CSE 472 - Social Media Mining

Project 1

Shreesh Nayak, Skanda Suresh

{1217214310, 1217132644}

30th September, 2019

1 Data Collection

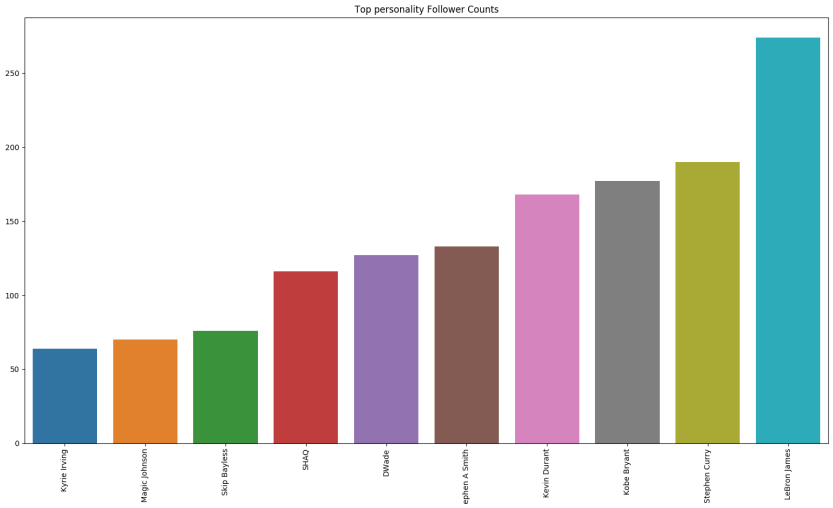
We chose to scrape the social media website Twitter using Python's library **twython**. We decided to make a social network consisting of a few NBA players, a few media personalities and the people they are followed by. First, we determined which NBA players and media personalities to get the data from. We chose the following people as they are highly active on Twitter: Lebron James, Stephen A. Smith, Kevin Durant, Stephen Curry, Shaquille O'Neal, Dwayne Wade, Magic Johnson, Kyrie Irwing, Skip Bayles and Kobe Bryant. Once we selected the 10 people, we scraped their profiles and got the data for 50 followers for each person and stored this in a .csv file named with the person's Twitter handle. We decided to get each followers' data, because it would provide us with a network of roughly 500 people or lesser based on the amount of intersection between players and also based on the number of protected accounts. Then for each of the 510 people in the network, we scraped each profile and got a list of their friends' (people they're following)

IDs. This would allow us to determine whether each person followed more than one player and whether they followed anyone else in the network. We created a social_network dictionary in Python, where the keys represent the person's Twitter ID and the values represent the list of Twitter IDs of the people they are following. As we looped through the 510 people, we would add a person to the network dictionary only if they weren't already present in it. If they weren't already in the network dictionary, we would make an API call to determine the list of their friends' IDs, after which we would add the person to the network. This was done to avoid making any unnecessary API calls as we are limited to 15 calls every 15 minutes. We also needed to keep in mind that we would not be able to get data for a person if their account was protected, which would result in a TwythonAuthenticationError. If we'd catch this Exception, then the person would be added to our network with a value of '-1' and would be cleaned out later on. Furthermore, every time a person was added to our network, we would dump our social_network dictionary into a .pkl file using Python's library **pickle**, to save all the data in the memory. As we had two developer accounts, we decided to each work on 255 people, and then combine our separate dictionaries. The final combined dictionary contained 411 entries. The code for the following section can be found in **python-twitter-scraper.py** and **merge_network.py**

2 Data Visualisation

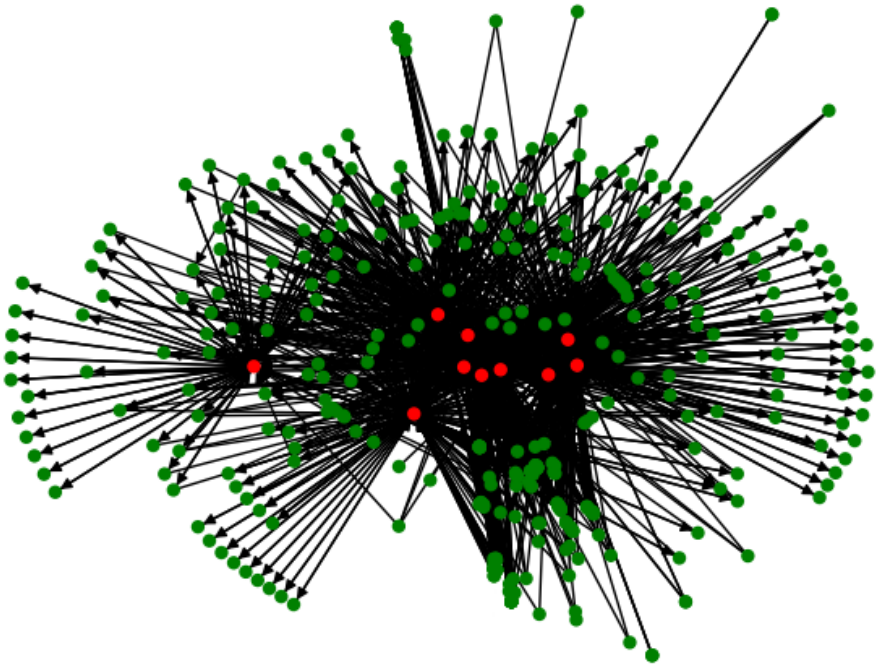
We used NetworkX to visualize our graph. To do this, we first had to parse through our now combined social_network dictionary and create a Numpy adjacency matrix. To do this, we created two separate dictionaries called id_2_index and index_2_id to map the ID of each person to an index in the

matrix and vice versa as dictionaries only provide a one way mapping. After the adjacency matrix was created, we stored it in the social_network.npy file. The follower count for each node is computed by taking sum along (axis = 0). The top 10 most followed nodes are displayed below:

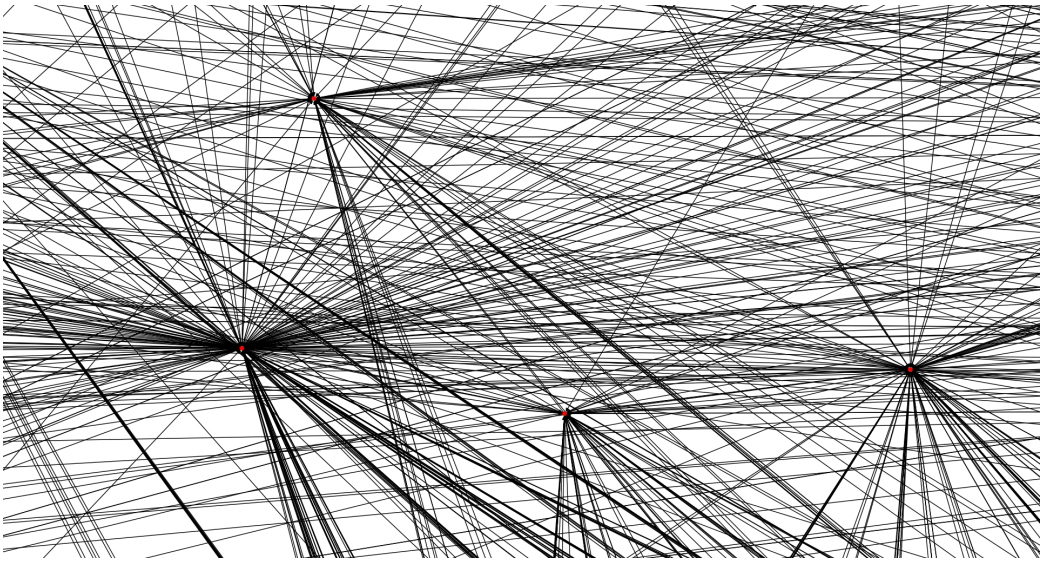


We used Numpy as NetworkX has a function called 'from_numpy_array' to create a graph from a numpy matrix. The adjacency matrix was created in such a manner that the rows would represent the people you are following. So if in a particular row, if there's a '1' for a particular column, then it means that the person who is represented by the row is following the person represented by the column.

Below is a depiction of our social network, *the red nodes represent the 10 most popular celebrities who were scraped.*



Zooming into some key nodes of the network



The code for the following section can be found in **make_matrix.py**

3 Network Measures Calculation

Now that the graph is constructed and visualized using **networkx**, we make use of the same package to calculate 3 particular measures, namely:

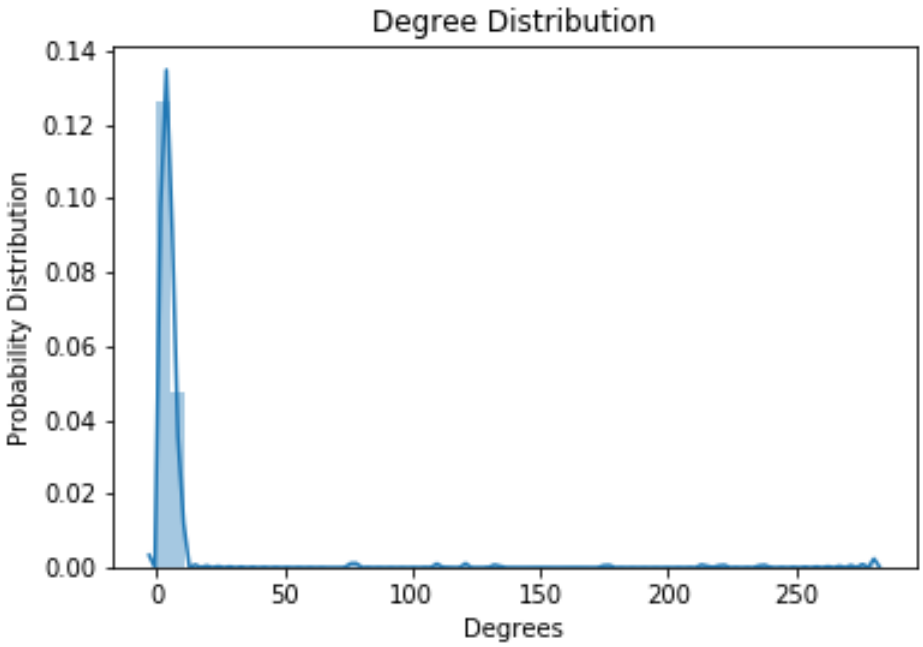
- Degree Distribution
- Eigenvector Centrality
- Betweeness Centrality

These measures help identify the primary nodes in the social network taking into consideration certain properties like connectivity, information flow, etc.

3.1 Degree Distribution

The Degree Distribution plots a Probability Distribution Function of the frequency of degrees of each node. This makes use of the concept of Degree Centrality. Degree Centrality essentially ranks the importance of a node based on the number of connections it has. In our case we make use of a directed graph. Thus for a given node, the number of incoming edges (prestige), and outgoing edges (gregariousness) form the in degree and out degree respectively. The degree of an edge is calculated as the sum of its in and out degree. For vertex V_i;

$$d_i = d_i(in) + d_i(out) \tag{1}$$



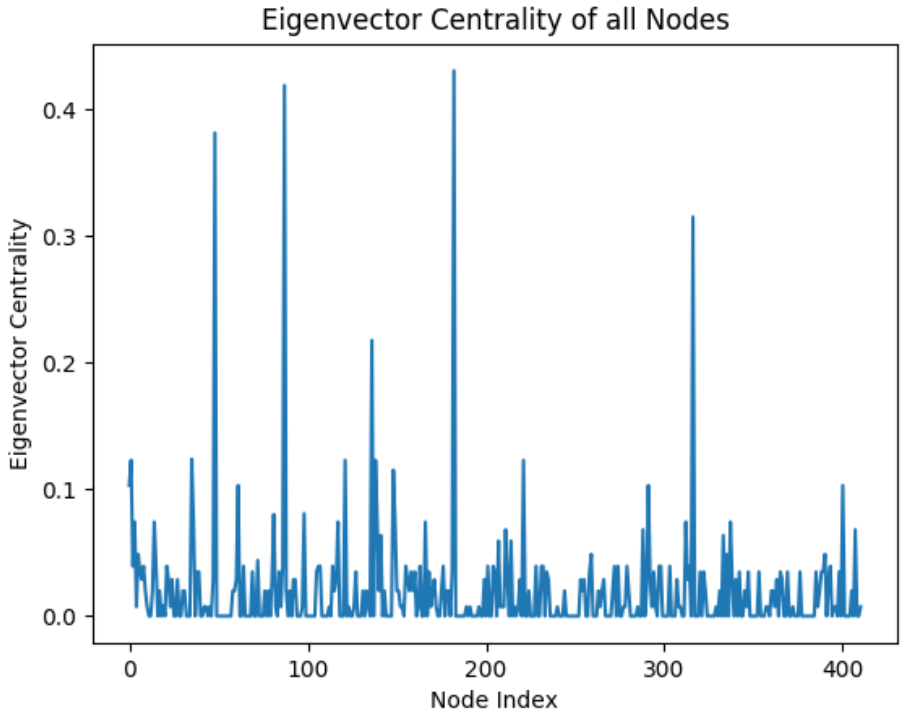
We notice that the node corresponding to **ID 87** (LeBron James) has the highest degree = **280** (**274** followers and **6** friends).

3.2 Eigenvector Centrality

Eigenvector centrality is an extension of degree centrality. It takes into account the fact that nodes connected to relevant nodes would rank higher in the relevance list. It is based on the principle:

A node is important if it is linked to by other important nodes.

We compute the eigenvector centrality for each node and plot the distribution as shown below.



Higher the Eigenvector centrality, more relevant the node. We notice from the above plot, ID's **182** and **87** have the highest values, and they correspond to players Steph Curry and LeBron James, who happen to be the 2 most popular players on twitter.

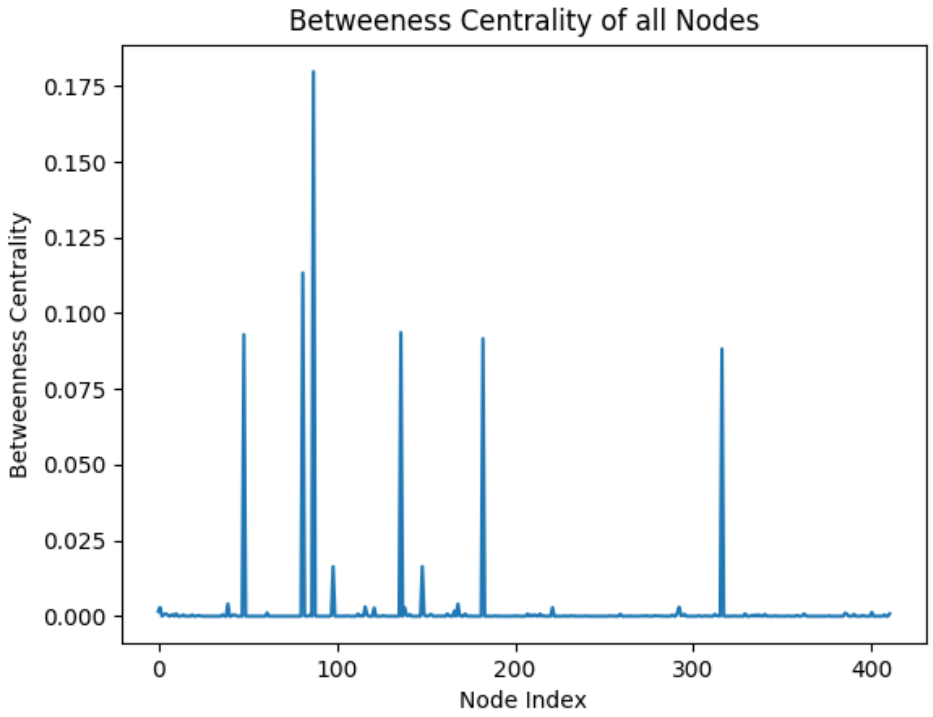
3.3 Betweeness Centrality

Betweeness centrality represents the relevance of a node by measuring its potential for information flow control in a network. Intuitively it show how often the node occurs in a path of communication between any 2 nodes. It is the fraction of the number of paths in which the node appears amongst all shortest paths between 2 nodes in the network. For a given vertex v, and

any two vertices s and t;

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2}$$

The betweeness centrality plot across all nodes is as shown below.



We notice ID **87** (LeBron James), has the highest Betweeness Centrality amongst all. The code for the following section can be found in **network_measures.py**

4 Conclusion

We have scraped Twitter data for 10 popular NBA celebrities and media personalities, and their followers. The social network consists of **411** nodes. Intuitively we can see that the top 10 celebrities would form the most influential nodes. Amongst these celebrities, notably two names LeBron James and Stephen Curry (corresponding to Node Indices (ID's) **87** and **182**) have

the highest number of followers in general (approx. 30 and 18 million respectively). Our experiments indicate that these two nodes are the most relevant across all network measures.

References

[1] Python Library, Twython: https://twython.readthedocs.io/en/latest/

[2] Python Library, NetworkX: https://networkx.github.io/documentation/stable/

[3] Python Library, Pickle: https://docs.python.org/3/library/pickle.html

[4] Python Library, Numpy: https://numpy.org/devdocs/

[5] Python Library, Matplotlib: https://matplotlib.org/3.1.1/index.html

[6] Python Library, Seaborn: https://seaborn.pydata.org/introduction.html

[7] Python Library, CSV: https://docs.python.org/3/library/csv.html

[8] Python Library, JSON: https://docs.python.org/3/library/json.html

[9] Python Library, Time: https://docs.python.org/3/library/time.html

[10] https://stackoverflow.com/questions/28560876/is-there-a-way-to-run-pagerank-algorithm-on-networkxs-multigraph

[11] https://stackoverflow.com/questions/28996913/drawing-a-directed-graph-using-a-link-matrix-with-networkx

[12] https://stackoverflow.com/questions/20133479/how-to-draw-directed-graphs-using-networkx-in-python

[13] https://stackoverflow.com/questions/44692644/plotting-networkx-graph-in-python

[14] https://www.silkstream.net/blog/2014/06/playing-with-followers-with-twython-csv.html

[15] https://stackoverflow.com/questions/27450325/accommodating-twitters-rate-limit-in-pythons-twython

[16] https://codereview.stackexchange.com/questions/115954/writing-lists-to-csv-file

[17] https://www.programiz.com/python-programming/working-csv-files

[18] https://stackoverflow.com/questions/6916542/writing-list-of-strings-to-excel-csv-file-in-python

[19] https://www.fullcontact.com/developer/docs/rate-limits/

[20] https://codereview.stackexchange.com/questions/101905/get-all-followers-and-friends-of-a-twitter-user

[21] http://2017.compciv.org/guide/topics/python-nonstandard-libraries/twython-guide/twitter-twython-api-basics.html

[22] https://stackoverflow.com/questions/29090711/grabbing-a-users-followers-with-twython

[23] https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-ids

[24] https://stackoverflow.com/questions/53958700/plotting-degree-distribution-of-networkx-digraph-using-networkx-degree-histogram

[25] https://www.datacamp.com/community/tutorials/social-network-analysis-python

[26] https://programminghistorian.org/en/lessons/exploring-and-analyzing-network-data-with-python

[27] https://www.analyticsvidhya.com/blog/2018/04/introduction-to-graph-theory-network-analysis-python-codes/

[28] https://blog.dominodatalab.com/social-network-analysis-with-networkx/