COURSE WORK – LEVEL 4
COMPSCI4077

NETWORK BASED SOCIAL MEDIA ANALYTICS

2333730b, Jordyn Brown

https://github.com/jordynbrown/2333730bWebScienceAE

Section 1: Introduction

The process involved the use of a number of applications, in order to both collect and analyse the chosen Twitter data. Twitter offers stream and REST APIs that are available for use to collect and download data about live tweets. In order to access these APIs, creation of a developer application was required. This grants the retrieval of publicly available information only, meaning that the APIs will allow the collection of the posted tweets themselves, but will not automatically award permission to private messaging conversations.

Through use of a developer account on Twitter, I first generated a new Twitter developer application – name: '2333730bWebScienceAE', app id: 17466571 – which, as previously mentioned, was required to run the **Twitter APIs**. One facility provided by Twitter applications was the generation of OAuth unique keys as a security measure, to both identify users and grant the correct access. The developer application itself generates a distinct set of keys and tokens which operate as the credentials delivered to the Twitter API on each request. Of these, there are four: the Consumer Key (API Key), Consumer Secret (API Secret), Access Token, and Access Token Secret. Once generated, these could now be used to access Twitter and collect the desired information.

With my developer application established, I then downloaded **MongoDB** onto my local machine in which to store the collected tweets. MongoDB requires the creation of a database and collection that I would use to direct the collated data into the database. In this case, the database and collection would be named 'webScience' and 'Coronavirus' respectively. Now with the Twitter application and MongoDB prepared, I next arranged the Python script which would be used to specify and format the collected tweets. My script was based on the code provided by 'SamDelgado' on GitHub, accessible here:

https://github.com/SamDelgado/twitter-to-mongo/blob/master/twitter-to-mongo.py#L51

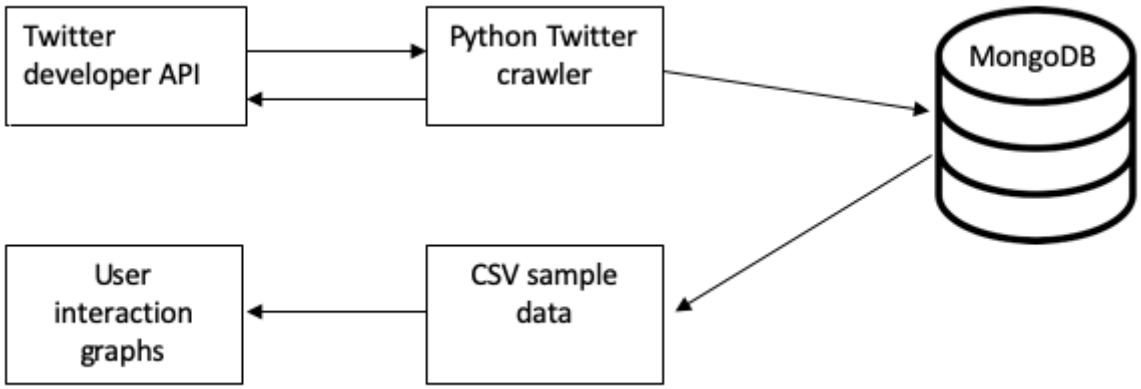
The script (described in Section 2) found in 'twitter_stream_api.py', executed the web crawler itself. This specified the fields of interest in the tweets and stored such data in the Coronavirus database. Majority of tweets were collected by the crawler from 12:00 – 14:00 on 04/03/2020. It should also be noted however that after preparing the script, I had run the crawler initially to test it was functioning properly. As such, the first couple of hundred results were collected on the 03/03/2020. After running the script for 2 hours as directed, the database held roughly 160,000 tweets.

Following the collection of tweets, the next step was to perform clustering on a sample of the collected data. This initiated data analysis of the tweets. Due to the performance of my system, I chose to take a sample of 5000 tweets (coronavirus_sample.csv). The clustering method chosen was K-means and the number of clusters was chosen to be 8. After execution, the script outputs the natural groups in the data based on usernames, hashtags and text. This time my script was based on examples provided on Stack Overflow:

https://stackoverflow.com/questions/27889873/clustering-text-documents-using-scikit-learn-kmeans-in-python?fbclid=IwAR13agTGUdH3e7Xdpt2x6ee6R8vrzjWCuguWgCgTklOcmcYBwVdO6ak8c3k

Section 3 will discuss this in detail. For ease of analysis, I wrote the clusters to a text file (Clusters.txt) which has been included on my GitHub.

User interaction graphs were generated using available software on Netlytic and Cortext. This captured username data, retweets and mentions to generate a series of network graphs which could be analysed and compared, with statistics drawn from the placed nodes.



Section 2: Data Crawl

Although the Twitter streaming and REST APIs provide different functionality, they can be used in conjunction to enhance the desired data collected by the user. With the aim to collect 1% of twitter data (a substantial amount of data), the streaming API seemed most appropriate. This API is used to collect large volumes of live tweets and can implement a filter to restrict the tweets obtained.
The streaming of twitter data using the web crawler was facilitated by the use of Tweepy. Streaming with Tweepy places majority of the control of the script with the Tweepy library, which arranges authentication of user credentials, establishes the connection, and both constructs and terminates the streaming session. Furthermore, it coordinates the routing of the collected data and reads it into the database. In the described code, it is evident that Tweepy was necessary to collect the volume of tweets desired with minimal error.

The streaming API, located in the twitter_stream_api.py file, first establishes the connection to the required port, creates the Mongo database and collection in which the data will be stored, and states that each collected tweet should be given a unique id within the database from which it can be indexed. Following this, keys and tokens are provided to ensure authentication of the user streaming the data. To collect and store the data, Tweepy provides a listener class called 'StdOutListener' that inherits from the 'StreamListener' class to which the collected tweets will be routed. Within this class, I have defined methods which will be called depending on the data passed to the listener. The 'on_data' method handles these messages and the execution of functions.

It was also necessary to parse the collected data into json, to format the tweets in the database. The following code selected the fields from the tweet that I was interested in, formatted the results and stored the data in the Coronavirus collection. After initially running the script, I found that it was essential to wrap all of this in a try/except as the program would throw a 'str_id' exception.

The 'on_error' method deals with any errors which my occur in the script. This was simply implemented to print the error message to the console.

Once the main functionality of the script had been completed, I used filters to only pull data of a specific topic or nature. In light of the current climate, I chose to track the most recent tweets regarding the Coronavirus by collecting those with the word 'Coronavirus' in the text, or using the hashtag '#Coronavirus'. While I know that there were more keywords or hashtags that I could have used, such as COVID-19, I knew that the volume of tweets would already be substantial. Due to performance, I chose to limit the words/hashtags tracked. Another, perhaps necessary filter for me, was to ensure that the tweets collected were limited to those of the English language.

Lastly, the main function invokes an instance of the listener. Here, Tweepy deals with authorisation and establishes the filter to only collect tweets with the keywords and language specified.

Implementing a REST API in conjunction with the stream API can enhance the crawling experience and the data collected. The REST API does not deal with data in real time but instead acts on historical data, allowing the user to create tweets, read user profiles and the data of followers. However, the REST API does not collect as much data as the stream API. To create a hybrid architecture, I decided to collect the followers of each user who was live tweeting. In this way, links between users can be analysed. Again, I restricted this due to system performance as some users had millions of followers.

Section 3: Grouping of tweets

With the tweets collected, I decided to write a script to cluster the usernames, hashtags and text data, and analyse the results. This code was based on the code discussed on the following Stack Overflow issue, found at:

https://stackoverflow.com/questions/27889873/clustering-text-documents-using-scikit-learn-kmeans-in-python?fbclid=IwAR13agTGUdH3e7Xdpt2x6ee6R8vrzjWCuguWgCgTklOcmcYBwVdO6ak8c3k

Due to the type and volume of data obtained, I decided to use scikit-learn K-means clustering. This time of clustering is described as being suitable for data without clearly stated groupings. K-means will find the natural categories in the data based on the number of desired clusters specified by the user. The algorithm will loop through the data, allotting a data point to each member of the set based on its properties and how similar it is to other data. Such data points will assign them to one of the k natural categories. While I had filtered to collect data related to the Coronavirus, I wanted to find the natural groupings within this data as there was no obvious way in which I could visually group the collection.

After exporting the MongoDB results into a csv file, I decided to take a sample of 5000 entries due to system performance, upon which to create the clusters. The script read the csv file 'coronavirus_sample.csv' and first extracted the necessary columns, assigning them to a variable: usernames, hashtags and tweets respectively. While each could have been performed separately, it seemed unnecessary. In order to create the clusters, integer data needs to be available. As such, the string data collected from the Tweets needed to be vectorized to enable analysis. Finding the ideal number of clusters was a matter of best guess. I first tried 8 clusters as seen in Figure 1. However, the numbers for each cluster did were drastically uneven. Trying 15 clusters as in Figure 2 seemed to help somewhat, but the spread did not seem to justify the need for so many clusters as the majority of these now held

only 1 data point. Choosing 10 clusters (Figure 3) appeared to be the best fit. The spread of data was preferable, although still not ideal.

Figure 1
Usernames in 8 clusters

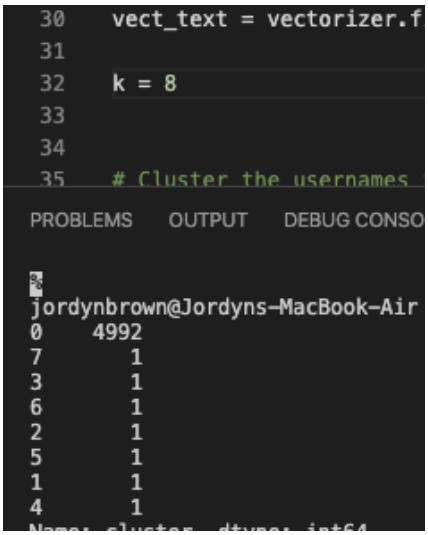


Figure 2
Usernames in 15 clusters

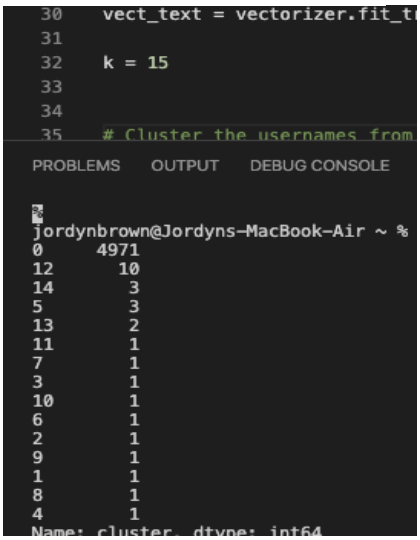
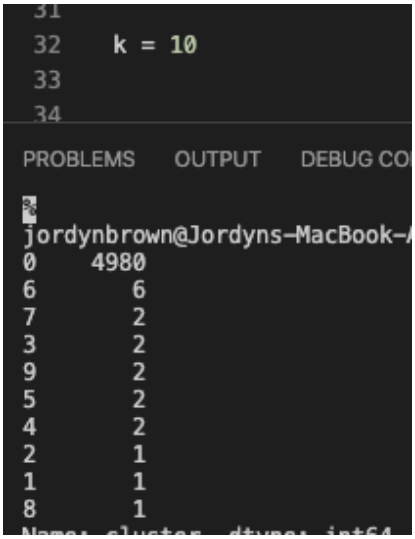


Figure 3
Usernames in 10 clusters



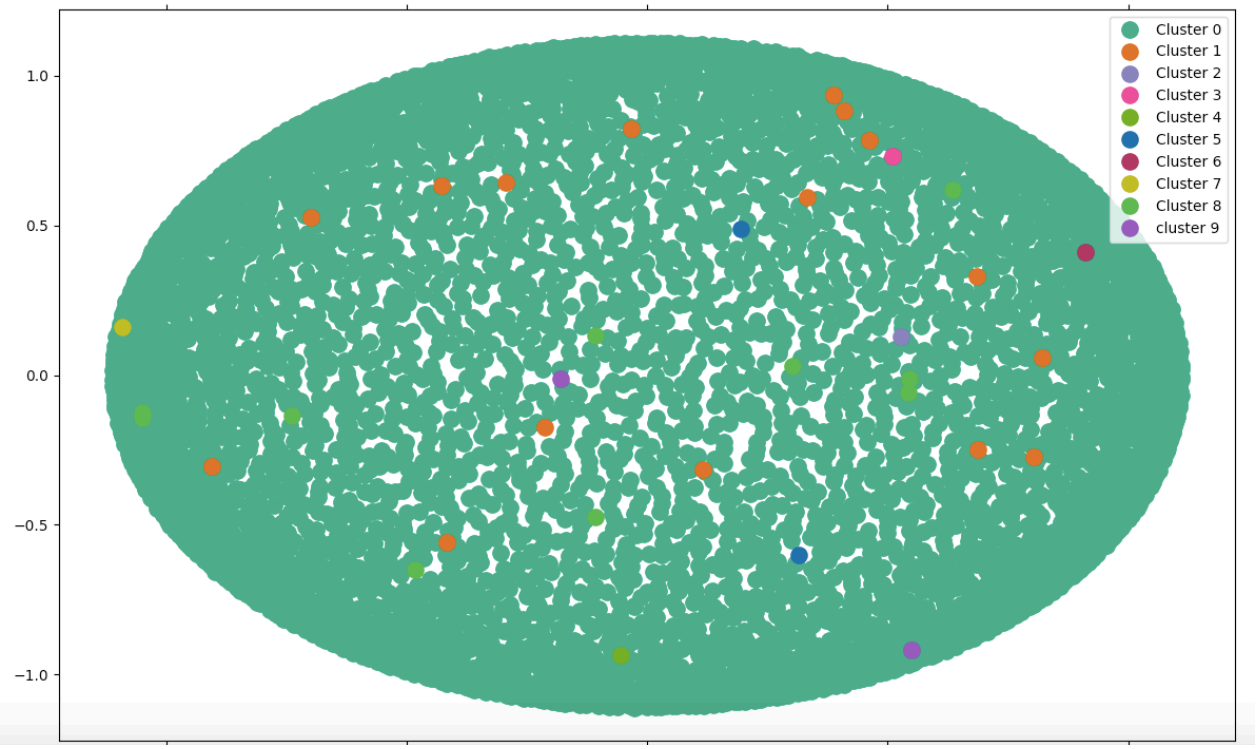
The file then outputs a list of 10 clusters for username, hashtags and text, writing this to a text file. To analyse such data, I decided to produce graphs for each of the subjects.

Outputted cluster data

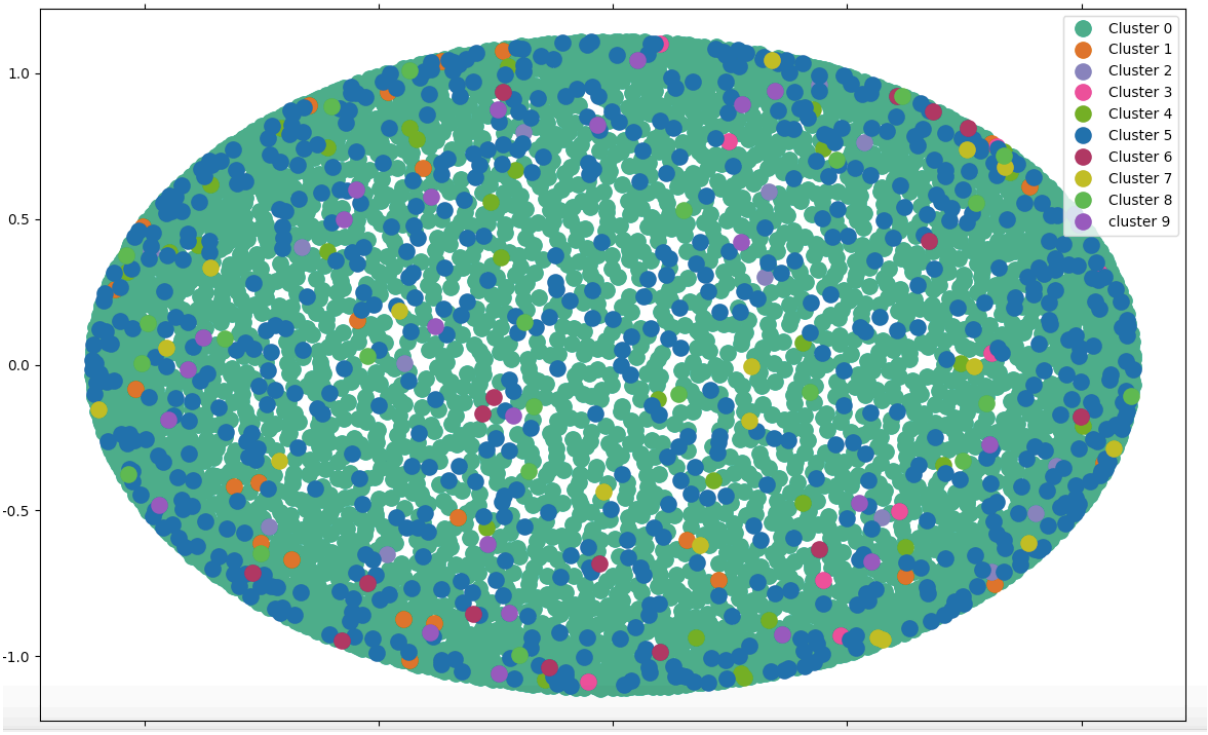
Clusters	Username	Hashtags	Text
0	friend hywhliqtup furloughs 441 ch8lrpu eqnjxiekbs batters directly businesses contr	accompany 2ioabkm5ba 2922 293 29510400 29510500 2clrtfww8t 2dtt8lia3f 2eo8fcq2nu 2g5ryy3nqn	positif kes dah hausofhilton di unnecessary malaysia guys 50 safe
1	born_blunt jordyinit cancels cancgseattle candi candicdebacad9 candidate candidates candygram30 cannabis	0awx5spv 239 35dszewi8d a1responder 5u4pvxgoa5 0xv0nzcniv 0nnxik4nvb 600 0ihy67gubl 2g5ryy3nqn	music ladygaga make sought racists illnes themlotsdad feared contagious actively
2	gentlemen cancelled cancels cancer cancgseattle candi candicdebacad9 candidate candidates candygram30	128 absteward 5u4pvxgoa5 a1responder 167 134m 35dszewi8d 900 26ymit0p 8up69mg2	äôt fckin rosnalju shouldn told wash hands sargee_ apparently racist
3	gagamediadotnet jordyinit candidates cancelling cancels cancer cancgseattle candi candicdebacad9 candidate	1970s 3sahovytsb 186 5u4pvxgoa5 a1responder 1fda0zurti 183 150 68 35dszewi8d	absolutely klopp jurgen asked class spot response futbolbible speaks sense
4	gog jordyinit candidate cancelling cancels cancer cancgseattle candi candicdebacad9 candidates	5u4pvxgoa5 a1responder 3sahovytsb 35dszewi8d 400 1chn0mjbmy 1o8wlkaget 130 5ajj3cpbdz 22rh966idc	cases new coronavirus total https confirmed deaths reports rt 15

5	impose jordyinit cancelled cancels cancer cancgseattle candi candicdebacad9 candidate candidates	1kdev_ a1responder 5u4pvxgoa5 35dszewi8d 1wbli7s8pu 18 1970s 14th 1r6algccda 3sahovytsb	trump just president wanna feel shit salary rt people coronavirus
6	coronavirusuk jordyinit capitalism cancels cancer cancgseattle candi candicdebacad9 candidate candidates	200304 1st a1responder 5u4pvxgoa5 35dszewi8d 21000 1fxocepxqd 3sahovytsb 1wcy8jpefx 1xjlt16xj6	photo 30s xoevilo unrelated confined manhattan completely mid woman patient
7	farting jordyinit cancelled cancels cancer cancgseattle candi candicdebacad9 candidate candidates	a1responder 5u4pvxgoa5 35dszewi8d 1800 15 044 22 198 1tas1k6mud 0tfknu0gjr	death rate toll globally coronavirus rises flu infected global killed
8	beholdisrael jordyinit cancels cancer cancgseattle candi candicdebacad9 candidate candidates candygram30	101 133 61culyozfx a1responder 5u4pvxgoa5 155zo76ots 5bepsja32p 40496 abe 35dszewi8d	schools universities italy coronavirus rt country shah_rymie earth antibodies close
9	ignorance jordyinit candygram30 cancels cancer cancgseattle candi candicdebacad9 candidate candidates	2009 1tiqjwy8vf a1responder 5u4pvxgoa5 35dszewi8d 43a856gwk9 060 21000 22 20	rt coronavirus https people äôs trump amp like new just

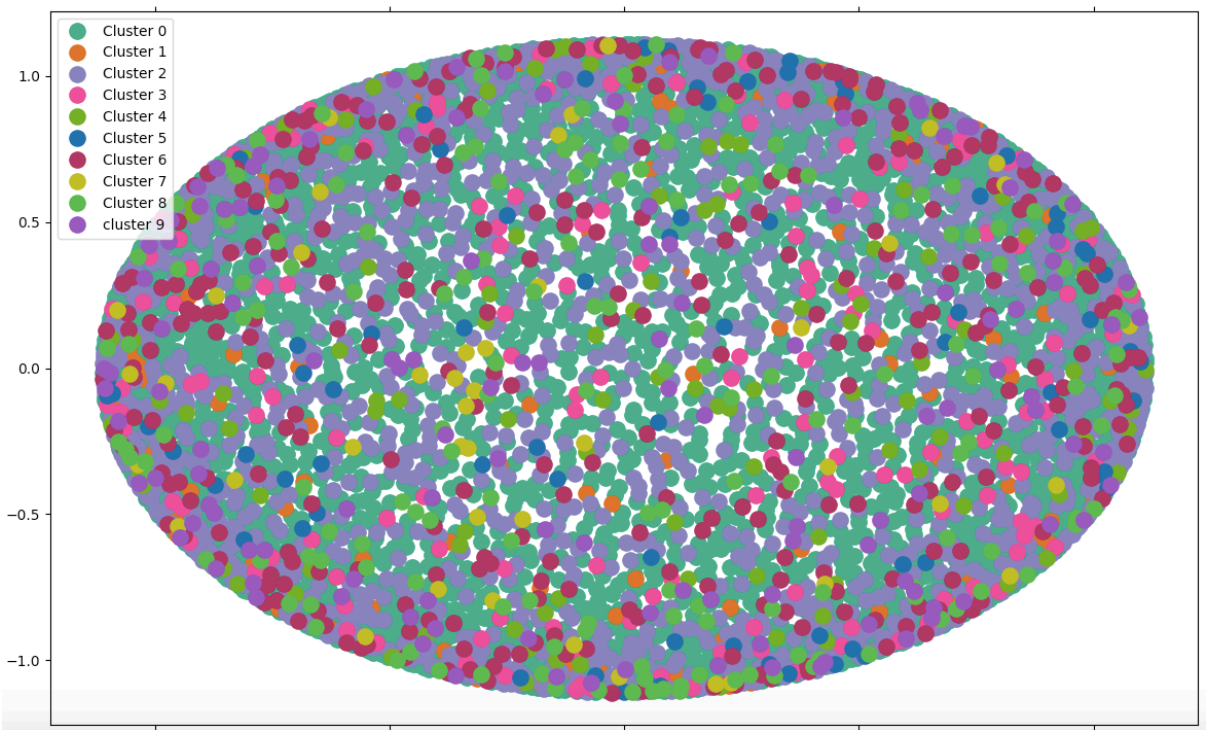
Cluster graph of usernames



Cluster graph of hashtags



Cluster graph of text



As can be clearly observed in the table and graphs, the clusters created for the 3 sets of data all differ significantly.

Decent clusters should have between roughly 5-30% of the data set in each cluster. For each topic the clusters should have contained between 250 – 1500 items per cluster. According to Figure 3, the username data instead created clusters containing between 0.02% - 99.84% of items. Figure 4 shows that the hashtag data created clusters containing between 0.18% - 82.54%, while figure 5 shows the text data creating clusters containing between 0.88% - 51.4%. From this we can derive that the number of clusters and the clusters themselves are not the model clusters we would want.

Max size →

Min size →

```
0    4127
5     715
4      31
1      24
9      23
8      23
7      17
6      16
2      14
3       9
Name: cluster,
Top hashtags p
```

Figure 4
Hashtags in 10 clusters

← Max size

← Min size

```
0    2570
2    1355
6     307
8     163
3     154
9     131
4     108
1      91
5      76
7      44
Name: cluster,
Top text per c
```

Figure 5
Text in 10 clusters

The uneven spread of data in all subjects, but particularly in the username clusters, tells us that the data is so widely spread that perhaps the algorithm cannot cluster meaningful data in the given number of clusters. For the given data, this could mean that the algorithm cannot find many natural groupings and so produces small groupings, then forcing the remaining data into one large cluster. As such, we could propose that there are an extremely small number of connections between the usernames in the sample data. The hashtag and text clusters improve respectively, telling us that natural groupings have been found more easily.

This can also clearly be understood from the cluster graphs. The username graph is overwhelmed by one colour, cluster 0. The hashtag graph displays the increased spread of data and balance between cluster 0 and 5. The text graph better meets expectations of what a cluster graph should look like, with the colours of all 10 clusters recognisable. However, in all graphs, the strong overlapping of clusters tells us that there is poor between-class variance.

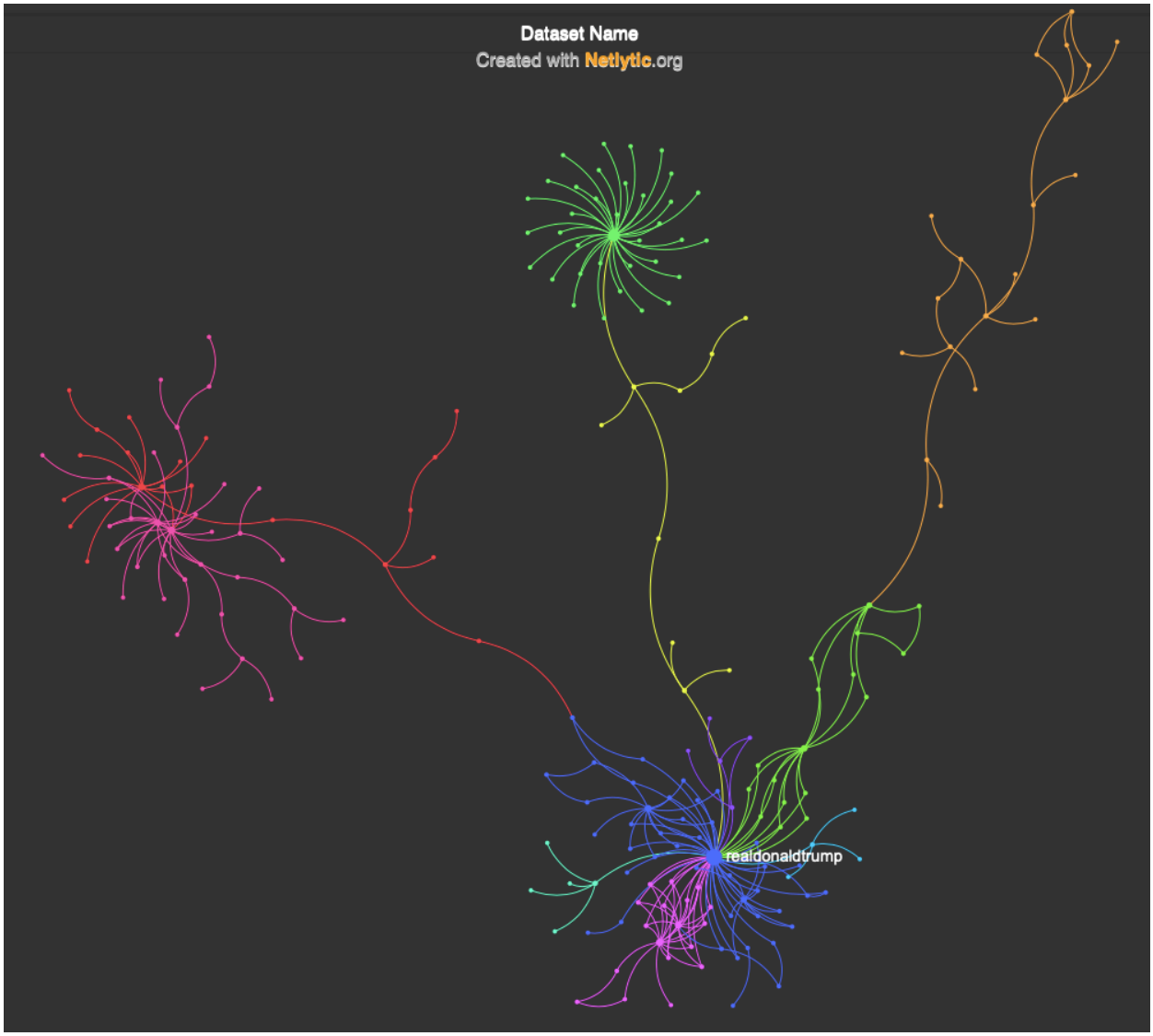
	Usernames	**Hashtags**	**Text**
Total size of data	5000	5000	5000
Average size of group	500		
Min size of group	1	9	44
Max size of group	4980	4127	2570

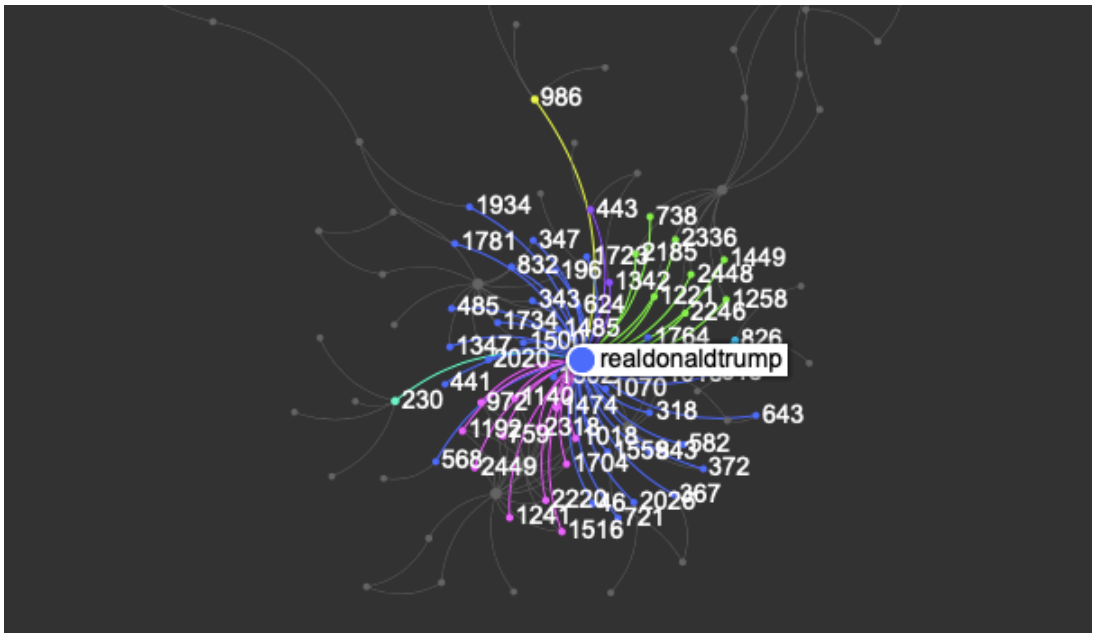
Section 4: Build a User Interaction Graph

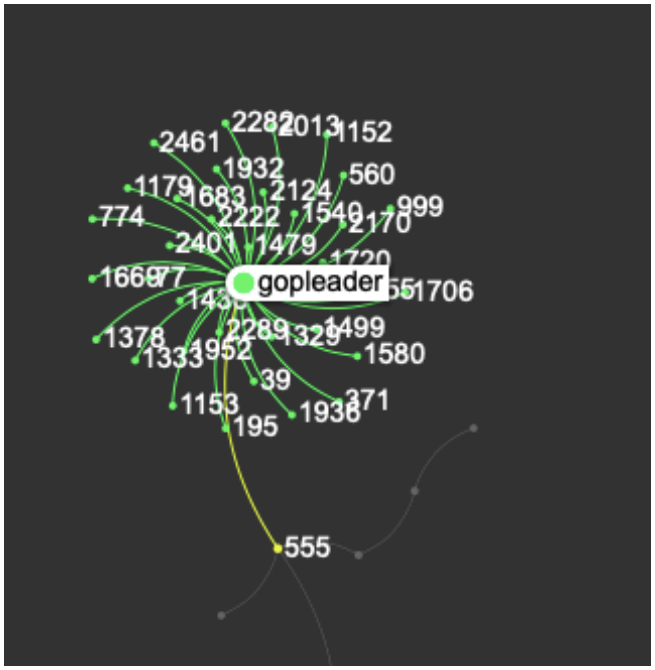
To build the user interaction graphs I used 'Netlytic'; a cloud-based platform for analysis of social media data. This software allows a maximum of 2500 entries in the csv uploaded to be analysed. As such, the graphed data is a sample of the original sample. For each of the following graphs I used the LGL layout most suitable for large graphs, as I feel that this most clearly shows the relationships between data.

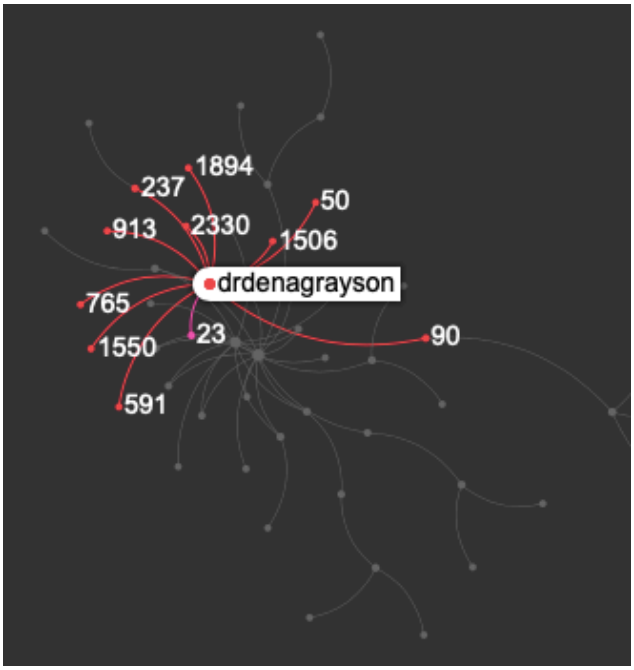
The network is a communication web, built by linking usernames that are mentioned in the text of user's tweets.

Users occurring together in general data

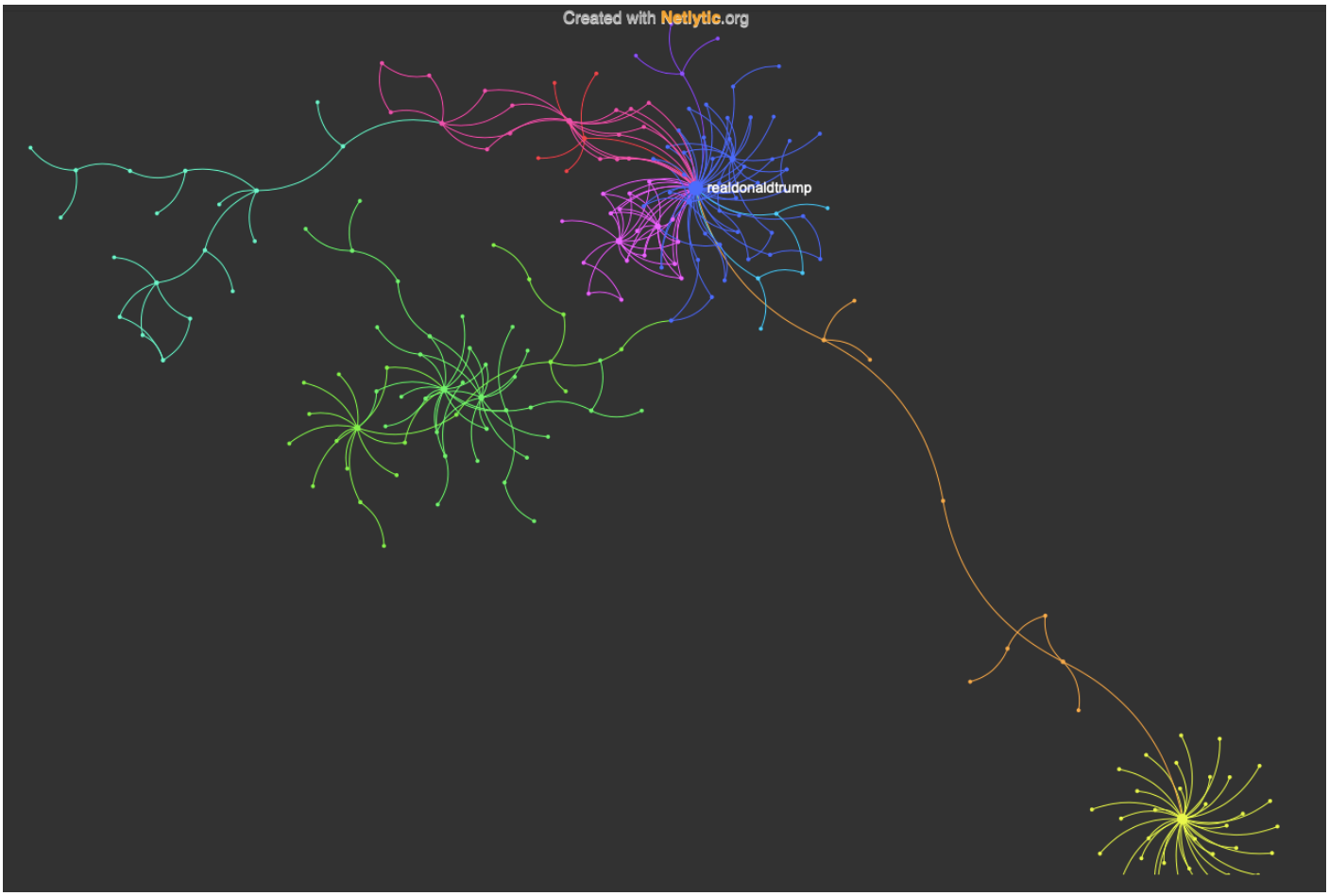


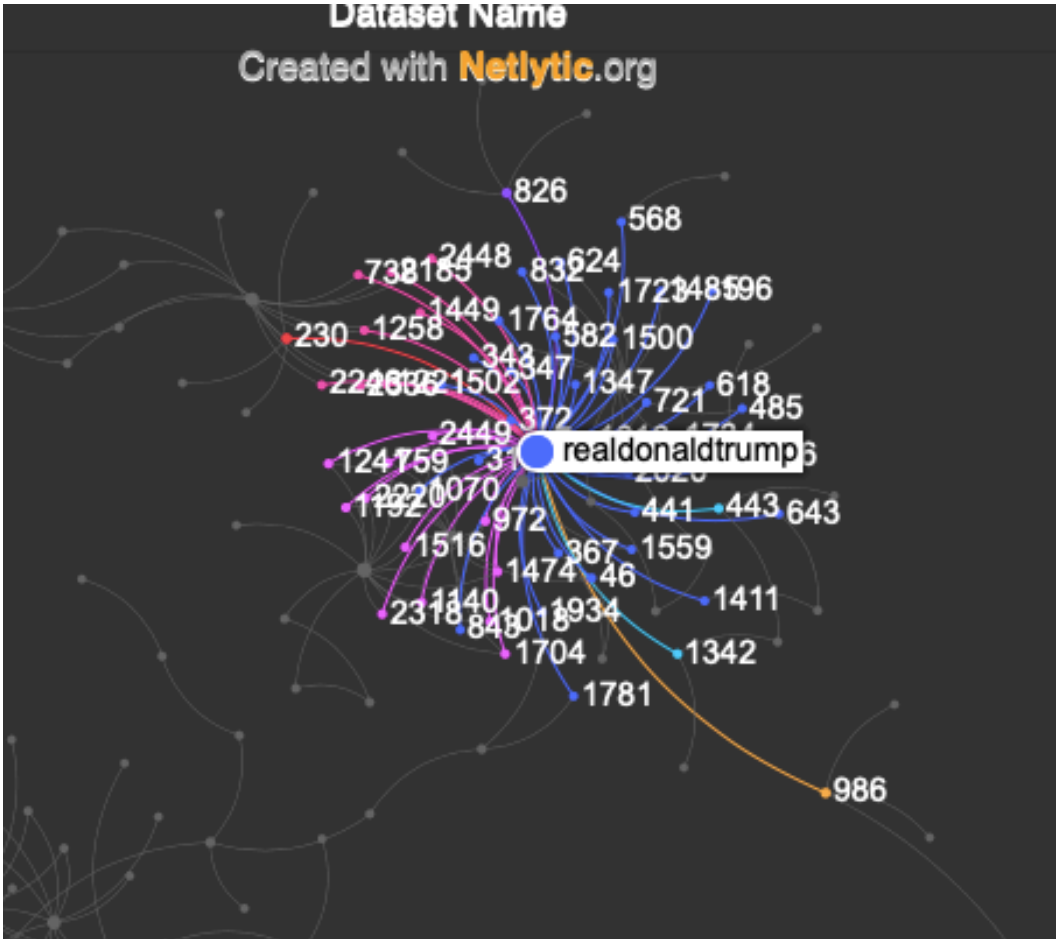


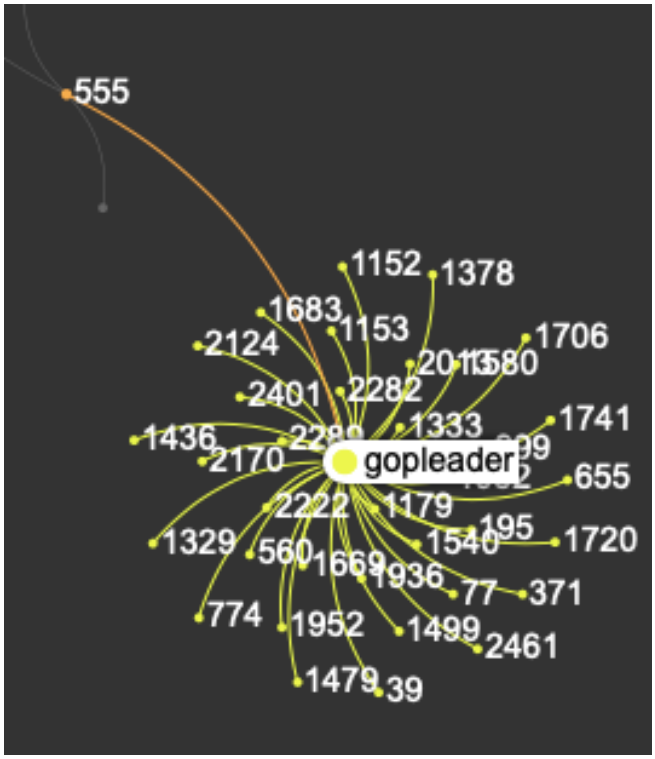


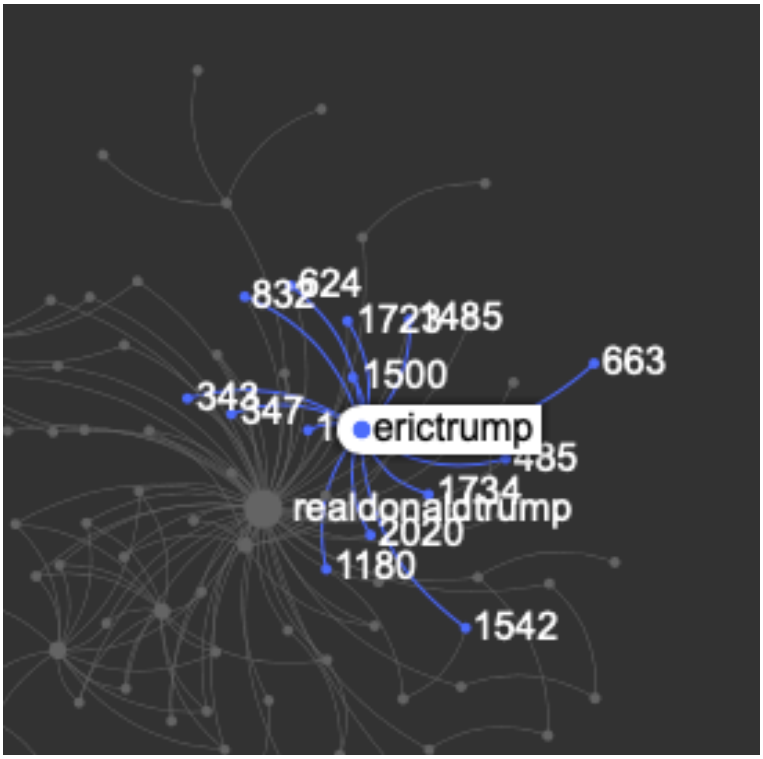


Retweet Network

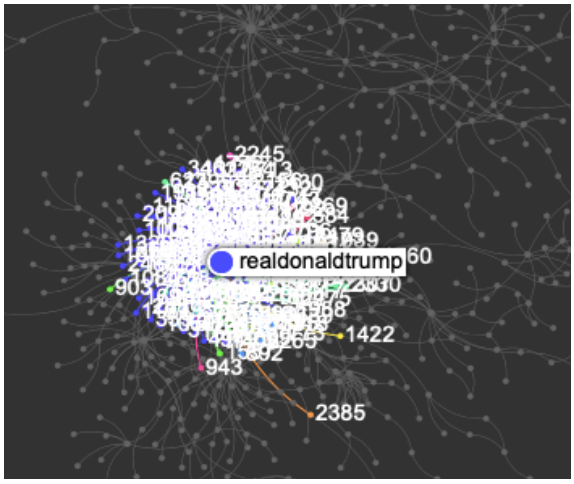


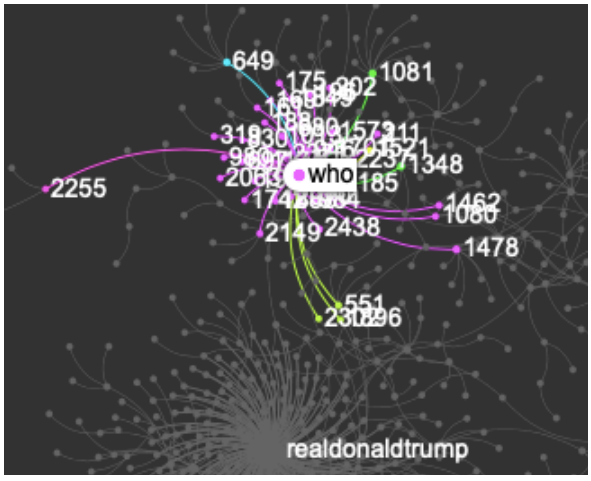

Dataset Name
Created with Netlytic.org
826
568
2448
624
832
738
1449
1764
230
1258
582
1500
343
347
1502
1347
721
618
485
372
2449
realdonaldtrump
1241
759
1070
441
443
643
972
1516
367
1559
1474
46
1411
1140
1934
2318
848
1018
1704
1342
1781
986


555
1152
1378
1683
1153
2124
1706
2401
2282
1741
1436
1333
2170
gopleader
655
2222
1179
195
1329
560
1540
1720
1669
1936
77
371
774
1952
1499
2461
1479
39


624
832
1723
485
663
1500
342
347
erictrump
485
realdonaldtrump
1734
2020
1180
1542

Quote Tweets Network





Analysis of networks:

	General Data Network	Quote Tweets Network	Retweet Network
Network properties	Diameter: 1 Density: 0.006245 Reciprocity: 0.000000 Centralization: 0.131800 Modularity: 0.729600	Diameter: 1 Density: 0.001428 Reciprocity: 0.000000 Centralization: 0.099980 Modularity: 0.818600	Diameter: 1 Density: 0.006297 Reciprocity: 0.000000 Centralization: 0.130700 Modularity: 0.730100

Given the data for each network in the table, we can draw the following conclusions:

Diameter: Relates to the size of the network. Calculates the largest distance between 2 nodes. In this case we can see that the size of each network was roughly the same. As a sample of data was used in each case, this makes sense. Even filtering for just quotes and retweets, the csv files were of roughly the same size.

Density: Calculated by dividing the number of existing ties by the number of possible ties. Illustrates how close participants are within a network. The nearer this number is to 1, the more interdependent the links are which suggests that users are talking with many others. In the graphs we see a significant decrease in the quote tweets network from the density of the general data network, as would be expected. However, the density increases once again when analysing the retweet network. This is perhaps a result of the spread of data having decreased. If the data is solely concentrated on retweets and not including all data, then it is expected that there will be more ties between users.
The numbers in each suggest that there are a small number of conversations happening between users. Perhaps these are numerous chats all happening between very small groups of users.

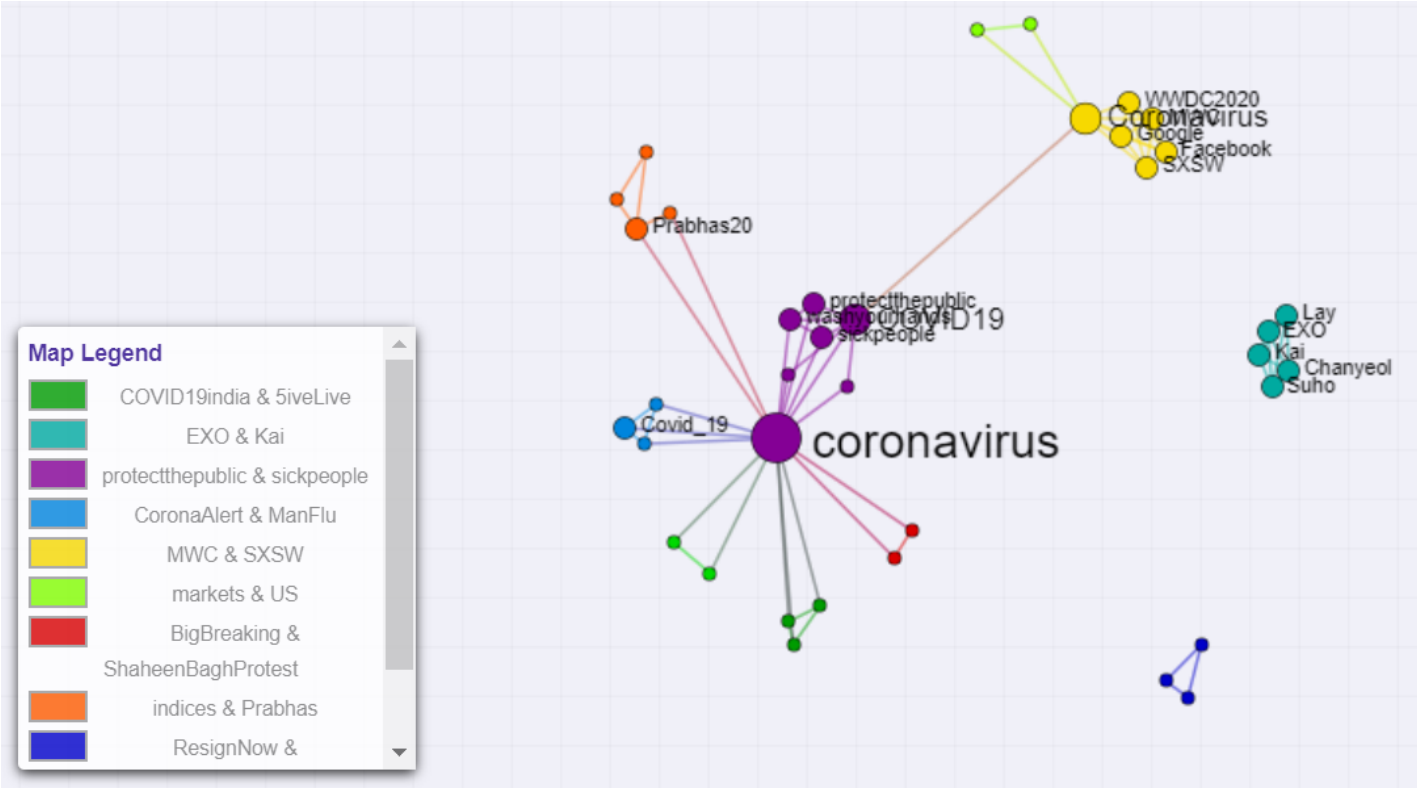
Centralization: A measure of the average degree centrality of network nodes. If values are closer to 1, there are a very small number of users who dominate the topics and conversation. If they are nearer to 0 then conversations are flowing more freely between users. For all of our network graphs, centralization is much closer to 0 and so we can assume that there are many participants all contributing to different topics, not focused on one or two concentrated topics.

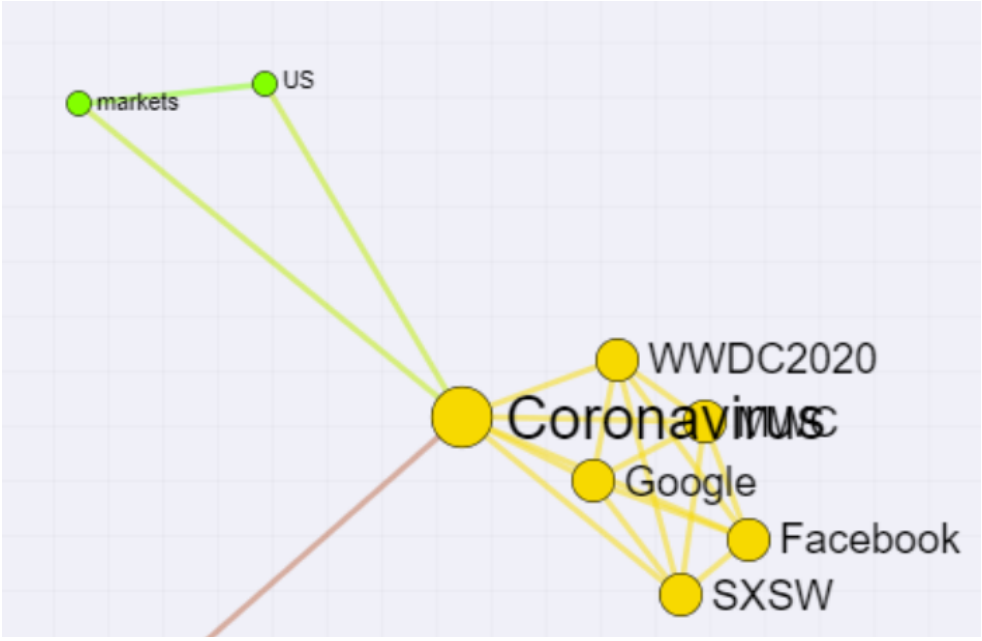
Modularity: According to the Netlytic site, values of modularity greater than 0.5 implies that there are clear separations between the clusters which represent individual groups in the network. All of our graphed data gives a modularity value higher than 0.5 and so we can assume that there are numerous distinct subjects/groups in the networks, rather than 1 or 2 clusters overlapping with varying levels of interaction.

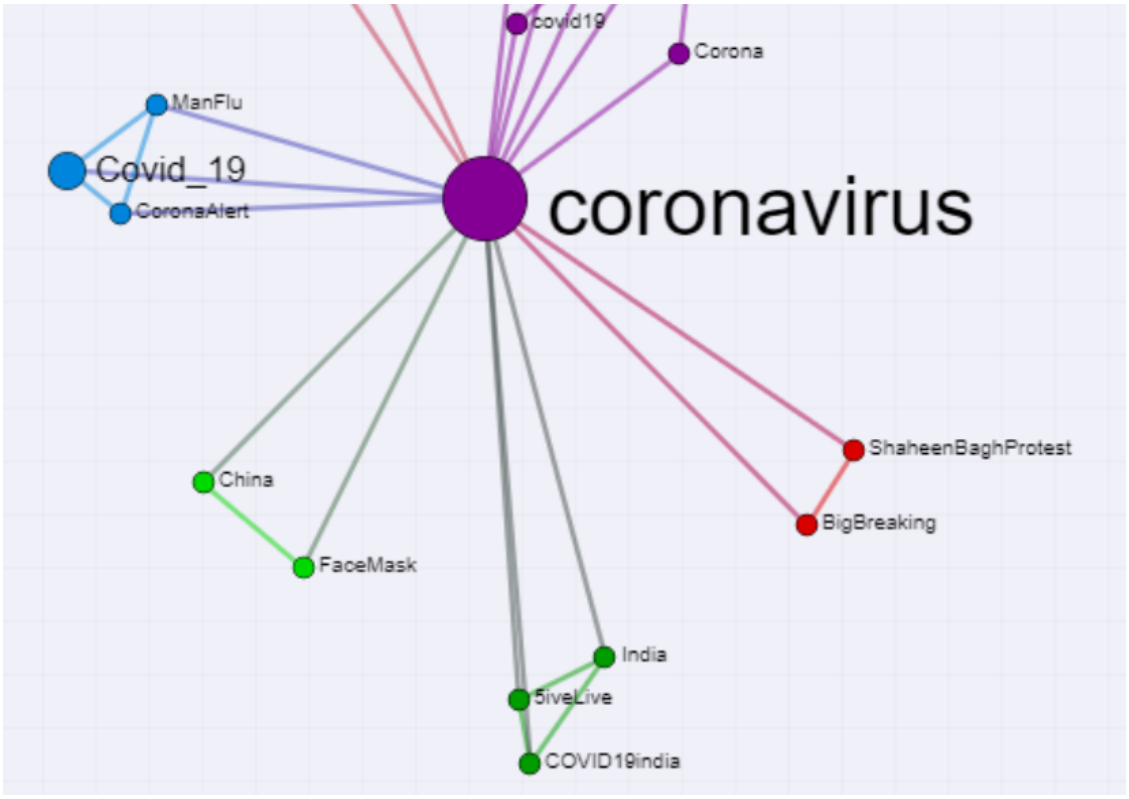
I have included close up images of clusters within the 3 graphs to exhibit the in degree of certain users. This tells us of their popularity, and as such these users are identifiable as super users.

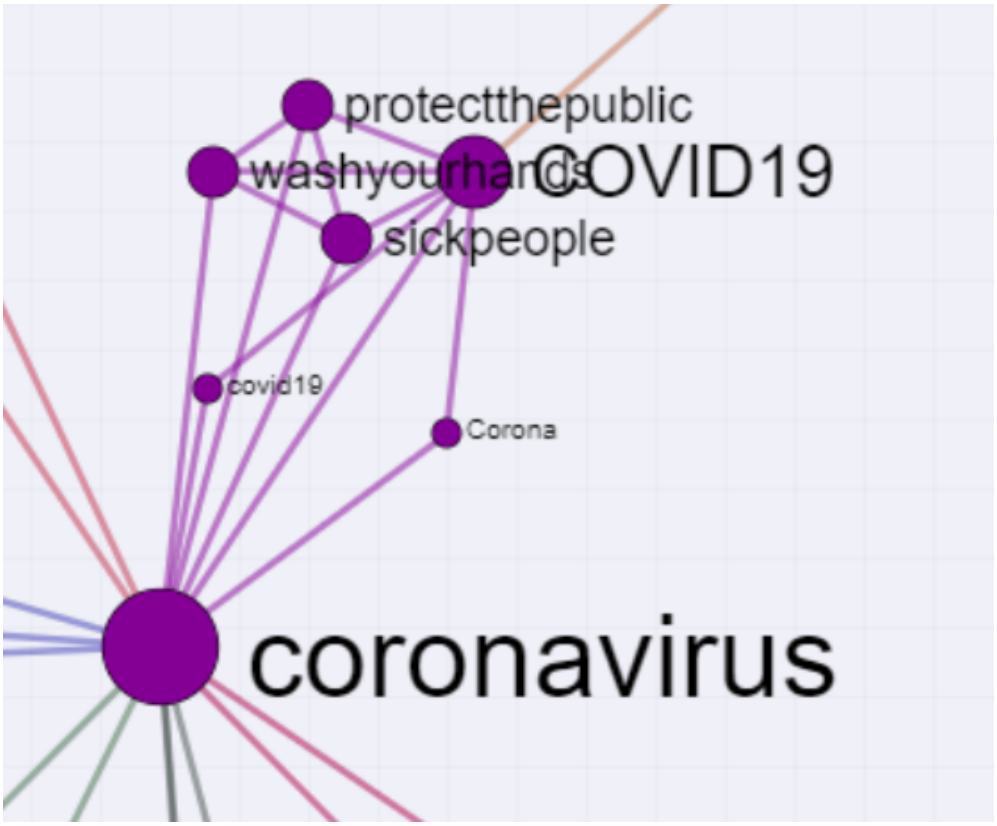
Hashtag Graphs

The following graphs are hashtag co-occurrence diagrams. They display the links between the most mentioned hashtags occurring together, hence the co-occurrence relationship. The subsequent figures show the hashtag co-occurrence graphs for a sample of the general data collected.






covid19
Corona
ManFlu
Covid_19
CoronaAlert
coronavirus
ShaheenBaghProtest
China
BigBreaking
FaceMask
India
5iveLive
COVID19india


protectthepublic
washyourhands
COVID19
sickpeople
covid19
Corona
coronavirus

Observing the hashtag co-occurrence graphs, it is clear that ‘coronavirus’ is the most used hashtag, spanning links to most other grouped subjects. Variations of ‘coronavirus’ such as ‘Corona’ and ‘covid19’ are linked closely as expected, as they will most likely be used in conjunction or in place of using the full term of ‘coronavirus’. Hashtags ‘washyourhands’, ‘sickpeople’ and ‘protectthepublic’ also possess co-occurrence links, which also meet expectations when talking about healthcare.

‘India’, ‘5iveLive’ and ‘COVID19india’ are appropriately grouped, all with links to the same geographical area.

‘China’ and ‘FaceMask’ were also grouped as having co-occurrence links. This could be an implication of recent news reports.

Other hashtag groups relate to the cancelling of certain events for ‘Google’ and ‘Facebook’, which has spanned conversations about concerns with the US stock market. This follows in the links to ‘US’ and ‘markets’ shown on the graph.

Section 5: Network Analysis Information

Information about ties are automatically calculated with Netlytic when building the graphs. The figures below state show the statistics computed by the software.

Figure 6
Quote tweets network

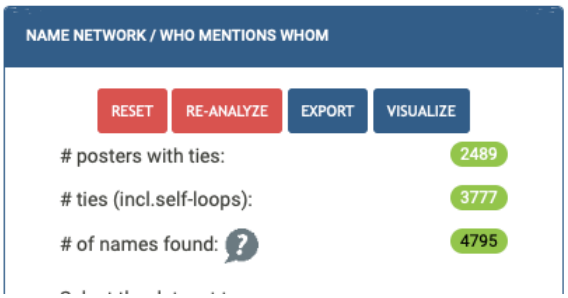


Figure 7
Retweet data network

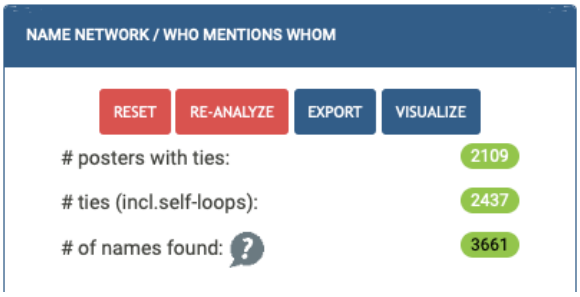
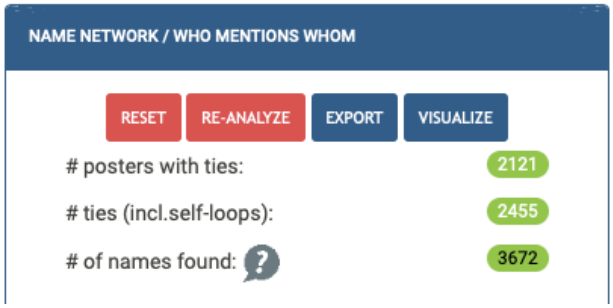


Figure 8
Network representing all data



In the figures above, the following number of ties are given for each graph on a set of 2500 users for each:

- Quote tweets network: 3777 ties
- Retweet data network: 2437 ties
- All data network: 2455 ties

The method given for calculating tries can be altered. I chose for the software to link a user to all names found in their tweet/s, and also to link users whose usernames co-occur in the same tweets.

References

Document Clustering with Python. http://brandonrose.org/clustering [Accessed 05 March 2020].

Nabila Shahid., 2015. Clustering text documents using scikit-learn kmeans in Python. https://stackoverflow.com/questions/27889873/clustering-text-documents-using-scikit-learn-kmeans-in-python?fbclid=IwAR13agTGUdH3e7Xdpt2x6ee6R8vrzjWCuguWgCgTklOcmcYBwVdO6ak8c3k [Accessed 07 March 2020].

Tavish Srivastava., 2013. Getting your clustering right (Part 1). https://www.analyticsvidhya.com/blog/2013/11/getting-clustering-right/ [Accessed 07 March 2020].

Jasmine Daly., 2016. K-Means Cluster Analysis of Poker Hands in Python. https://jasminedaly.com/2016-05-25-kmeans-analysis-in-python/ [Accessed 05 March 2020].

Oleg Żero., 2019. Top three mistakes with K-Means Clustering during data analysis. https://zerowithdot.com/mistakes-with-k-means-clustering/ [Accessed 09 March 2020].

Ankit Prasad., 2019. K-Means Clustering for Beginners using Python from scratch. https://medium.com/code-to-express/k-means-clustering-for-beginners-using-python-from-scratch-f20e79c8ad00 [Accessed 06 March 2020].

Marsden, Peter V., and O'Malley, A. James., 2009. The Analysis of Social Networks. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2799303/ [Accessed 10 March 2020].

Tompkins, Tori., 2018. Visualising Network Data in Power BI with Python Integration and NetworkX. https://adatis.co.uk/visualising-network-data-in-power-bi-with-python-integration-and-networkx/ [Accessed 10 March 2020].

Roesslein, Joshua., c2019-2020. Streaming with Tweepy. http://docs.tweepy.org/en/latest/streaming_how_to.html [Accessed 03 March 2020].

Hanneman, Robert A., and Riddle, Mark., Introduction to social network methods. https://faculty.ucr.edu/~hanneman/nettext/C8_Embedding.html [07 March 2020].

Trevino, Andrea., 2016. Introduction to K-means Clustering. https://blogs.oracle.com/datascience/introduction-to-k-means-clustering [Accessed 07 March 2020].

Tabassum, Jeniya., and Xu, Wei., 2018. Twitter API Tutorial. http://socialmedia-class.org/twittertutorial.html [Accessed 03 March 2020].

Miranda, LJ., 2017. How to stream Twitter using Python. https://ljvmiranda921.github.io/notebook/2017/02/24/twitter-streaming-using-python/ [Accessed 02 March 2020].

South, Laura., 2019. How to collect tweets from the Twitter Streaming API using Python. https://www.storybench.org/how-to-collect-tweets-from-the-twitter-streaming-api-using-python/ [Accessed 04 March 2020].

Cortext.https://documents.cortext.net/lib/mapexplorer/explorerjs.html?file=https://assets.cortext.net/docs/11ef6cfdc30a30fc24993f981ff38681# [Accessed 09 March 2020].

Abascal-Mena, Rocio., Lema, Rose., and Sèdes, Florence., 2014. From Tweet to Graph: Social Network Analysis for Semantic Information Extraction. https://hal.archives-ouvertes.fr/hal-01147320/document [Accessed 09 March 2020].

Delgado, Sam., 2014. https://github.com/SamDelgado/twitter-to-mongo/blob/master/twitter-to-mongo.py#L51 [Accessed 02 March 2020].

Netlytic., c2006 – 2019. https://netlytic.org [Accessed 11 March 2020].