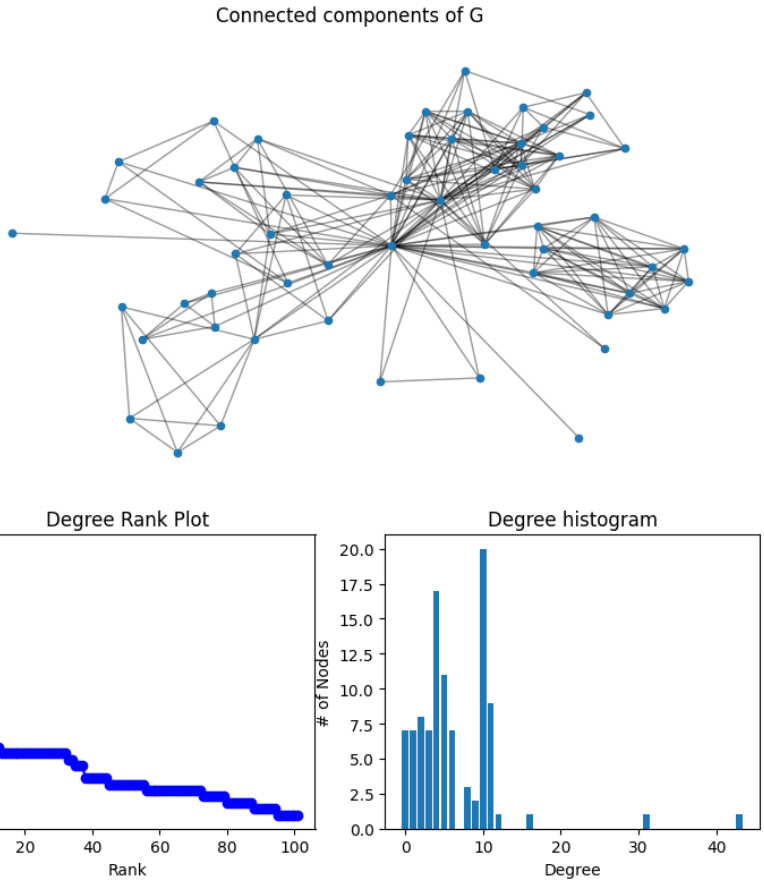
```
node_scalar = 400
edge_scalar = 10

sizes = [x[1]*node_scalar for x in updated_node_order]
widths = [x*edge_scalar for x in updated_edges_2]

pos = nx.spring_layout(G, k=0.5, iterations = 100, seed=1)
#pos = nx.get_node_attributes(G, "pos")

nx.draw(G, pos, with_labels=False, font_size = 6,
    node_size=sizes, width = widths)
```

✓ 0.2s

Result:
For the keyword '#WearAMask' (pro-vaccine attitude):
Network Graph:



Connected components and Degree distribution histogram:



Diameter of the connected graph:

```
    dia = nx.diameter(Gcc)
    print("The diameter of the connected graph is: ",dia)
[81] ✓ 0.3s
...  The diameter of the connected graph is:  3
```

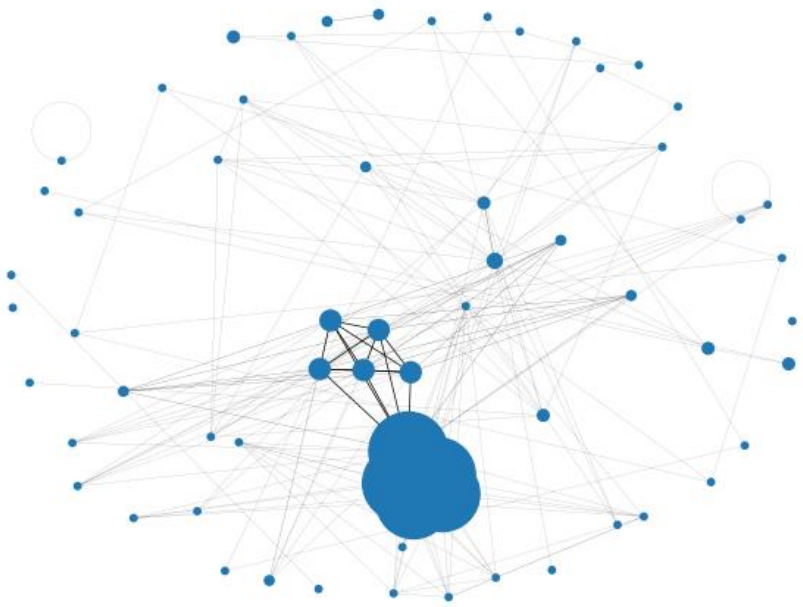
Degree centrality:
We know that the node with the highest number of edges will have the largest degree centrality. In my case, it should be the node “WearAMask” since it is the primary keyword used for extracting the tweets (which means that it will be present in all of the tweets). To verify the same:

```
    dc = nx.degree_centrality(G)
    print(list(dc.keys())[list(dc.values()).index(max(dc.values()))],max(dc.values()))
[74] ✓ 0.1s
...  wearamask 0.42574257425742573
```

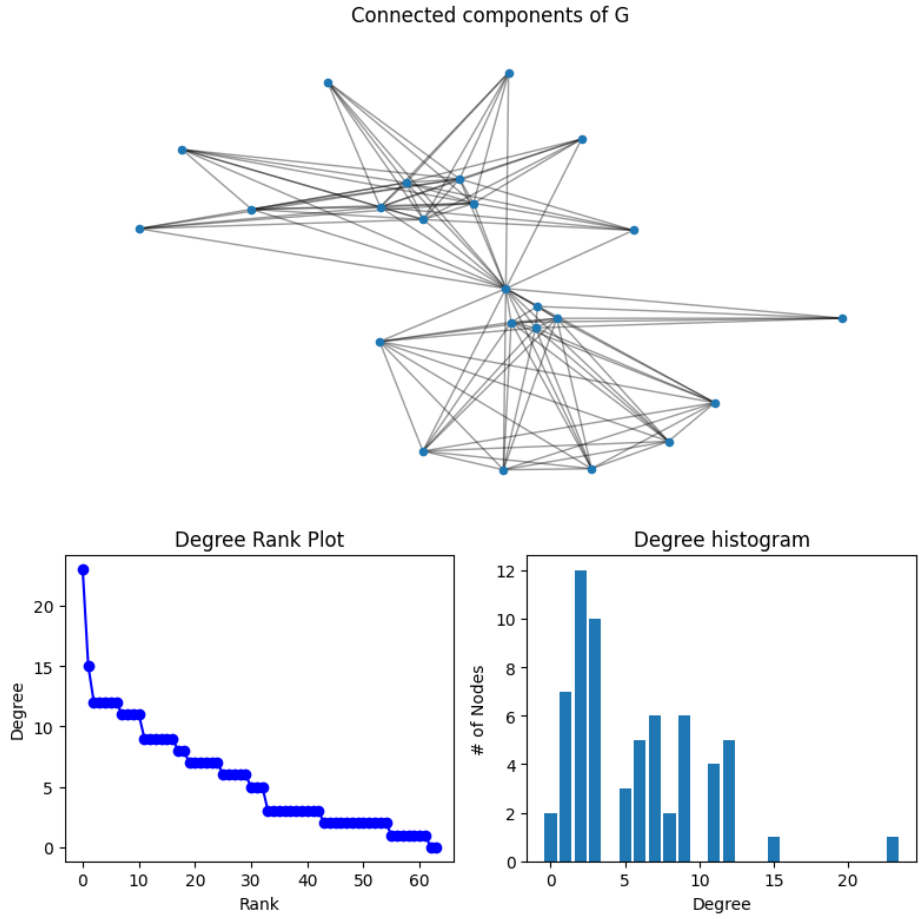
Betweenness Centrality:
Like degree centrality, the node “WearAMask” must have the hight betweenness centrality. To verify the same:

```
    bc = nx.betweenness_centrality(G)
    print(list(bc.keys())[list(bc.values()).index(max(bc.values()))],max(bc.values()))
[77] ✓ 0.8s
...  wearamask 0.187013201320132
```

For the keyword ‘#NoVaccineMandates’ (anti-vaccine attitude):



Connected components and Degree Distribution Histogram:



Diameter of the connected graph:

```
    dia = nx.diameter(Gcc)
    print("The diameter of the connected graph is: ",dia)
[135] ✓ 0.5s
...  The diameter of the connected graph is:  2
```

Degree centrality:

In this case, the degree centrality of the node ‘NoVaccineMandates’ should have the highest degree centrality. However, this is only valid if any hashtag appears only once in a particular tweet. The hashtag I used to fetch the tweets returns 200 tweets, of which more than 175 tweets have been tweeted by bots, and those tweets have one other keyword appearing more than once in each tweet. Hence, the degree centrality of that hashtag is higher:

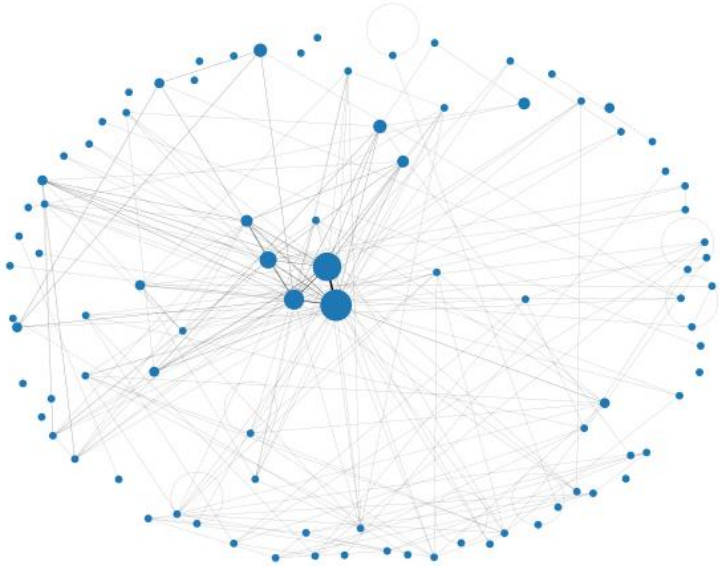
```
    dc = nx.degree_centrality(G)
    print(list(dc.keys())[list(dc.values()).index(max(dc.values()))],max(dc.values()))
[227] ✓ 0.4s
...  cdnpoli 0.36507936507936506
```

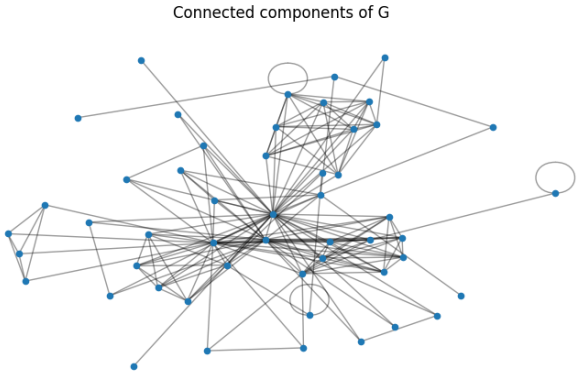
Similarly, the betweenness centrality:

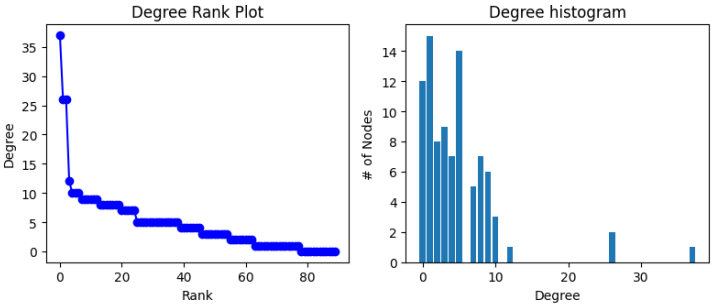
```
[228]
    bc = nx.betweenness_centrality(G)
    print(list(bc.keys())[list(bc.values()).index(max(bc.values()))],max(bc.values()))
✓ 0.4s

cdnpoli 0.07083119986345791
```

Takeaways:

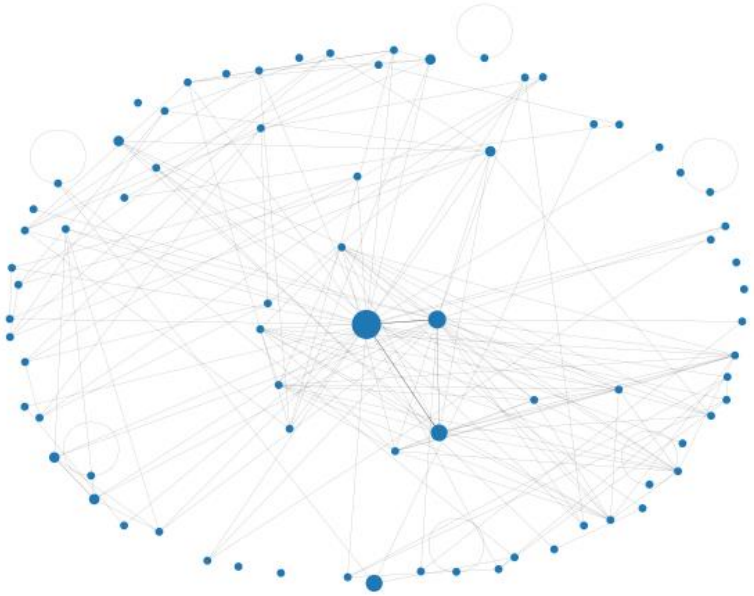
1. When we talk about positive keywords like 'WearAMask,' the tweets containing these hashtags can be for a variety of uses. For instance, someone using these hashtags might just be spreading awareness, while someone might be urging people to take precautions, while someone might be advertising an event and mentioning that it is mandatory to #WearAMask. As a result, the graph is dense. On the other hand, the tweets containing keywords like 'NoVaccinesMandate' or 'NoVaccineForMe' are generally meant as a protest towards the vaccinations, and do not contain a variety of other hashtags, which results in a sparse graph.
2. When we talk about the positive approach, since it is a dense graph, we can see that the highest degree is much more than what it is for the negative approach. Similarly, there are many more nodes with higher degree like 10.
3. The network graph for each of the approach proves the above statements. The positive approach graph shows a larger number of nodes with more nodes concentrated in the middle of the region, while the negative graph only has a handful of nodes in the central region. The size appears to be larger on the negative graph, however, it must be noted that these sizes are relative and not absolute.
4. To prove the claim that positive networks tend to be denser, here are some more plots of positive networks:
    a. Keyword - #MaskUp

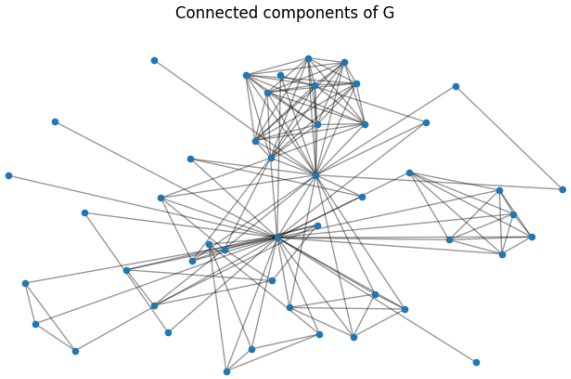


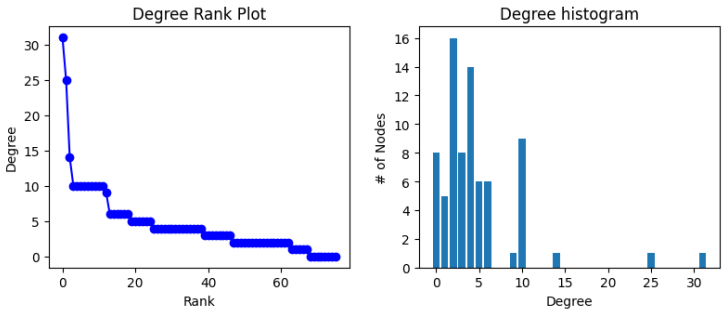

Connected components of G


Degree Rank Plot
Degree
Rank
0
5
10
15
20
25
30
35
0
20
40
60
80
Degree histogram
# of Nodes
Degree
0
2
4
6
8
10
12
14
0
10
20
30

b. Keyword - #GetVaccinated







Both these examples show that there are more nodes in the network graph and connected nodes graph than the negative attitude graph.