Web Science(H)

Network Based Social Media Analytics

Scott Johnston

2323166J

Sourcecode – https://github.com/scottaj99/web-science

Data –

https://github.com/scottaj99/web-science/blob/master/web-sci/original_text.json (For parts 1-4)

https://github.com/scottaj99/web-science/blob/master/web-sci/secondary_text.json (For parts 4-5)

1.Introduction

a. I have developed software for this task in the form of several python scripts. The first of my scripts is titled twitterstream.py which is a crawler I used to gather twitter data using the tweepy twitter API library, and store it in a mongoDB database, using the pyMongo module. This script used a list of keywords to filter a stream of tweets for a set amount of time. The tweets are returned as a json object which I then stripped for the important information before uploading to the database. I used the following youtube video in order to learn about how to use the streaming API, however built heavily on this using the tweepy documentation and twitters own documentation for assistance: https://www.youtube.com/watch?v=wlnx-7cm4Gg&t=1075s.

   My second script clustering.py was used for the grouping of my tweets based on content analysis. This script uses Kmeans to group terms in tweets, usernames and hashtags, and also a cluster of tweets with usernames as dict items which extracts all usernames and hashtags from the tweets and outputs the most popular of these to see if theres any emerging patterns. I used the following stack overflow question and answer as reference to creating this: https://stackoverflow.com/questions/27889873/clustering-text-documents-using-scikit-learn-kmeans-in-python?fbclid=IwAR2pMoqfEDVSXGlZXPDK_0CmY1EmANCKUjPwfXxAl2CDBlo83db3EY6Qw_A I also had to use the regular expression module python docs in order to create my username/tweet dict cluster.

   My third script mentions.py is used for the capturing and organising of user information in order to generate a user interaction graph, showing the connections between users and mentioned users. I used pandas to normalise this data and matplotlib and network to generate the graphs.

   My forth script hashtags.py was used to generate hashtag information in the form of another interaction graph. This worked by developing nodes for each hashtag that appeared in any tweet, and if two hashtags were used in the same tweet, an edge would be formed. Again I used pandas to normalise this data and matplotlib and network to generate the graphs.

   These scripts were based on this article: https://medium.com/future-vision/visualizing-twitter-interactions-with-networkx-a391da239af5 and the referenced github on this website.

   I then created a script called other_interactions.py which is again similar to mentions.py and hashtags.py but this was to capture information on replies, retweets and quote tweet networks. I had to do another stream to obtain this information which was a half hour long stream collecting all rows of the data (rather than just the refined rows I selected in twitterstream.py).

   I generated 2 json files with sample data, each with 100 different items from the database. The original data is stored in original_text.json and my secondary stream to complete the tasks stated above, is stored in secondary_text.json. I generated these two files using a simple script with a few lines of code called write_json.py.

b. I collected the data used in sections 2 and part of 3 over the duration of an hour, however due to my method of data crawl, I had to go back and stream once again, adding in other fields such as retweets, quote tweets and replies. This meant some of my data from parts 4 and 5 of this report are using a different dataset, however the same words were tracked in the streaming.

<u>2. Data Crawl</u>

a. I used the Tweepy Twitter streaming API in order to collect my data. The code worked by firstly configuring a connection to the database. When the stream begins it initialises a variable 'duplicates' to 0, and sets a time limit of 3600 seconds (1 hour), and begins the timer from 0. When a tweet which fits the stream filter is received, the on_data function will be called. This function checks if the time limit has been reached and if not, the tweets data will be received in json format. The function will then try and strip the json object of it's valuable fields, of which I selected the following: tweet ID, username, followers, date, language, hashtags, mentions, time they were created, tweet text, and user location. If possible, this object will be added to the mongo database. If not possible, the tweet is a duplicate so the duplicate field will be incremented by one. If the timer has ended the function will return false and the script will stop running. To run this you require a file twitter_credentials.py which has access_token, access_token_secret, consumer_key and consumer_secret which have to be read into the script to run the app, however I chose not to include this in my repository as these are private keys and could be a breach of security.

b. To enhance the crawling I added a track to the stream which is a list of terms that the stream should listen for. I chose to use all 20 premier league teams as football has a very large twitter community, hence why I was able to collect just over 46,000 tweet objects in my one hour of streaming. I configured the stream as follows:

```
stream.filter(track=keywords)
```

with keywords being a list of all 20 premier league football teams.

I considered adding further filters such as language = eng however chose not too since Liverpool were playing Champions League the evening which I ran my stream so felt there may be a lot of contribution from all around Europe, and even worldwide.

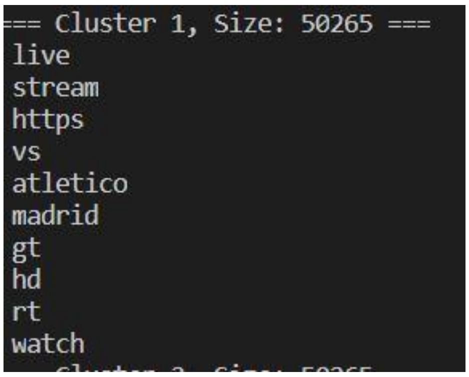
3. Grouping of Tweets

a. To group my tweets I used the sklearn module to cluster my data using KMeans. I created 4 sets of clusters: One set was clusters based on terms used in tweets, another set was hashtags used in tweets, a third set was usernames who tweeted, and a forth set clustered tweets and extracted the usernames and hashtags. For each set, I created 6 clusters. I kept the rest of the parameters in the call to K means I kept the default values of random for initialisation, 10 for number of times the algorithm would run with different centroid seeds and 300 as the max number of iterations of the algorithm for a single run. Sine I used Kmeans, all clusters were the same size: term clusters were 50265 terms long, hashtags were 1138, usernames 32888 and tweets 45984.

b. To extract usernames and hashtags from my forth set of clusters mentioned, I used the regular expression module. I had to manually add the @ sign to the tweeter in order to be able to track this username, before extracting all usernames and hashtags by searching for the terms that appeared after the @ and # symbols respectively. I then displayed these for each cluster. As the screenshot shows, there wasn't a particular stand our user or hashtag as the counter generally collected very few from the clusters (3 or less). This contasts with counter of the most_common(5) I collected earlier on as this showed the hashtag 'LIVATL' was used 2133 times, and the user @BroadcastNnn tweeted 58 times.

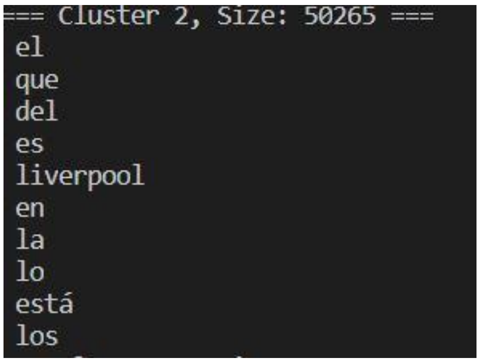
Below are the results of my four clusters:

The clustering of terms seemed to work very well as we see a cluster which clearly used terms promoting a livestream ("live","stream","https","hd") which is common around the time of games being played:



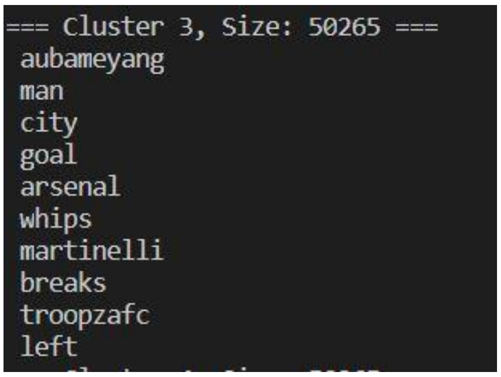
```
=== Cluster 1, Size: 50265 ===
live
stream
https
vs
atletico
madrid
gt
hd
rt
watch
```

a cluster which was clearly picking up on Spanish words (as Liverpool were playing Athletico Madrid at the time of my stream):



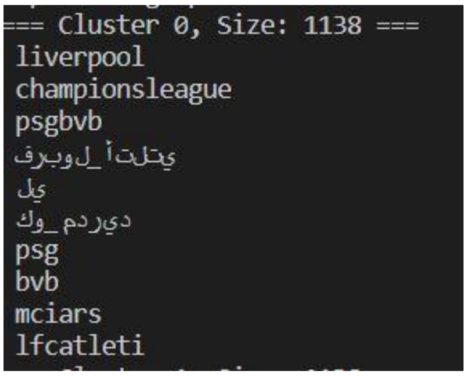
```
=== Cluster 2, Size: 50265 ===
el
que
del
es
liverpool
en
la
lo
está
los
```

And a cluster with terms relating to Arsenal vs Man City which was a game that was meant to be played that day (“Aubameyang”, “Martinelli”, “arsenal”, “city”):



```
=== Cluster 3, Size: 50265 ===
 aubameyang
 man
 city
 goal
 arsenal
 whips
 martinelli
 breaks
 troopzafc
 left
```

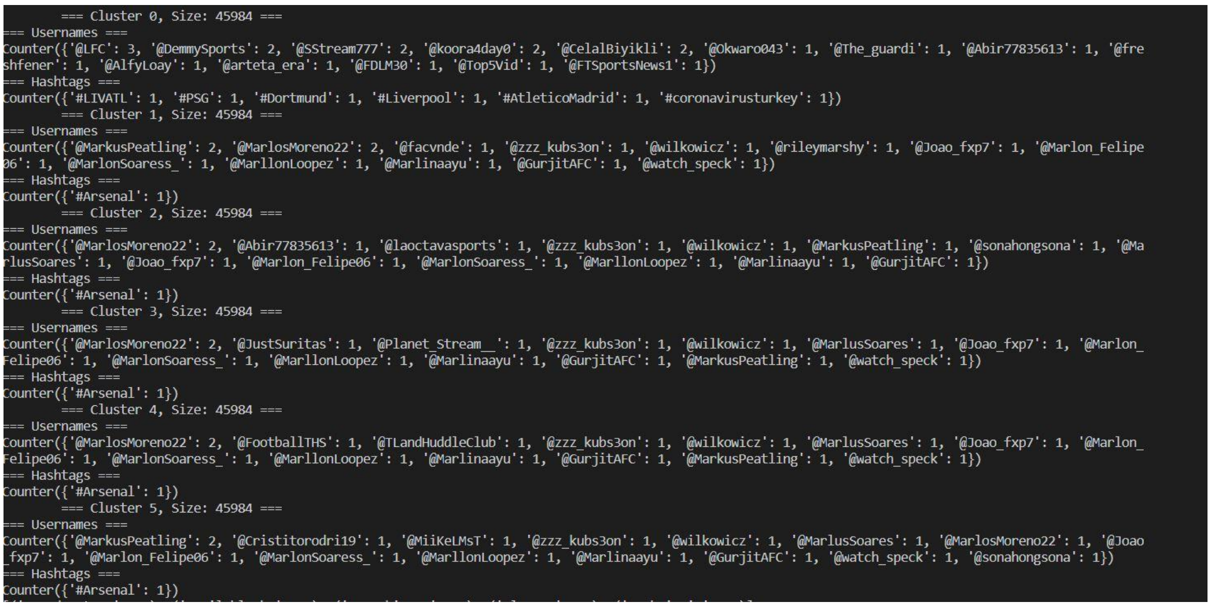
The clustering of hashtags wasn’t quite as effective however one cluster did seem to pick up on a lot of terms relating to champions league and the two fixtures that were being played that night (“liverpool”, “championsleague”, “psgbvb”):



```
=== Cluster 0, Size: 1138 ===
 liverpool
 championsleague
 psgbvb
 يتلتيأ_ليفربو
 لي
 كو_مدريد
 psg
 bvb
 mciars
 lfcatleti
```

The clustering of usernames didn’t seem very effective and the usernames tracked were almost completely random.

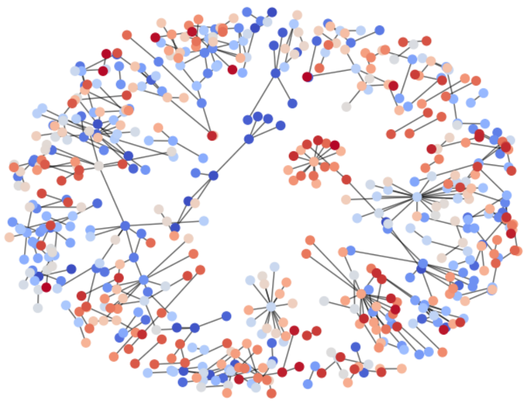
The following was the output of when I tracked the usernames and hashtags and their occurrences in each cluster:

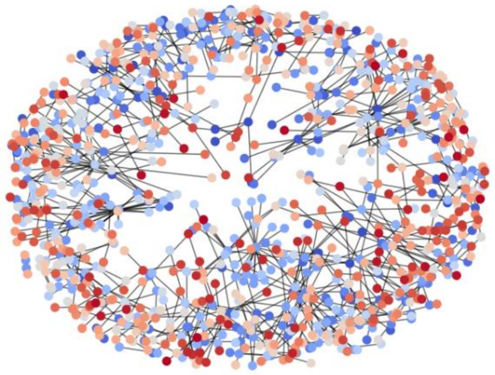


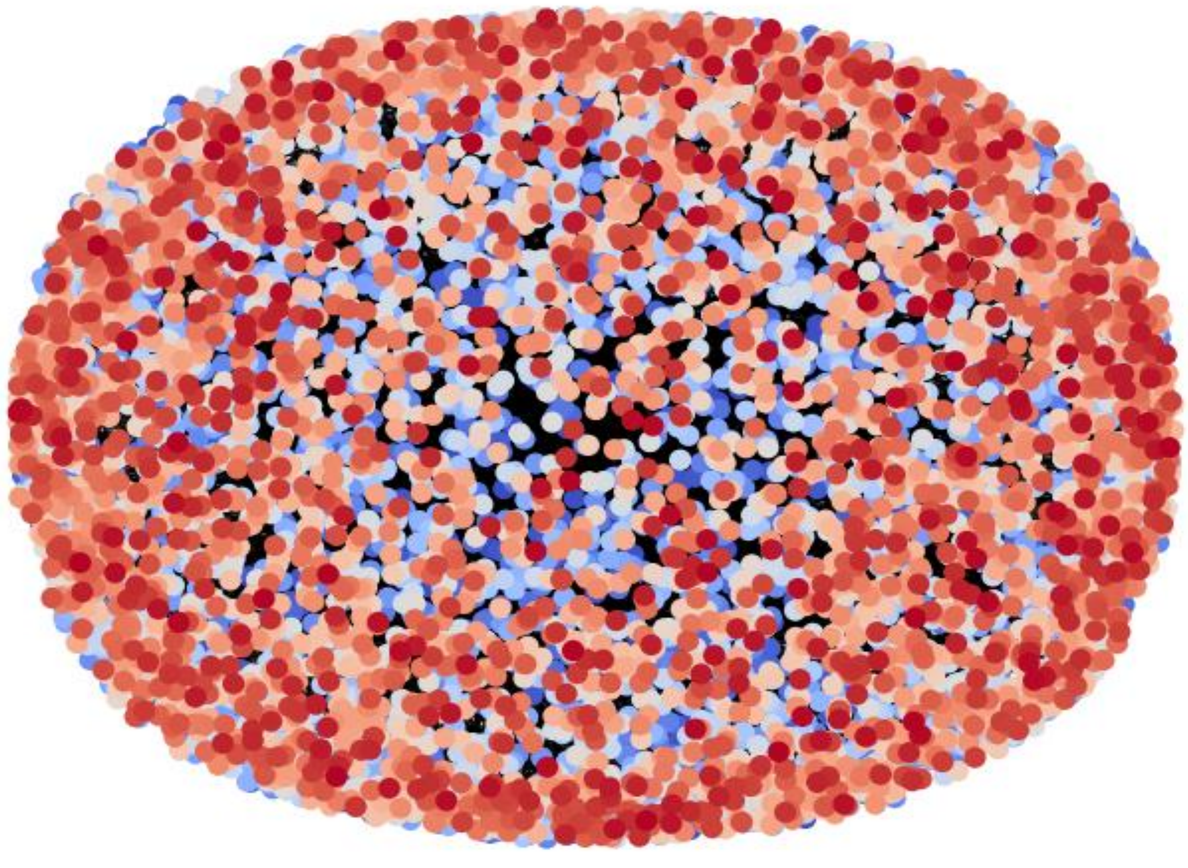
```
        === Cluster 0, Size: 45984 ===
=== Usernames ===
Counter({'@LFC': 3, '@DemmySports': 2, '@SStream777': 2, '@koora4day0': 2, '@CelalBiyikli': 2, '@Okwaro043': 1, '@The_guardi': 1, '@Abir77835613': 1, '@fre
shfener': 1, '@AlfyLoay': 1, '@arteta_era': 1, '@FDLM30': 1, '@Top5Vid': 1, '@FTSportsNews1': 1})
=== Hashtags ===
Counter({'#LIVATL': 1, '#PSG': 1, '#Dortmund': 1, '#Liverpool': 1, '#AtleticoMadrid': 1, '#coronavirusturkey': 1})
        === Cluster 1, Size: 45984 ===
=== Usernames ===
Counter({'@MarkusPeatling': 2, '@MarlosMoreno22': 2, '@facvnde': 1, '@zzz_kubs3on': 1, '@wilkowicz': 1, '@rileymarshy': 1, '@Joao_fxp7': 1, '@Marlon_Felipe
06': 1, '@MarlonSoaress_': 1, '@MarllonLoopez': 1, '@Marlinaayu': 1, '@GurjitAFC': 1, '@watch_speck': 1})
=== Hashtags ===
Counter({'#Arsenal': 1})
        === Cluster 2, Size: 45984 ===
=== Usernames ===
Counter({'@MarlosMoreno22': 2, '@Abir77835613': 1, '@laoctavasports': 1, '@zzz_kubs3on': 1, '@wilkowicz': 1, '@MarkusPeatling': 1, '@sonahongsona': 1, '@Ma
rlusSoares': 1, '@Joao_fxp7': 1, '@Marlon_Felipe06': 1, '@MarlonSoaress_': 1, '@MarllonLoopez': 1, '@Marlinaayu': 1, '@GurjitAFC': 1})
=== Hashtags ===
Counter({'#Arsenal': 1})
        === Cluster 3, Size: 45984 ===
=== Usernames ===
Counter({'@MarlosMoreno22': 2, '@JustSuritas': 1, '@Planet_Stream__': 1, '@zzz_kubs3on': 1, '@wilkowicz': 1, '@MarlusSoares': 1, '@Joao_fxp7': 1, '@Marlon_
Felipe06': 1, '@MarlonSoaress_': 1, '@MarllonLoopez': 1, '@Marlinaayu': 1, '@GurjitAFC': 1, '@MarkusPeatling': 1, '@watch_speck': 1})
=== Hashtags ===
Counter({'#Arsenal': 1})
        === Cluster 4, Size: 45984 ===
=== Usernames ===
Counter({'@MarlosMoreno22': 2, '@FootballTHS': 1, '@TLandHuddleClub': 1, '@zzz_kubs3on': 1, '@wilkowicz': 1, '@MarlusSoares': 1, '@Joao_fxp7': 1, '@Marlon_
Felipe06': 1, '@MarlonSoaress_': 1, '@MarllonLoopez': 1, '@Marlinaayu': 1, '@GurjitAFC': 1, '@MarkusPeatling': 1, '@watch_speck': 1})
=== Hashtags ===
Counter({'#Arsenal': 1})
        === Cluster 5, Size: 45984 ===
=== Usernames ===
Counter({'@MarkusPeatling': 2, '@Cristitorodri19': 1, '@MiiKeLMsT': 1, '@zzz_kubs3on': 1, '@wilkowicz': 1, '@MarlusSoares': 1, '@MarlosMoreno22': 1, '@Joao
_fxp7': 1, '@Marlon_Felipe06': 1, '@MarlonSoaress_': 1, '@MarllonLoopez': 1, '@Marlinaayu': 1, '@GurjitAFC': 1, '@watch_speck': 1, '@sonahongsona': 1})
=== Hashtags ===
Counter({'#Arsenal': 1})
```

4. Capturing and Organising User and hashtag Information

a. My script mentions.py was used to generate user interaction graph based on a network of users and the users they had mentioned. This is done by first using the pandas module to read in the in a normalised json form to allow it to be iterated by rows easily. For each row of the table, the get_mentions() method is called, which will return the username of the current user, and a list of all usernames that were mentioned. Then for each mentioned user, an edge will be created between the two usernames, if one does not already exist, with the two nodes representing the two users. Matplotlib is then used to visually represent this data. I only used 10000 tweets to form this graph as the script took a long time to run due to the fact it is exponential. The graphs are hard to decipher info from so below I have three graphs: One of the mention interaction graph after 500 tweets, then 1000 tweets and finally 10000 tweets, to show how these graphs are built up.

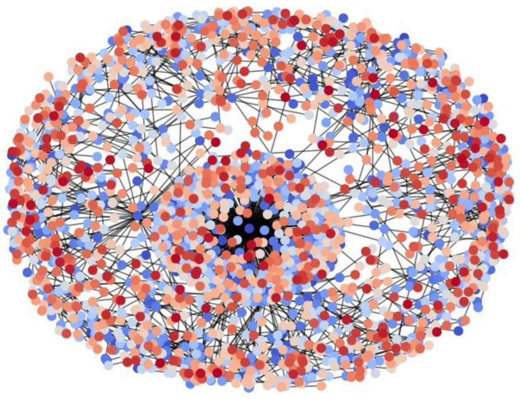




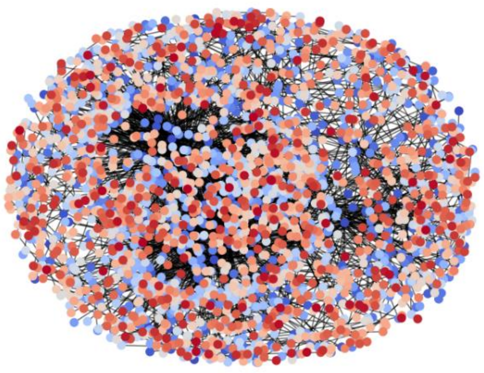


To capture a replies, retweets and quote tweets network, I had to stream a new set of data which took in every field of the tweet object, and not just the special ones I had selected. This meant that my results were from a dataset of just over 9000 tweets. The networks for these all worked pretty much the same as I basically just created an edge between two nodes (representing usernames), for each of the three interactions: retweets, replies and quote tweets. The results were as follows:

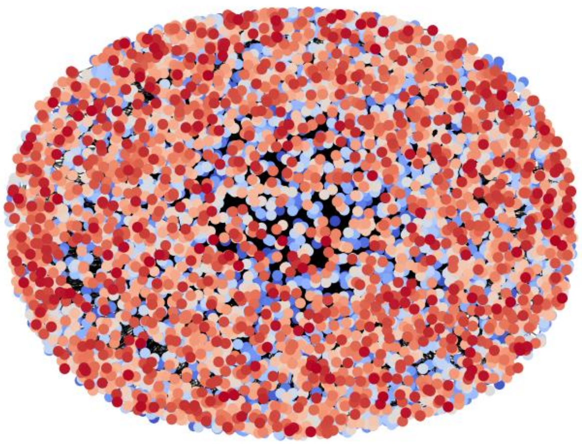
For my reply graph, there was 2197 nodes and 1516 edges. This generated the following graph:



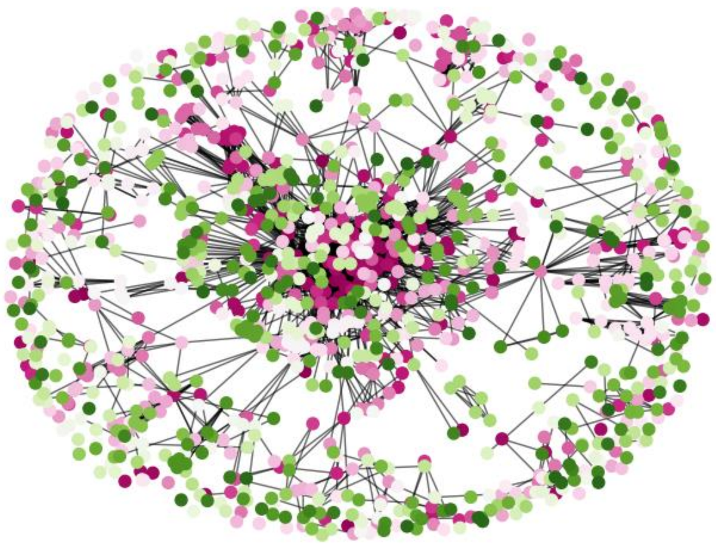
For the Quote tweet graph, there was 2699 nodes and 2356 edges present in the graph. This generated the following graph:



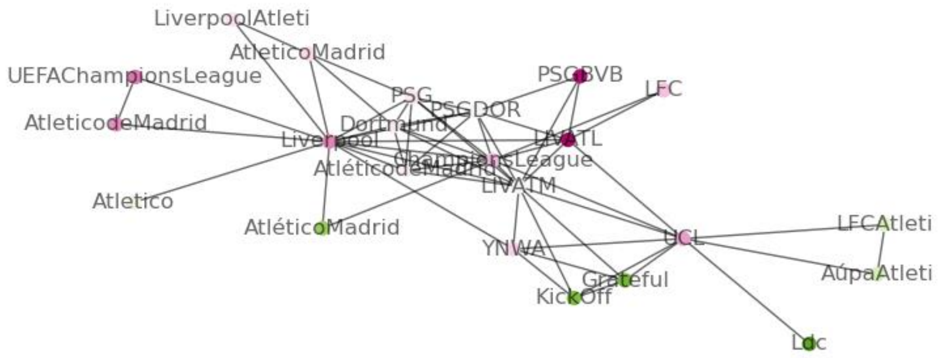
For the retweet graph, there was 6135 nodes and 5501 edges present. This generated the following graph:



b. The script hashtags.py was used to capture hashtag information occurring together. This was very similar to the script used in part a, in the sense that I first normalised the data using the pandas then called a method to return the username and a list of the users hashtags. The program differs however with what I do with he data. For hashtags I create an edge between every node, with each node representing a hashtag, which appear in the same tweet. For example a tweet like the following: “I love Liverpool #liverpool #lfc #ynwa” would create three edges; one between Liverpool and lfc, one between lfc and ynwa and one between Liverpool and ynwa. The following graph is the result of all 46000 tweets, which generated 1239 nodes and 2250 edges:



I then did the same with just 500 tweets, labelled the nodes and zoomed in on a specific section in order to show how the connections looked:



5. Network Analysis Information

For my mentions.py script which collects 10000 tweets and returned a graph showing the network of users connected by mentions, I got a graph containing 7154 nodes with 7392 edges. This means that there is more connections than there are users which results in an average degree of 2.1 for nodes in the graph. This means that each user on average is either mentioning a user of being mentioned by a user more than once, however the most frequent degree was 1 and so the most common interaction from users was to only mention or get mentioned once. The maximum degree of the graph was 105 which I assume belonged to a clubs account such as @LFC (Liverpool's official account.) It is hard to tell from the graph but I imagine that the average degree of 2.1 probably comes from a few highly mentioned users and several users only mentioned/mentioning once.

For my other_interactions.py script which collected all of my secondary stream tweets (approximately 9000), I returned a series of statistics for these graphs too.

For the replies network, there were 2197 nodes and 1516 edges. This means that there were more nodes than edges and so, in general the graphs nodes aren't very connected. This can further be shown by an average degree of 1.4 and the most common degree of nodes being 1. There was however a node with a degree of 398 which can clearly be seen from the graph in Part 4, which suggests at least one user had a lot of interactions. This would suggest the user was probably one of the clubs I was tracking in my stream and they had probably put out a tweet which requested a lot of user interaction. On further research this was the case as a lot of replies were to @SpursOfficial who had asked users to reply with their favourite spurs goals : https://twitter.com/SpursOfficial/status/1239582293905346561

For the quote tweets network, there was 2699 nodes and 2356 edges. This again means there are more nodes than edges, so in general the graph isn't very connected, however it is more so than the replies graph. This can be shown by the average degree of nodes being 1.7 but the most common degree of nodes still being 1. There was however a few quote tweets with large degrees as can be seen from the graph from section 4, with the node with the largest amount of degrees having 281. This would suggest perhaps a club or a few clubs have tweeted out looking for a large number of interactions, which has resulted in this large number.

For the retweets network, there are 6135 nodes and 5501 edges. This is again more connected than replies and quote tweets however still isn't very well connected, as there are more nodes than there are edges. This can be shown by the average degree of the nodes in the graph being 1.8 and the most common degree of nodes in the graph being one. The above graph is so crowded due to the huge number of nodes however the large areas of black show the ties that were being formed, with the node with the highest degree having 463 connections. This suggests that there was some accounts that a lot of users were retweeting from, but there was also some accounts receiving a lot of retweets.

For my hashtag information, things have to be looked at slightly different. This is because in previous graphs, I generated an edge with every node I added however with my hashtag graph, I added nodes even when there were no edges. Despite this there was still 1240 nodes and 2250 edges. These numbers may sound like a well connected graph however the most frequent degree of the nodes found in the graph was still 0. The average degree of a node was 3.6 which suggests each hashtag on average appeared alongside 3.6 other hashtags in the graph at one point. The largest degree of a node was 202 which suggests that for at least one hashtag, it appeared along side a lot of other hashtags. An example of this can be shown from my second screenshot from above of the Liverpool node which appeared to have a degree of ~10, meaning it appeared along side 10 other hashtags.