COMPSCI4077 - Web Science (H)
Network based Social Media Analytics

Eleonora Della - 2244079D

11 March 2020

- The Github repository with the **source code** can be found here: `https://github.com/eleondella/COMPSCI4077`
- Instructions on how to run the program are specified in the **README.md** file
- The **sample data** to run the source code is in the SourceCode folder together with the python files: `https://github.com/eleondella/COMPSCI4077/blob/master/SourceCode/sample_data.json`

Contents

1 Introduction

1.1 Overview

The aim of this coursework was to build a Twitter Crawler which would be used to capture tweets using both the Streaming and REST APIs which would then be used for conducting analysis. Implementation was divided into four stages which are summarised as follows:

1. Capturing data using both Streaming and REST API
2. Grouping the tweets using K-Means Clustering and providing statistics for resulting groups
3. Organise user and hashtag information into dictionaries forming networks
4. Analysing the networks by giving network-based measures

The K-means clustering approach followed in part 2 was inspired by a similar implementation found on `https://github.com/fikriakbar-chng/Twitter-Data-Clustering`.
For parts 3 and 4, parts of the code implemented were taken by the Interaction Network.ipynb supplied in `https://github.com/ugis22/analysing_twitter`

1.2 Implementation

The Twitter Crawler and the analysis part has been implemented using the Python programming language. Furthermore, any tables, graphs and other visualisations or outputs have been exported accordingly into the main folder. The file structure for this coursework can be summarised by Figure 1.

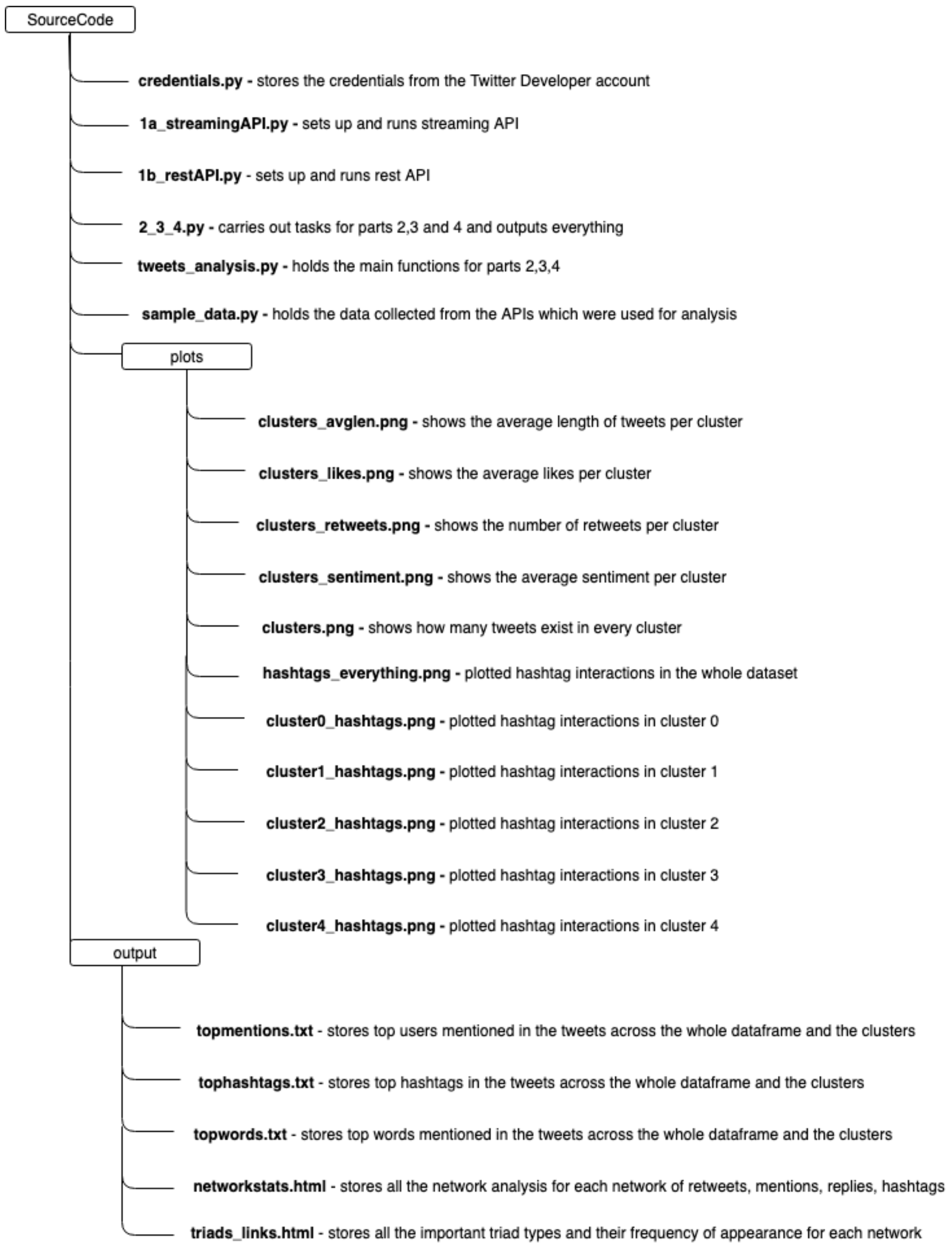

SourceCode
credentials.py - stores the credentials from the Twitter Developer account
1a_streamingAPI.py - sets up and runs streaming API
1b_restAPI.py - sets up and runs rest API
2_3_4.py - carries out tasks for parts 2,3 and 4 and outputs everything
tweets_analysis.py - holds the main functions for parts 2,3,4
sample_data.py - holds the data collected from the APIs which were used for analysis
plots
clusters_avglen.png - shows the average length of tweets per cluster
clusters_likes.png - shows the average likes per cluster
clusters_retweets.png - shows the number of retweets per cluster
clusters_sentiment.png - shows the average sentiment per cluster
clusters.png - shows how many tweets exist in every cluster
hashtags_everything.png - plotted hashtag interactions in the whole dataset
cluster0_hashtags.png - plotted hashtag interactions in cluster 0
cluster1_hashtags.png - plotted hashtag interactions in cluster 1
cluster2_hashtags.png - plotted hashtag interactions in cluster 2
cluster3_hashtags.png - plotted hashtag interactions in cluster 3
cluster4_hashtags.png - plotted hashtag interactions in cluster 4
output
topmentions.txt - stores top users mentioned in the tweets across the whole dataframe and the clusters
tophashtags.txt - stores top hashtags in the tweets across the whole dataframe and the clusters
topwords.txt - stores top words mentioned in the tweets across the whole dataframe and the clusters
networkstats.html - stores all the network analysis for each network of retweets, mentions, replies, hashtags
triads_links.html - stores all the important triad types and their frequency of appearance for each network

Figure 1: File Structure

In order to run the files as given please follow the instructions given on the **README.md** file in the Github repository.

1.3 Data Collected

Data was stored for both API's by using MongoDB in a database called "twitterdb" and a collection called "tweets". In total, I have collected 15470 tweets.

For the streaming API I ran it on the 7th of March between 19:10 and 20:10 UK time and collected a total of 10470 tweets. Then, I run the rest API at 20:10 to collect 5000 more tweets based on the top trends in the UK. I didn't run the two concurrently as there would be less chance of a connection since the streaming API collects data in real time than from previous tweets. The data collected has been exported by

MongoDB using the following command into a JSON file by running the following command in the terminal:

```
mongoexport --collection=tweets --db=twitterdb --out=sample_data.json
```

2 Data Crawl

In order to get access to Twitter APIs, the Tweepy library [3] was used due to its simple integration in Python. It was very straightforward to use and with great documentation online.

2.1 Streaming API

I used Twitter Streaming API for collecting 1% data by filtering all tweets with English as their set language and using the United Kingdom's coordinates by latitude using the following function:

```
myStream.filter(languages = ["en"], locations=[-6.38,49.87,1.77,55.81], is_async=True)
```

The filtering is done on an instance of the StreamListener class, of which the most important function worth mentioning is the `on_data(self, data)` function where a connection is established with the database, data from Twitter is encoded into JSON, the date format has been changed and finally inputted into MongoDB.

```
def on_data(self, data):
try:
    # Connects to the MongoDB
    client = MongoClient()

    # Use twitterdb database. If it doesn't exist, it will be created.
    db = client.twitterdb

    # Decode the JSON from Twitter
    datajson = json.loads(data)

    format_created_at = datetime.strptime(datajson['created_at'], '%a %b %d %H:%M:%S +0000 %Y')

    datajson['created_at'] = format_created_at

    #insert the data into the mongoDB into a collection called "tweets"
    db[self.collection].insert_one(datajson)
        except Exception as e:
            print(e)
```

2.2 Rest API

In order to implement the hybrid architecture and enhance the data crawling, I have used the REST API with it's **Trends** and **Crawler** features. On top of the Streaming API filtering done, additional filtering was done by collecting 5000 tweets from the **top five** trends in the UK at the time. This was done by finding the WOEID (*"Where On Earth IDentifier"*) of the United Kingdom by looking through the world trends and then getting the top trends by WOEID specified. When the REST API is run the 5 top trends will be printed in the terminal. In the sample data used for analysis, the top 5 trends retrieved in the UK were:

1. #ENGvWAL
2. #SaturdayNightTakeaway
3. #BURTOT
4. #GreatestDancer

5. Burnley

The `tweepy.Cursor` was then set up to get 5000 tweets from all 5 trends but due to the REST API having a limit in the tweets retrieved per 15 minutes, a timer was set so that it could wait 15 minutes and then try again until all 5000 tweets were retrieved.

```
for tweet in tweepy.Cursor(api.search,
                 q=" OR ".join(names[0:5]),lang="en").items(5000):
    while True:
        try:
            client = MongoClient()
            # Use twitterdb database. If it doesn't exist, it will be created.
            db = client.twitterdb
            #print(tweet["_json"])
            # Decode the JSON from Twitter
            datajson = tweet._json

            format_created_at = datetime.strptime(datajson['created_at'],
            '%a %b %d %H:%M:%S +0000 %Y')
            datajson['created_at'] = format_created_at

            #insert the data into the mongoDB into a collection called twitter_search
            #if twitter_search doesn't exist, it will be created.
            db["tweets"].insert_one(datajson)
            break

        except tweepy.TweepError:
            time.sleep(60 * 15)
            continue
```

3 Grouping of Tweets

3.1 Method of Grouping: K-Means Clustering

The aim was to group together tweets by content analysis. My choice of method was K-Means clustering, which by identifying the number of clusters required (also known as centroids), every tweet (data point) is allocated to the nearest cluster, while keeping the centroids as small as possible. Before any tweets could be divided into clusters, they required to go under vectorisation which would then allow for the clustering algorithm to occur. The TFIDFvectorizer [1] was used from the `sklearn` library and was applied and fitted on the "text" field from each tweet by also removing any stopwords. After the tweets were vectorised into a TFIDF matrix, the K-Means clustering was carried out through the same library. For the K-means algorithm, 5 clusters were specified due to the reason of the 5 top trends being collected which seemed appropriate. The number of tweets were divided between the clusters as shown in Figure 2.

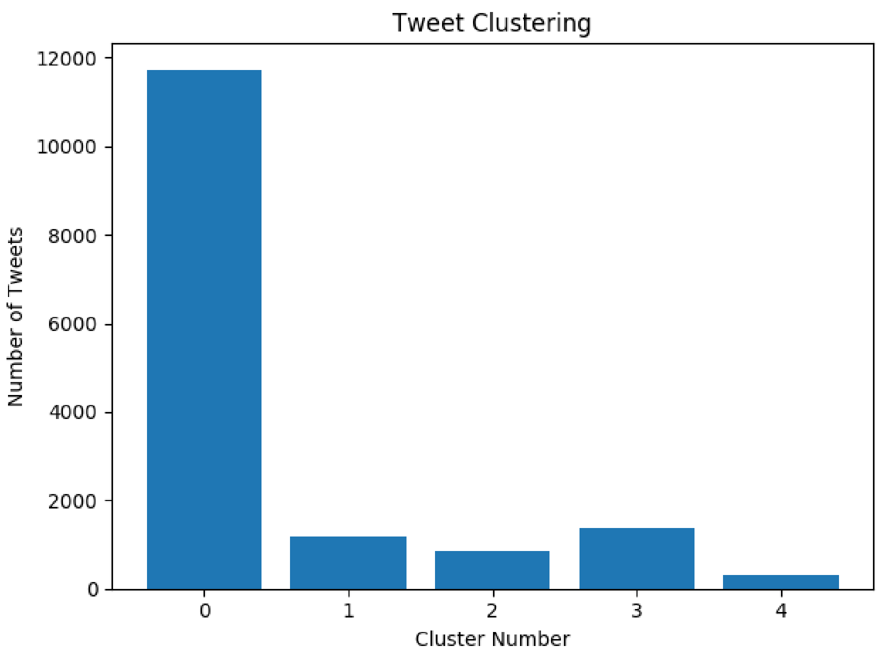


Figure 2: K-means: No. of Tweets per Cluster

3.2 Username & Hashtag Identification

For each of the clusters and also the whole dataset, appropriate statistics have been gathered to summarise the data which are also printed in the terminal when the code is run, as shown in Table 1.

Data	Average Followers No.	Average Tweet Text Length	Average No. of Likes	Average No. of Retweets	Average Sentiment
EVERYTHING	8565.88	66.40	0.71	9.75	0.27
Cluster 0	3915.38	63.07	0.19	1.17	0.25
Cluster 1	7875.28	74.10	2.08	5.59	0.32
Cluster 2	20004.90	66.86	3.00	13.44	0.46
Cluster 3	42686.63	86.27	2.62	38.54	0.18
Cluster 4	2899.14	72.96	0.23	211.40	0.49

Table 1: Statistics Obtained on Clusters and the Original Dataset

Furthermore, we also collected the top user mentions, hashtags and words for each and have stored them in the .txt files: **topmentions.txt, tophashtags.txt, topwords.txt** respectively.

Data	Top User Mention	Top Hashtag
EVERYTHING	itvtakeaway - 320	SaturdayNightTakeaway - 1146
Cluster 0	GreatDancerTV - 68	ENGvWAL - 172
Cluster 1	itvtakeaway - 255	SaturdayNightTakeaway - 1138
Cluster 2	GreatDancerTV - 201	GreatestDancer - 877
Cluster 3	LastWordOnSpurs - 213	BURTOT - 317
Cluster 4	EnglandRugby - 101	ENGvWAL - 293

Table 2: Top User Mentions and Hashtags

When trying to interpret Table 2 we can see that four out of the five trends were the top hashtags for the clusters, this shows that content was appropriately clustered by trend as expected. The last trend was not a hashtag, which is common with Twitter at trends don't have to be just hashtags, therefore explaining why #ENGWAL was the top hashtag for two of the clusters as it was indeed the top trend.

4 Capturing & Organising User and Hashtag Information

4.1 User Information

For this part, the aim was to come up with a way to capture user interactions and differentiate them by type of network. For my implementation, I have collected user interactions for **user mentions, retweets and replies** and have decided to represent them using a dictionary data structure which allowed me to capture not only the users interacting together but also how often they interact with one another like the one shown below from the mentions network.

```
{
    'Directions4Men': {'TimeToChangeWa1': 1, 'anotherhobo': 1, 'warringtonmus': 1},
    'DekaT42': {'NUFCTheMag': 1},
    'sjbryant': {'kwjourno': 1, 'Ant71': 1, 'Hurtigruten': 1},
    'StevenMacnamara': {'smh': 1}, 'Bigtoemangod': {'Freeeedog': 1},
}
```

The above dictionary can be interpreted as follows:

- **Directions4Men** has mentioned **TimeToChangeWa1**, **anotherhobo**, **warringtonmus** one time each in the sample tweets collected by our crawler.

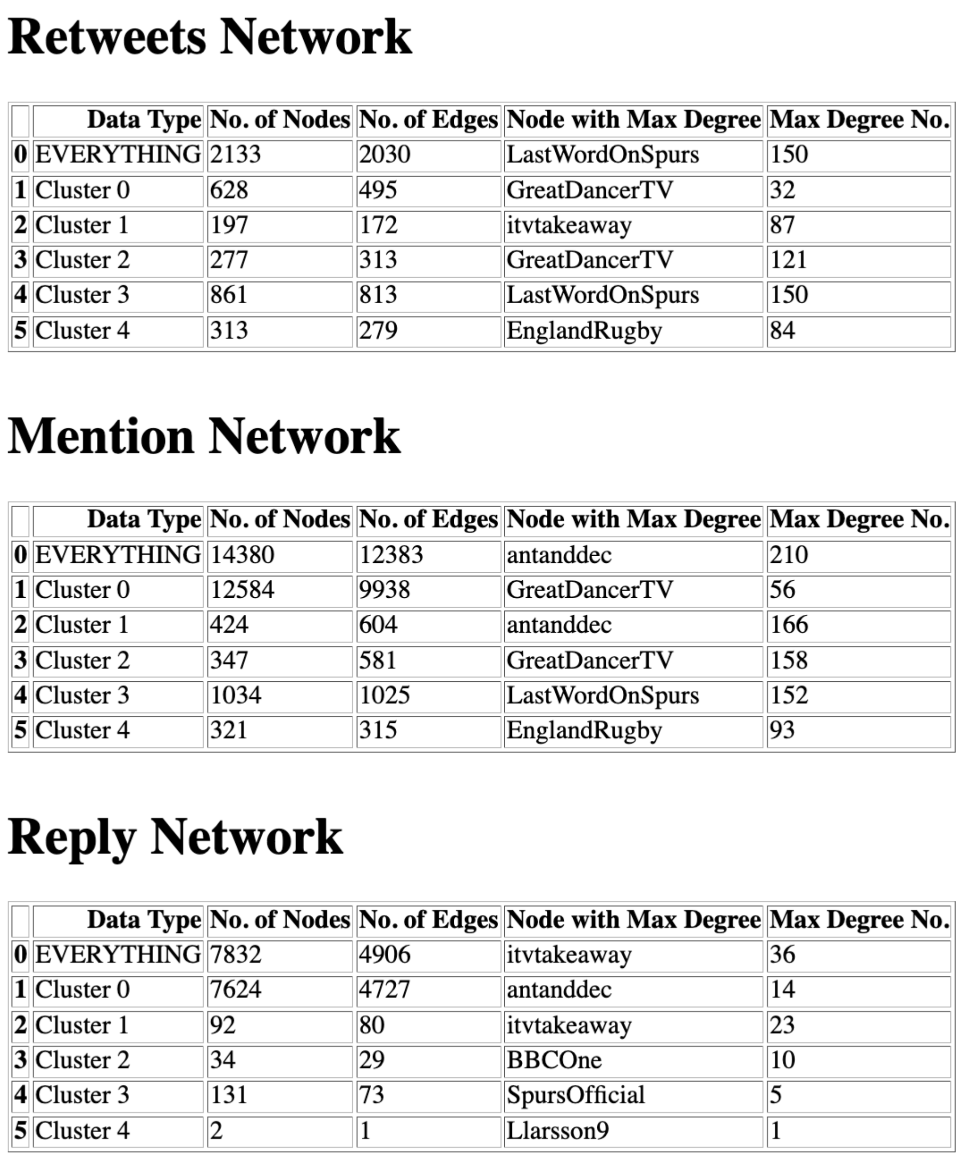
For every aforementioned network type, the following steps were undertaken in order to form a dictionary of user interactions and build a graph:

1. A Dataframe containing only the tweets was acquired by filtering from the original Dataframe.
2. For each tweet, the author was identified and according to the network type, the appropriate columns were searched by a for loop to gather all interactions with the author made in that tweet in a list and returned.
3. Each interaction in that list was then passed into the dictionary of users discussed above where the frequencies were also incremented if two users interacted more than once.
4. Using the `networkx` library, a directed graph was build by first instantiating using the command `nx.DiGraph()` and adding an edge from the author to the users he interacted with in the particular network like this: `mention_graph.add_edges_from([(auth_username, int_username)])`. An example graph was not constructed as due to the data being so much, and most users are unique it would be impossible to read.

These steps can all be found in the implemented functions named as follows:

- `get_retweet_network(dataframe)` - get the interactions between the different users who retweeted each others tweets
- `get_mentions_network(dataframe)` - get the interactions between the different users who mentioned each other
- `get_reply_network(dataframe)` - get the interactions between the different users who replied to each others tweets

In order to analyse the user interactions, for each network some important information were collected for each cluster and the original data. These were stored on tables in the *networkstats.html* and are shown in Figure 3.



Retweets Network

	Data Type	No. of Nodes	No. of Edges	Node with Max Degree	Max Degree No.
0	EVERYTHING	2133	2030	LastWordOnSpurs	150
1	Cluster 0	628	495	GreatDancerTV	32
2	Cluster 1	197	172	itvtakeaway	87
3	Cluster 2	277	313	GreatDancerTV	121
4	Cluster 3	861	813	LastWordOnSpurs	150
5	Cluster 4	313	279	EnglandRugby	84

Mention Network

	Data Type	No. of Nodes	No. of Edges	Node with Max Degree	Max Degree No.
0	EVERYTHING	14380	12383	antanddec	210
1	Cluster 0	12584	9938	GreatDancerTV	56
2	Cluster 1	424	604	antanddec	166
3	Cluster 2	347	581	GreatDancerTV	158
4	Cluster 3	1034	1025	LastWordOnSpurs	152
5	Cluster 4	321	315	EnglandRugby	93

Reply Network

	Data Type	No. of Nodes	No. of Edges	Node with Max Degree	Max Degree No.
0	EVERYTHING	7832	4906	itvtakeaway	36
1	Cluster 0	7624	4727	antanddec	14
2	Cluster 1	92	80	itvtakeaway	23
3	Cluster 2	34	29	BBCOne	10
4	Cluster 3	131	73	SpursOfficial	5
5	Cluster 4	2	1	Llarsson9	1

Figure 3: Statistics collected for User Networks

The following interpretations can be made looking at the statistics collected:

- Most users belong to the mention network.
- Nodes with max degrees usually are related to the trends collected.

4.2 Hashtag Information

Hashtags were captured and organised using the same steps as with users in a dictionary data structure by the function `get_hashtags_network(dataframe)`. For the hashtags appearing together in the same tweet all interactions were considered in every direction. In order to visualise this a bit better:

- If a tweet had three hashtags #SaturdayNightTakeaway #GreatestDancer #ENGvWAL then the dictionary created would be the following:

```
{
    'SaturdayNightTakeaway': {'GreatestDancer': 1, 'ENGvWAL': 1},
    'GreatestDancer': {'SaturdayNightTakeaway': 1, 'ENGvWAL': 1},
    'ENGvWAL': {'SaturdayNightTakeaway': 1, 'GreatestDancer': 1},
}
```

A directed graph has been created using the `networkx` library to represent the hashtag interactions that occur both in the whole dataset of tweets collected and the clusters. Adding direction to the graph even for hashtags was appropriate to use, as it allowed later on in the triad analysis to show a complete triad of hashtags occurring in one tweet together and also same with links, for two hashtags. All graphs for both clusters and the original data were plotted for the hashtag networks.

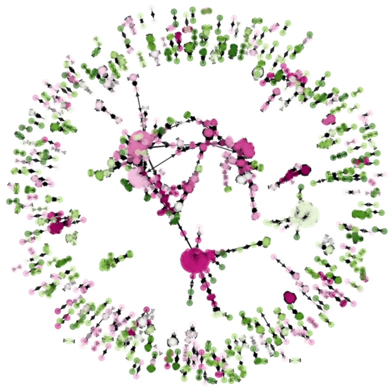


Figure 4: Hashtags Interaction Graph for the whole dataset

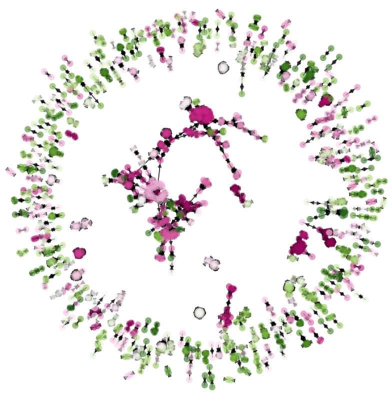


Figure 5: Hashtags Interaction Graph for Cluster 0

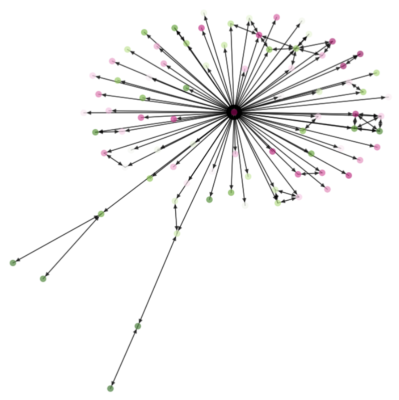


Figure 6: Hashtags Interaction Graph for Cluster 1

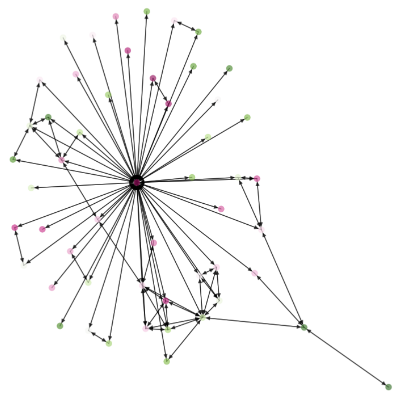


Figure 7: Hashtags Interaction Graph for Cluster 2

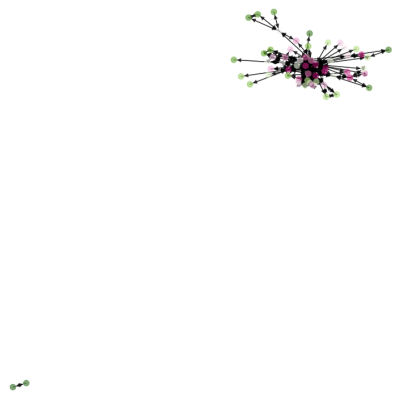


Figure 8: Hashtags Interaction Graph for Cluster 3

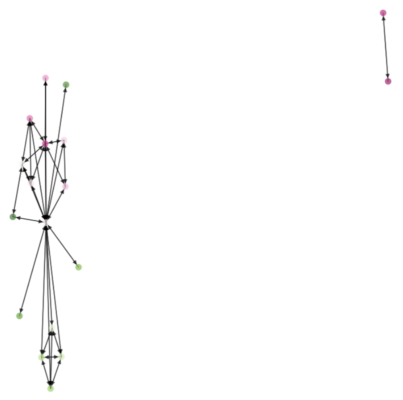
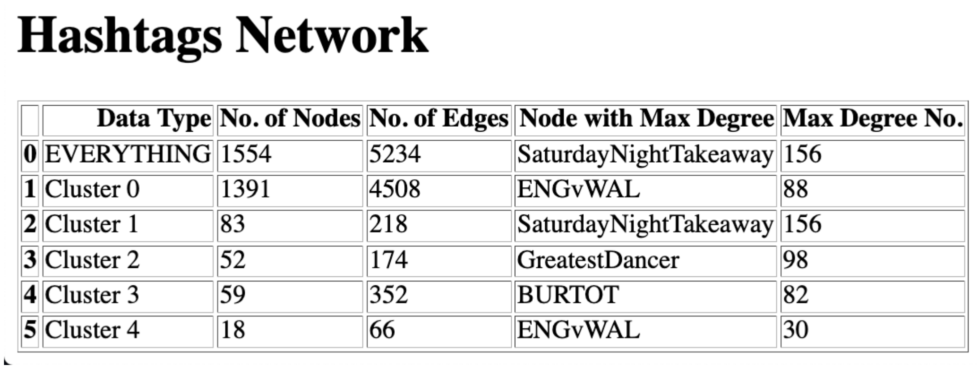


Figure 9: Hashtags Interaction Graph for Cluster 4

Finally similar analysis like with the users was also carried out for the hashtags networks shown in Figure 10 which has also been outputted in the *networkstats.html* file. We believe that since our clustering method was done based on content, the hashtag networks prove just that as different types of hashtags are on each network.



Hashtags Network

	Data Type	No. of Nodes	No. of Edges	Node with Max Degree	Max Degree No.
0	EVERYTHING	1554	5234	SaturdayNightTakeaway	156
1	Cluster 0	1391	4508	ENGvWAL	88
2	Cluster 1	83	218	SaturdayNightTakeaway	156
3	Cluster 2	52	174	GreatestDancer	98
4	Cluster 3	59	352	BURTOT	82
5	Cluster 4	18	66	ENGvWAL	30

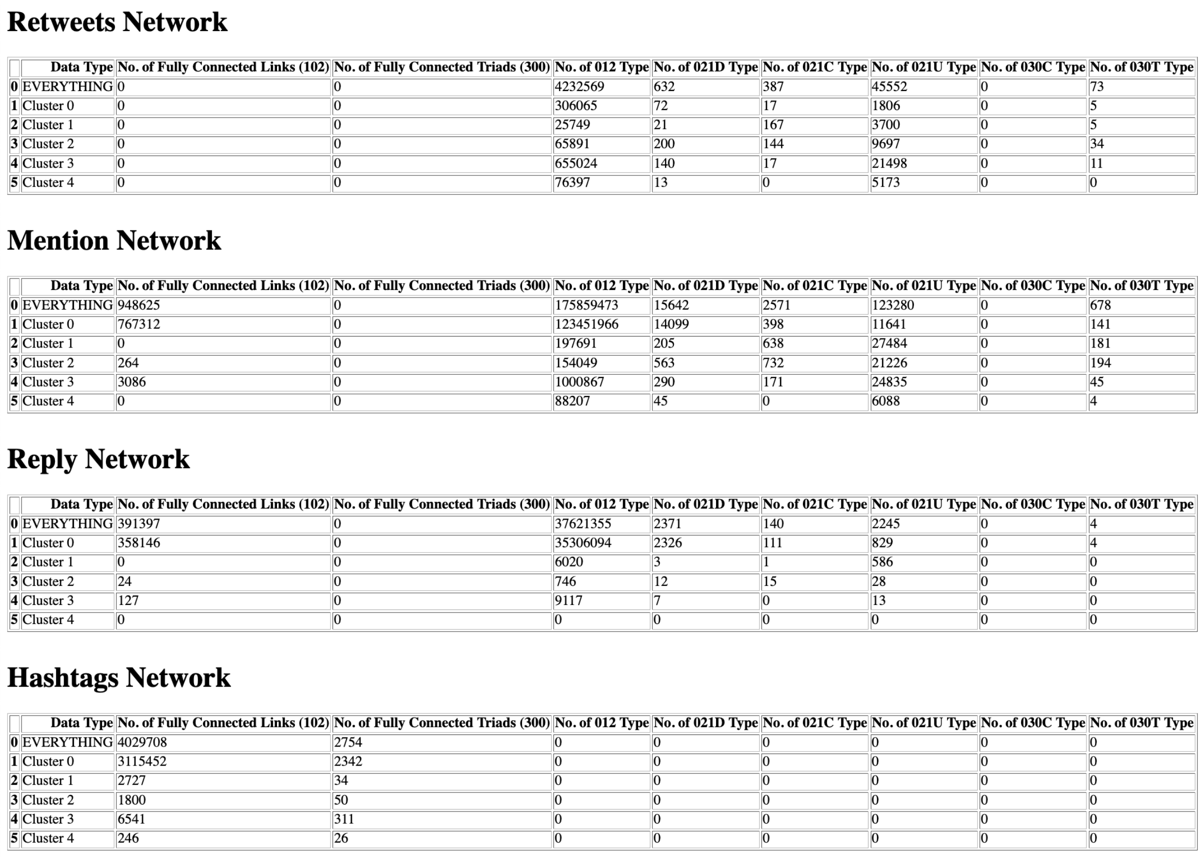
Figure 10: Statistics collected for Hashtags Networks

A few important points should be made by interpreting both the graphs and the statistics:

- It is shown that the content analysis done by K-means clustering weas appropriate as most hashtags are cluttered together representing communities.
- The nodes with Max Degrees for each clusters is noticed to be four out of the five trends collected, which is another indicator the clustering was done correctly. The fifth trend was not a hashtag hence why ENGvWAL which was the top trend at the time of collection was the node with the maximum degree for two clusters.

5 Network Analysis

By capturing and organising information for both hashtags and users in a dictionary whilst also creating the graphs at the same time, it was quite straightforward to carry out triad analysis on our data. With `networkx`'s `triadic_census(graph)` function[2], a count of how many of the 16 possible types of triads are present in a directed graph was collected in a dictionary. Furthermore, by using the returned dictionary, all data was visualised using a dataframe for each type of network (as shown in Figure 11 and outputted into the file **triads_links.html**.



Retweets Network

	Data Type	No. of Fully Connected Links (102)	No. of Fully Connected Triads (300)	No. of 012 Type	No. of 021D Type	No. of 021C Type	No. of 021U Type	No. of 030C Type	No. of 030T Type
0	EVERYTHING	0	0	4232569	632	387	45552	0	73
1	Cluster 0	0	0	306065	72	17	1806	0	5
2	Cluster 1	0	0	25749	21	167	3700	0	5
3	Cluster 2	0	0	65891	200	144	9697	0	34
4	Cluster 3	0	0	655024	140	17	21498	0	11
5	Cluster 4	0	0	76397	13	0	5173	0	0

Mention Network

	Data Type	No. of Fully Connected Links (102)	No. of Fully Connected Triads (300)	No. of 012 Type	No. of 021D Type	No. of 021C Type	No. of 021U Type	No. of 030C Type	No. of 030T Type
0	EVERYTHING	948625	0	175859473	15642	2571	123280	0	678
1	Cluster 0	767312	0	123451966	14099	398	11641	0	141
2	Cluster 1	0	0	197691	205	638	27484	0	181
3	Cluster 2	264	0	154049	563	732	21226	0	194
4	Cluster 3	3086	0	1000867	290	171	24835	0	45
5	Cluster 4	0	0	88207	45	0	6088	0	4

Reply Network

	Data Type	No. of Fully Connected Links (102)	No. of Fully Connected Triads (300)	No. of 012 Type	No. of 021D Type	No. of 021C Type	No. of 021U Type	No. of 030C Type	No. of 030T Type
0	EVERYTHING	391397	0	37621355	2371	140	2245	0	4
1	Cluster 0	358146	0	35306094	2326	111	829	0	4
2	Cluster 1	0	0	6020	3	1	586	0	0
3	Cluster 2	24	0	746	12	15	28	0	0
4	Cluster 3	127	0	9117	7	0	13	0	0
5	Cluster 4	0	0	0	0	0	0	0	0

Hashtags Network

	Data Type	No. of Fully Connected Links (102)	No. of Fully Connected Triads (300)	No. of 012 Type	No. of 021D Type	No. of 021C Type	No. of 021U Type	No. of 030C Type	No. of 030T Type
0	EVERYTHING	4029708	2754	0	0	0	0	0	0
1	Cluster 0	3115452	2342	0	0	0	0	0	0
2	Cluster 1	2727	34	0	0	0	0	0	0
3	Cluster 2	1800	50	0	0	0	0	0	0
4	Cluster 3	6541	311	0	0	0	0	0	0
5	Cluster 4	246	26	0	0	0	0	0	0

Figure 11: Triad and Links found for all networks

By reviewing the data obtained on the fully connected (flow in both directions) triads and links for each network as shown above, the following are observed:

- **There are no links or triads which are fully complete on the retweet network**: This makes total sense as usually people don't retweet non-popular tweets therefore popular accounts and it would unlikely for popular accounts to retweet smaller users posts which would give a two-way link or triad. If data was collected for longer there might have been a few occurrences but not many.
- **There were many fully connected links but no triads for the mention network**: When users mention each other in tweets, it is unlikely that they will mention both people that they interacted with in the previous tweet that they were mentioned on. However, again, if data was collected for longer there might have been a few occurrences but not many.
- **Similar patterns with the mention were observed for the reply network too**: Taking in account that the reply button replies to the author of the tweet, it would be very unlikely to get triads here.
- **On the hashtags network there were relatively quite a lot of both fully connected triads and links as well compared to the rest of the networks**: Hashtags which were presented together in a tweet were inputted in the dictionary to form a complete interaction relationship therefore such an observation is not surprising.

Overall, it is clear that not many fully connected triads were found in the user interaction networks, however, when testing throughout the implementation stages with more data, a few were found. Due to the requirement to run the data for around an hour, the following limitation is observed. In future works, I would run the crawlers for a longer time to collect more sample data of a days or weeks worth, which would provide much more insights.

References

[1] *TFIDFVectoriser*. 2020. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`.

[2] *Triadic Census - Networkx*. 2020. URL: `https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.triads.triadic_census.html`.

[3] *Tweepy*. 2020. URL: `https://www.tweepy.org/`.