University of L'Aquila
Faculty of Computer Science

Social Mining final Project 2020/21

Submitted by:

Leonardo Serilli 274426
Gabriele Colapelle 274560

Submission date: 20.07.2021

Contents

1 Temporal/Content Analysis

The dataset is focused on tweets that discuss the vaccine and the pandemics and is composed of 27M tweets collected in January/February 2021. Once loaded the ids of the Twitter stream S of the reference dataset, it is necessary to complete the dataset by downloading the original tweets of the dataset

1.1 Download original tweets of the dataset

To download the original tweets dataset given a set of files containing the tweets IDs, we have used the python library **tweepy** that expose a lot of usefull API's for our purpose. In particular the API **api.statuses_lookup** that returns full Tweet objects for up to 100 tweets per request, specified by the id parameter.
The task to collect the entire list of tweets is divided in three step implemented on the notebook **TweetRetriever.ipynb**

- ID's retrieve from ids.zip
- ID's filtering
- API call

1.1.1 ID's retrieve from ids.zip

```
def retrieve_ids():
    zip = zipfile.ZipFile('ids.zip')
    ids = []

    for fileName in zip.namelist():
        if(".txt" in fileName):
            file = zip.open(fileName)
            for line in file:
                curr_id = next(file, 1)
                if(curr_id!=1):
                    ids.append((curr_id.replace(b'\n', b'')).decode("utf-8"))
    return ids
```

The code above describe the method used to read the zip file that contains a set of .txt files, each txt file contains a set of tweet IDs. Each retrieved ID is appended to a list that contains the full set of IDs

1.1.2 ID's filtering

The code below describe the method used to filter the entire set of IDs taking at least the 10% of the entire set in an uniform way. The function has in *input* a list of IDs, the size of the chunk, the number of choices for each chunk and return a list of ID's. To obtain at least the 10% of tweets, we slide the full list of IDs trough windows of 15 IDs, and from these 15 we append 2 random IDs to the final list of filtered IDs.

```
def filter_ids(ids, chunk_size=15, num_choices=2):

    i = chunk_size
    chosen_ids = []
    while(i < len(ids)):

        chunk_ids = ids[i-chunk_size:i]

        for chosen_id in numpy.random.choice(chunk_ids, num_choices).tolist():
            chosen_ids.append(chosen_id)

        i += chunk_size;

        clear_output(wait=True)
        print("taken ids", len(chosen_ids))
    return(chosen_ids)
```

1.1.3 API call

The code below describe the method used to retrieve the tweet objects given their IDs, taking only visualizable tweets written in English language. Each retrieved tweet is written as a Json Object ia .gzip file.

```
def retrieve_tweets(ids):
    i = 100
    n_tweets=0

    while(i < len(ids)):

            tweets = api.statuses_lookup(id_ = ids[i-100:i])
```

```
        clear_output(wait=True)
        print("chunk", i)
        print("current retrived tweets", len(tweets))
        print("number of tweets already in dataset.gz ", n_tweets)

        for tweet in tweets:
            if(tweet.lang == 'en'):
                with gzip.open('dataset.gz', 'a') as f:
                    f.write(json.dumps(tweet._json).encode('utf-8'))
                    f.write(b'\n')
                n_tweets += 1
        i += 100
```

1.1.4 problems faced

- Minimize the retrieving time
- Minimize the usage of RAM space during the API-call

1.2 Twitter stream analysis and time window generation

1.2.1 WindowGenerator

We had to analyze the retrieved twitter stream S and by considering time windows W of size 10 and 5 days respectively (moves forward an ideal window by 5 days and 60h respectively) The package **timeSeriesRetrivier** contains two classes that implement creation of windows of the required size. The two classes work in the same way on different time frames:

- **WindowGeneratorByDay** : create windows of size 10 with slide of 5 days
- **WindowGeneratorByHour** : create windows of size 5 with slide of 60h

WindowGeneratorByDay Implements the **getSlidingWindows** method that has as:

- *input*:None
- *output*: ArrayList<ArrayList<Integer» **allWindows** containing all IDs of tweets in a window for every window

this method iterates on all the generated windows of size 10, and for each window it generates a range-query on the window timeframe, taking the IDs of the tweets returned by the query

```java
package Main.timeSeriesRetriever;

public class WindowGeneratorByDay {

   //creates all windows of lenght 'len' from the whole dataset with a given
      slide; the method return an array of document ids
   public static ArrayList<ArrayList<Integer>> getSlidingWindows1() throws
      IOException{

      ArrayList<ArrayList<Integer>> allWindows = new
         ArrayList<ArrayList<Integer>>();
      .
      .
      .
      }
      return allWindows;
   }
```

```
}
```

WindowGeneratorByHour Implements the **getSlidingWindows()** method that take as:

- *input*:None
- *output*: ArrayList<ArrayList<Integer> > **allWindows** containig all IDs of tweets in a window for every window

this method iterates through all the generated windows of size 5 ,and for each window it generates a range-query on the window timeframe, taking the IDs of the tweets returned by the query.

```java
package Main.timeSeriesRetriever;


public class WindowGeneratorByHours {

    // generate all windows of 120h with a slice of 60h, and for each one it
       returns the array of their indexes
    public static ArrayList<ArrayList<Integer>> getSlidingWindows() throws
        IOException{

        ArrayList<ArrayList<Integer>> allWindows = new
            ArrayList<ArrayList<Integer>>();
        }
        return allWindows;
    }
}
```

1.2.2 TF-IDF

For each window w in W select its top 5.000 tokens according to their TF-IDF (TF must be obtained by considering the window w, the IDF must be obtained considering the whole dataset)
To implement this task, we have created a Java Class **TokenRetrivier** in the package **Timeseries** that contain two method to retrieve these top token. the first method : **getTokensFromWindow**

- *input* : ArrayList<Integer> **window** cointaning the tweets ID's of a window
- *output* : HashMap<String, Float> **tokens** with the tokens contained in the window and their frequency

```
package Main.timeSeries;

public class TokenRetriever {

    private static HashMap<String, Float>
        getTokensFromWindow(ArrayList<Integer> window)
    throws IOException {
        .
        .
        }
        //Return the map <token,frequency>
        return tokens;
    }
```

The second method : **getTopWindowTokens**

- *input* : Arraylist<interger> **window** input : containing the tweets ID's of a window and an integer **num_token** that represent the number of token that we want to retrieve, in this case 5000.
- *output* : HashSet<String> **topTokens** containing the top 5000 token according to the TF-IDF score

```

public static HashSet<String> getTopWindowTokens(ArrayList<Integer> window, int
    num_token) throws IOException{

        //call at the method tokenRetriever
        HashMap<String, Float> TFIDFallTokens =
            TokenRetriever.getTokensFromWindow(window);
        .
        .
        .
        return topTokens;
    }
```

1.2.3 Time-series

For each t in T^w built their time-series with a grain of 24h.
the Class **Timeseries_generator** in the **timeSeriesRetriever** package, implements the method **generateTS()** that deals with the generation of times series on a single token. The method parameters are:

- *input* :
  - String **token**, token on which we want to generate the time serie
  - ArrayList<Integer> **window** Arraylist containing IDs of tweets of a window
  - Long **start_mills** staring date of the window in milliseconds
  - Long **end_mills** ending date of the window in milliseconds
  - String **dirPath** path of directory where are stored all windows files
- *output* :write into a Json Object the pair $(token, json\_ts)$, json_ts contains the pair $(day + moth, frequency)$

```
package Main.timeSeriesRetriever;

public class Timeseries_generator {

    public static void generateTSbyDay(String token, ArrayList<Integer> window,
        Long start_mills, Long end_mills, String dirPath) throws IOException {
    .
    .
    .
    }
}
```

Main Class Now is defined the **main** Class to execute the methods defined up to now, in order to have a well-defined workflow.

```
import Main.onHandUtilities;


public class main {

    public static void main(String[] args) throws IOException {
```

```java
// ---------- Timeseries Retrieving for windows of lenght 10days with a
    sliding factor of 5 days ----------
ArrayList<ArrayList<Integer>> windows_byDay =
    WindowGeneratorByDay.getSlidingWindows1(); // retrive all ids of
    tweets for every window

long start_mills = onHandUtilities.startMillis();
long end_mills = start_mills + TimeUnit.DAYS.toMillis(9);

// iterate over windows
int i = 1;
for (ArrayList<Integer> win : windows_byDay) {
    if(win.size()!=0) {
        // retrieve top 5000 tokens based of tf-idf score from current
            window
        HashSet<String> tokens = TokenRetriever.getTopWindowTokens(win,
            5000);

        // write to file the timeseries for each token
        for (String token: tokens){
            Timeseries_generator.generateTSbyDay(token, win, start_mills,
                end_mills, "src/main/java/Main/timeseries_windowsByDay" );
        }
        i++;
    }
    // set the time interval for next window
    start_mills = start_mills + TimeUnit.DAYS.toMillis(5);
    end_mills = start_mills + TimeUnit.DAYS.toMillis(9);
}
// ---------- Timeseries Retrieving for windows of lenght 120 hours with
    a sliding factor of 60 hours ----------
ArrayList<ArrayList<Integer>> windows_byHour =
    WindowGeneratorByHours.getSlidingWindows();

start_mills = onHandUtilities.startMillis();
end_mills = start_mills + TimeUnit.DAYS.toMillis(5);
// iterate over windows
i = 1;
for (ArrayList<Integer> win : windows_byHour) {
    if(win.size()!=0) {
            // retrieve top 5000 tokens based of tf-idf score from current
               window
            HashSet<String> tokens = TokenRetriever.getTopWindowTokens(win,
               5000);

            // write to file the timeseries for each token
            for (String token: tokens){
               Timeseries_generator.generateTSbyDay(token, win, start_mills,
                  end_mills, "src/main/java/Main/timeseries_windowsByDay" );
            }
            i++;
         }
         // set the time interval for next window
         start_mills = start_mills + TimeUnit.HOURS.toMillis(60);
         end_mills = start_mills + TimeUnit.DAYS.toMillis(5);
      }
```

1.2.4 Sax

for each time serie generated in the last task , use the sax representation to filter them out by keeping those that expose the typical pattern of the collective attention.
In the package **Sax** are written three classes:

- **SaxBuilder** and **SaxWriter** are the two classes that implement the SAX representation
- **PatternOfCollectiveAttention_finder** to keep only the tokens that expose the typical pattern of the collective attention
- **main** Run this code to call all method described above

SaxBuilder contains all utilities method to perform the Sax representation

```
package Main.sax;


public final class Sax_builder {

   private final TSProcessor tsProcessor;
   private final NormalAlphabet na;
   private EuclideanDistance ed;
```

```
    //Constructor
    public Sax_builder() {

    }

    //Build a Sax from timeserie given as an array of double
    public SAXRecords ts2saxByChunking(double[] ts, int paaSize){

    }

    //get the ordered time serie from a json time serie
    public static double[] getOrderedTS(JSONObject ts_json){
    }
}
```

SaxWriter Contains the method **writeSaxToFIle()**

- *input*: path where are stored the timeseries
- *output*: write SAX representation of timeseries as a JsonObject in a .gzip file

```
package Main.sax;


public class Sax_writer {

// call the method ts2saxByChunking then write to file the result timeseries
    private static void writeSaxToFIle(String windowsDirpath) throws
        IOException {
    .
    .
    }

    public static void main(String args[]) throws IOException {

        writeSaxToFIle("src/main/java/Main/timeseries_windowsByDay");
        writeSaxToFIle("src/main/java/Main/timeseries_windowsByHour");

    }

    }
```

PatternOfCollectiveAttention_finder Contain the method **pattern_matcher()** defined as follow:

- *input* :
  - String **windowsDirpath** the directory where are stored the Sax representations
  - String **regex** the regular expression that we want to match
- *output* : write the tokens that match the regular expression into a JsonObject

```java
package Main.sax;

public class PatternOfCollectiveAttention_finder {

    private static void pattern_matcher(String windowsDirpath, String
        regex) throws IOException {
    .
    .
    }

}
```

Main this code write on file the json $\{"Token" :< TOKEN >, "sax" :< SAX >\}$ for each timeserie, associated with the token, in the file $" < WINDOW > /Timeseries.gzip"$, and write to the file $" < WINDOW > /collectiveAttention\_tokens\_alpha2.gzip"$ the json $\{"Token" :< TOKEN >, "sax" :< SAX >\}$ for each token which timeserie matches the pattern

```java
package Main.saxGenerator;

import java.io.IOException;

public class main {

    public static void main(String args[]) throws IOException {
        // write to file "<WINDOW>/saxRepresentations_alpha_2.gzip" the json
            {"Token": <TOKEN>, "sax": <SAX>} for each
        // timeserie, associated with the token, in the file
            "<WINDOW>/Timeseries.gzip"
        Sax_writer.writeSaxToFIle("src/main/java/Main/timeseries_windowsByDay");
        Sax_writer.writeSaxToFIle("src/main/java/Main/timeseries_windowsByHour");
        // write to the file "<WINDOW>/collectiveAttention_tokens_alpha2.gzip"
            the json {"Token": <TOKEN>, "sax": <SAX>} for each
        // token which timeserie match the pattern
        PatternOfCollectiveAttention_finder.matching_pattern_writer("src/main/java/Main/
        timeseries_windowsByDay", "a+b?bb?a+?a+b?bba*?");
        PatternOfCollectiveAttention_finder.matching_pattern_writer("src/main/java/Main/
        timeseries_windowsByHour", "a+b?bb?a+?a+b?bba*?");

    }
}
```

1.2.5 Computing the regular expression with RPNI algorithm

In addition to the regex provided by the professor ("a+b?bb?a+?a+b?bba*?) during the lesson, we have calculated another one using the RPNI algorithm.
We have selected about 20 positive words (words strongly related to the pandemic) and about 20 negative words (top 20 common words).
through the Classes **RPNI_Negative_writer** and **RPNI_Positive_writer** we can obtains the SAX representation of positive and negative words.

RPNI_Positive_writer generates and save to file Sax representation of positive words given the path where are stored these words

```
package Main.sax;

public class RPNI_Positive_writer {

    // generate and save to file Sax representation of positive words given the
        path where are stored these words

    public static void generateTopWordSaxFiles(String path) throws IOException {
    .
    .
    }


    public static void main (String args[]) throws IOException {
        String path = "src/main/java/Main/timeseries_windowsByDay";
        generateTopWordSaxFiles(path);
        path = "src/main/java/Main/timeseries_windowsByHour";
        generateTopWordSaxFiles(path);
    }
}
```

RPNI_Negative_writer generates and save to file Sax representation of negative words given the path where are stored these words

```
 package Main.sax;

public class RPNI_Negative_writer {

// generate and save to file Sax representation of negative words given the
    path where are stored these words
    public static void generateTopWordSaxFiles(String path) throws IOException {
        .
        .
    }

    public static void main (String args[]) throws IOException {

        String path = "src/main/java/Main/timeseries_windowsByDay";
        generateTopWordSaxFiles(path);

        path = "src/main/java/Main/timeseries_windowsByHour";
        generateTopWordSaxFiles(path);
    }

}
```

NB In the same package is present the class **RPNI_POC_Writer** that implements the method (not used) to match the regular expression found trough RPNI algorithm instahead of the string used in **PatternOfCollectiveAttention_finder**

Find compatible regular expression

Once we have the SAX representation of the positive and negative words we can find a compatible regular expression through the RPNI algorithm, to complete this task we have used a Python library **Inferrer** that offers the python script **cly.py** to calculate the final regular expression.

```python
import os
import sys

## Generate regular expression from a set of positive words and a set of
    negative words
def generateRegEx(curr_dir):
    for subdir, dirs, files in os.walk(curr_dir):
        for dir in dirs:
            tmp_currDir = curr_dir + "/" + dir
            positive = tmp_currDir + "/topWordsSax.txt"
            negative = tmp_currDir + "/topEnglishCommonWordsSax.txt"

            clear_redoundancy(positive, negative)

            str = "Generating regex for window: "
            print(str + dir)
            os.system("pipenv run python cli.py " + positive + " " + negative +
                " rpni" + " > " + tmp_currDir + "/RPNI_regex.txt" )

##clear the Sax Strings that occour either in negative set and positive set.

def clear_redoundancy(positive, negative):
    print("clearing redoundancy")
    try:
        fpos = open(positive)
        fneg = open(negative)

        pos_lines = fpos.readlines()
        neg_lines = fneg.readlines()

        c = 0;
        with open(negative, "w") as f:
            for neg in neg_lines:
                if((neg not in pos_lines) and (c<15)):
                    f.write(neg)
                    c += 1
    finally:
        fpos.close()
        fneg.close()
```

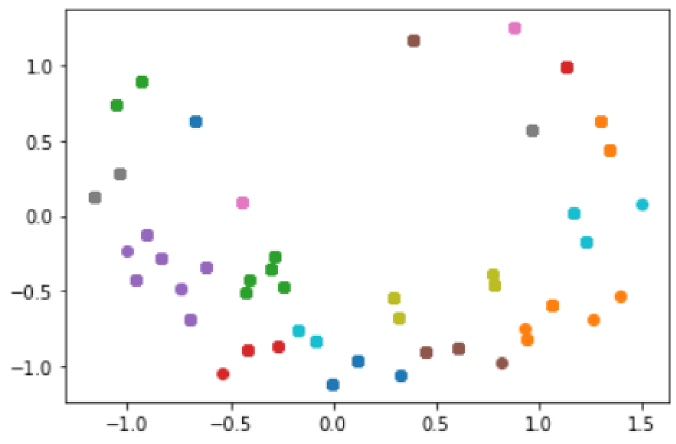
1.2.6 Clustering

With the K-Means algorithm, with k = $|T^w|/20$ group together all the tokens that expose an equal temporal behaviour (same or very similar SAX string).
Thus obtaining Clusters of Terms C^w for everytime window w in W.
In order to complete the task we have created a python notebook, the code is illustrated below.

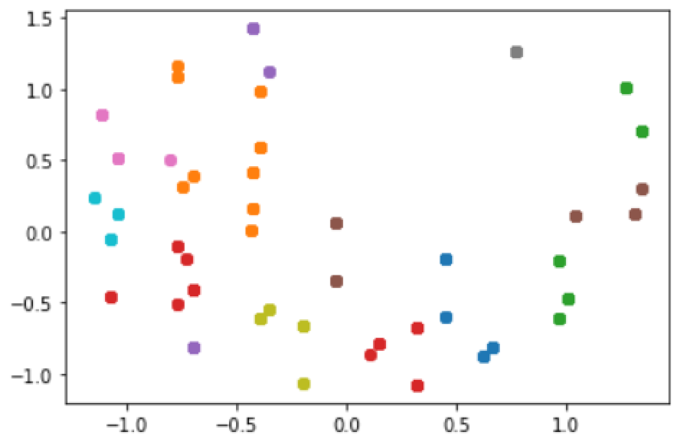
```
## Get the dataframe of SAX string that expose the typical pattern of the
    collective attention from the Array of SAX string.
## Each token is splitted letter by letter and is represented by a row of the
    dataframe

def getDF(sax_Arr):
    .
    .
    return df

## write to a file tokens that belong to the same cluster, these tokens are
    written on the same line
def writeClusters(collective_attention_df, clustersFilename, nClusters):


## From the dataframe containing the SAX string and an integer nclusters run
    the K-means algorithm and plot the result

def clusterer(df, nClusters):
    pca = PCA(2)
    kmeans = KMeans(n_clusters = nClusters)
    cluster_map = pd.DataFrame()

    df = pca.fit_transform(df)
    label = kmeans.fit_predict(df)
    filtered_label0 = df[label == 0]

    cluster_map['cluster'] = kmeans.labels_

    u_labels = np.unique(label)

    #plotting the results:
    for i in u_labels:
        plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
                    pyplot.legend(bbox_to_anchor=(1.1, 1.05))
        pyplot.show()
    plt.show()

    return cluster_map

## main program to run the K-means Algorith

for subdir, dirs, files in os.walk(curr_dir):
        for dir in dirs:
            collAtt_json = curr_dir + "/" + dir +
                "/collectiveAttention_tokens_alphabetSize2.gzip"
            sax_Arr = []
            tokens_Arr = []
            if(os.path.isfile(collAtt_json)):
                print("\nWindow: " + dir)
                with gzip.open(collAtt_json, 'r') as f:
                     for line in f:
                        line = json.loads(line)
                        sax_Arr.append(line["sax"])
                        tokens_Arr.append(line["token"])

                df = getDF(sax_Arr)

                df_tokens = pd.DataFrame(np.array(tokens_Arr))
                df_sax = pd.DataFrame(np.array(sax_Arr))

                nClusters = len(tokens_Arr)//20
                clusters = clusterer(df, nClusters)

                # code to write clusters to file
                collective_attention_df = pd.DataFrame()
                collective_attention_df["token"] = df_tokens[0]
                collective_attention_df["cluster"] = clusters["cluster"]

                clustersFilename = curr_dir + "/" + dir + "/clusters.gzip"

                writeClusters(collective_attention_df, clustersFilename,
                    nClusters)
```

Results Plots

Window: JSONtimeseries_21Jan_30Jan



Window: JSONtimeseries_26Jan_4Feb



1.2.7 problems faced

- few Lucene documentation
- How generate time windows and timeseries in a efficient way

1.3 Co-occurence Graph

For each time window w and each cluster c in C^w considering every token t in c build the co-occurrence graph of them (two word t_1 ,t_2 have an edge e if the tokens appear in the same tweet; the weight $w(e)$ of e is equal to the number of documents where both tokens appear)
In the package **clusters** are implemented two classes: **coOccurenceGraph_generator** and **WeightedGraph**.

1.3.1 coOccurenceGraph_generator

Implements the **coOcc_builder** method that write on file a graph for each cluster in every window

- *input*: String **windowsDirpath**, path where are stored the windows
- *output*: : write to file the generated graph for each cluster in every window

```
package Main.clusters;


public class coOccurenceGraph_generator {

   // build the co-occurence graph of all clusters of all windows and write
      them into a file
   public static void coOcc_builder(String windowsDirpath) throws IOException,
      ParseException {
   .
   .
   }
   public static void main(String[] args) throws IOException, ParseException {

      coOcc_builder("src/main/java/Main/timeseries_windowsByDay");
      //coOcc_builder("src/main/java/Main/timeseries_windowsByHour");
   }

}
```

WeightedGraph Class implement a Graph with nodes and edges in Json format.

```java
package Main.clusters;

public class WeightedGraph {

    static class Edge {
        String source;
        String destination;
        int weight;

        public String getSource() {
            return source;
        }
        public String getDestination() {
            return destination;
        }
        public int getWeight() {
            return weight;
        }
        public Edge(String source, String destination, int weight) {
            this.source = source;
            this.destination = destination;
            this.weight = weight;
        }
    }

    private ArrayList<String> nodes;
    private ArrayList<Edge> edges;

    public ArrayList<String> getNodes() {
        return nodes;
    }
    public ArrayList<Edge> getEdges() {
        return edges;
    }

    public WeightedGraph(){
        this.nodes = new ArrayList<String>();
        this.edges = new ArrayList<Edge>();
    }
    public void addEgde(String source, String destination, int weight) {
        Edge edge = new Edge(source, destination, weight);
        this.edges.add(edge);
    }

    public void addNode(String node) {
        this.nodes.add(node);
    }

    public void writeGraphToFile(String filePath){
        FilesUtilities.writeToFile(jsonGraph, filePath);
    }


}
```

1.3.2 Connected components and K-Core

For each cluster in every window Identify the Connected Components **CC** and extract the innermost core (K-Core). Thus, from each cluster and approach (CC and K-Core) now identify subgroups of tokens. These functions are implemented in the notebook **clustersAnalizer.ipynb** where are defined the following functions:

- **graph_from_zip(file)**: read the graph from a Json file into a nx.Grpah
- **draw(graph)** : draw the weighted graph
- **largestCC(G)** : find the largest connected component of a graph
- **get_K(K_core_G)** : return the value of k (min number of edges in the innermost core graph)
- **innermostCore(G)** : return the innermost core from G
- **getBest5(graphs)** : return the top 5 clusters based on the value of k

The main script takes iteratevly each cluster, build the graph from the given cluster and calculate the following subgraphs:

- largest connected component
- innermost core

```
curr_dir = "../socialMining/src/main/java/Main/timeseries_windowsByDay"
#curr_dir = "socialMining/src/main/java/Main/timeseries_windowsByHour"

for subdir, dirs, files in os.walk(curr_dir):
       for dir in dirs:
           clusters_dir = curr_dir + "/" + dir + "/clustersGraphs"
           if os.path.isdir(clusters_dir):
               print(clusters_dir)
               for filename in os.listdir(clusters_dir):
                   G = graph_from_zip(clusters_dir + "/" +filename)
                   draw(G)

                   largest_cc = largestCC(G)
                   draw(largest_cc)

                   inn_core = innermostCore(G)
                   draw(inn_core)
                   break
           print()
           break
```

script output

for each Graph defined on each cluster the output of the script will contain:

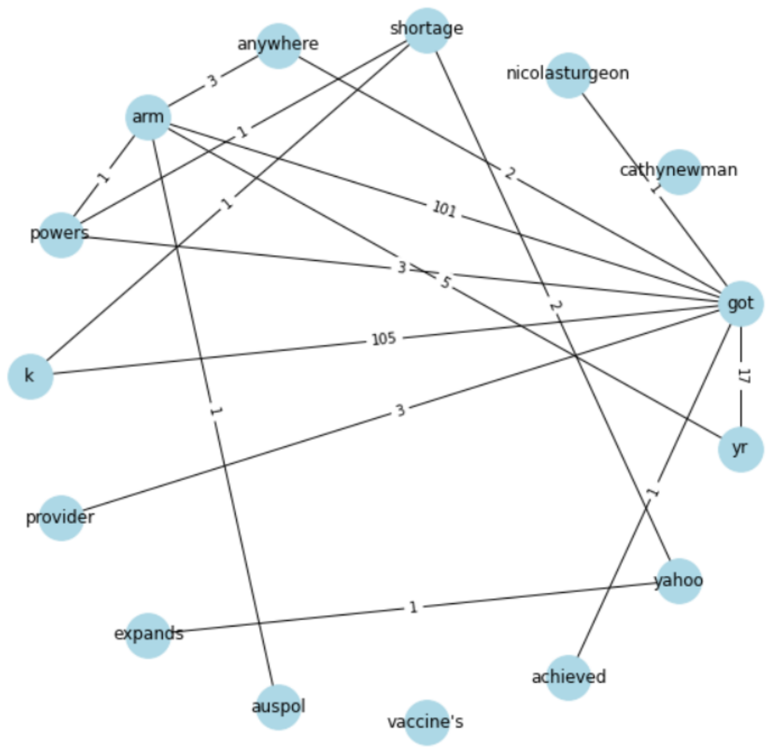


Figure 1: Initial graph

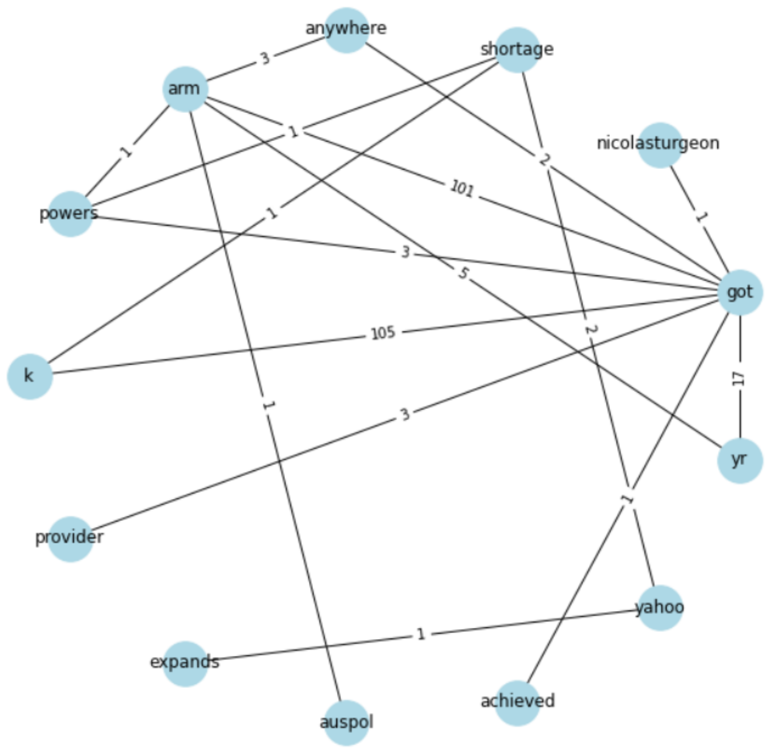


Figure 2: Largest connected component

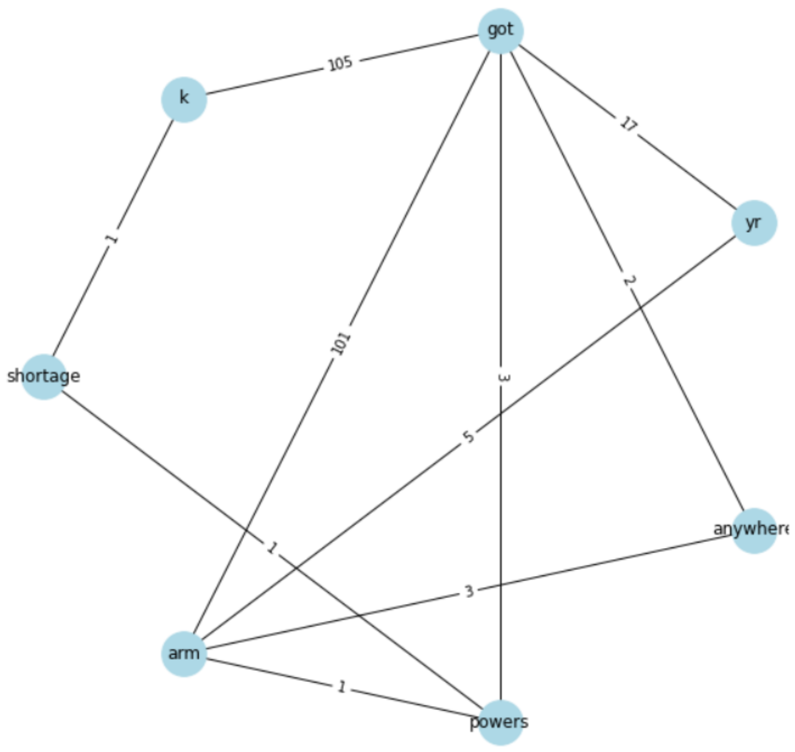


Figure 3: Innermost Core

Original Graph vs Largest connected component vs Innermost Core

- **Original graph**: many insignificant arcs and nodes, but contain all of information about the cluster
- **Largest connected component** : provide more information with respect to the original graph, in particular we can find tokens that could have a sort o correlation
- **Innermost core** : identify a community of terms that could occurs together with high frequency

in this case we have not set a threshold on the weight of the arches, but we have kept them all, increasing the threshold we will keep only the most correlated terms but we will lose some information

1.4 Top 5 clusters

for the top 5 obtained group of tokens (Clusters identified in the previous task) plot their timeseries (by using a temporal grain of 12h or/and 24h); compare their time series and comments about some possible kind of action-reaction that should be clearly identified. Below is illustrated the python script used to retrieve and plot the timeserie of the best 5 clusters of each window based on how bigger it's innermost-core is in terms of K.

```
for graphs in windows:

    best5 = getBest5(graphs)
    describe_graphs(best5)

    #print("\n" + '-'*50 + graphs[0]["window"] + '-'*50 + "\n")
    for cluster in best5:
        print("\n\t" + cluster["cluster"])
        for token in cluster["k-core"].nodes():
            allTS = getAllTS(best5[0]["window"])
            ts = getToken(token, allTS, graphs[0]["window"])
            pyplot.plot(ts, label=token)
            pyplot.legend(bbox_to_anchor=(1.1, 1.05))
        pyplot.show()
```

Timeseries plots Expose some significant timeseries

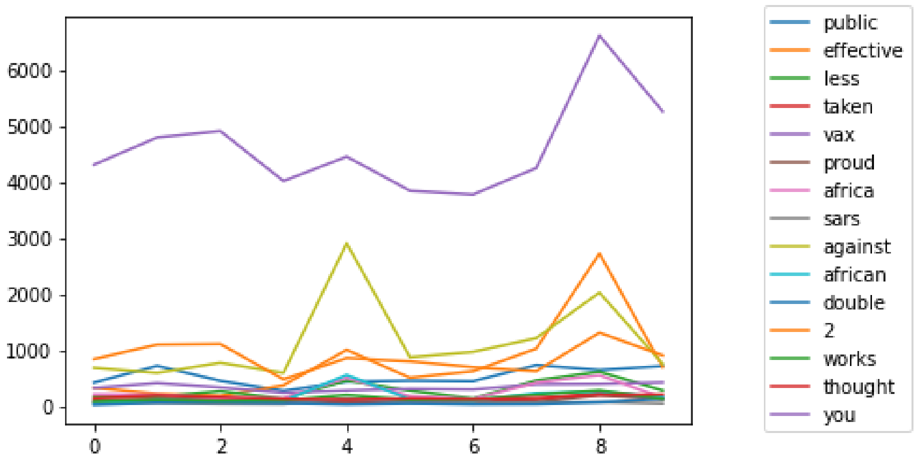


Figure 4: The plots is made on the window from the 21 of January to the 30 of January, represent the first cluster of the best five obtained. We can see that the most frequent term is **vax** with the higher value on the 29 of January.
Doing some research we have seen that on the 29 of January EMA recommends COVID-19 Vaccine AstraZeneca for authorisation in the EU, this could be the reason why the term has a very high frequency.

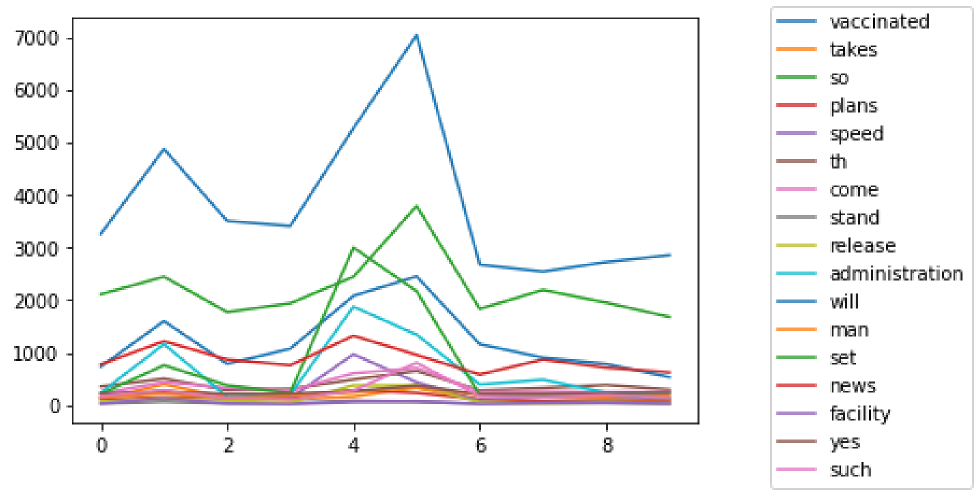


Figure 5: One of the best 5 obtained cluster from the window 11-20 January, is easy to see that the most frequent token is **vaccinated**, but are present a lot of other terms like **plans**, **speed**, **release**, **administration** which suggest a possible vaccination campaign. In fact on the 16th of January (day with the higher value for these tokens) began in India on the largest vaccination programme

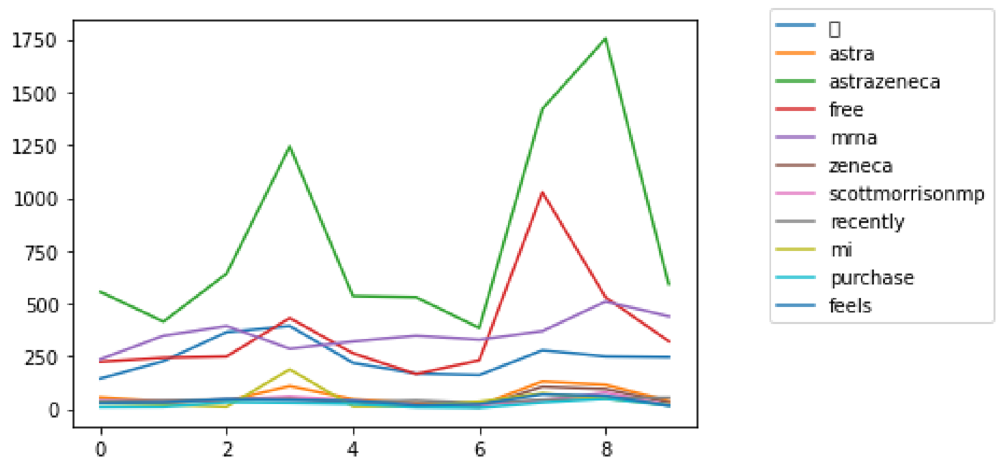


Figure 6: One of the best 5 obtained cluster from the window 31Jan-9Fab, the most frequent token is **astrazeneca** with the higher value on 8 February, because The World Health Organization's vaccine advisory panel has scheduled a review of the Astrazeneca COVID-19 vaccine for February 8

2 Network Analysis

2.1 Mention graph

Considering the initial twitter dataset, identify tweets of users that mention other users, and/or retweets other user's tweets and build a weighted and directed graph G^m where the weight of the edges represent the number of mentions or retweets identified. (on top 1000 users)

to implement this task, in the package **networkAnalysis** is implemented the **users_Gm_retrivier** Class that contains two methods: **getUsersIDs()** and **writeUsers_Gm_g**

getUsersIDs() has as :

- *input* : Long **start_mills** starting date of the window, Long **end_mills** ending date of the window
- *output* : ArrayList<String> **users_ids** that contains IDs of users that have posted a tweet in the current window

writeUsers_Gm_graphs() takes as:

- *input* : String **windowsDirpath** the directory where are stored the windows
- *output* : write the Graph into the widow directory

```java
package Main.networkAnalysis;

public class users_Gm_retriever {

    // return users ids of the given window
    public static ArrayList<String> getUsersIDs(Long start_mills , Long
        end_mills) throws IOException {

        ArrayList<String> users_ids = new ArrayList<String>();
        .
        .
        .
        }
        return users_ids;
    }

    // write to the all Gm graphs to different file inside the directory
        "<WINDOW_DIR>/User_clusterGraphs/<.gzip_file>"

    public static void writeUsers_Gm_graphs(String windowsDirpath) throws
        IOException {
        File window_files = new File(windowsDirpath);
        .
        .
        .
    }

    public static void main(String[] args) throws IOException, ParseException {
        writeUsers_Gm_graphs("src/main/java/Main/timeseries_windowsByDay");
    }
}
```

2.2 Posted tweets graph

Define the weighted and undirected graph G^t with users on vertices based on the identified clusters in precendece. Two users have an edge if they both had posted a tweet (in one of the identified time window w) which contains one of the tokens t in $T^{w'}$ of the cluster c in C^w . The weight $w(e)$ of the edge represents the number of times the two users are interested in the same topic. the **users_Gt_retriver** Class contains two methods that implement the generation of graph G^T: **getUsersIDsFromToken()** and **writeUsers_Gt_graphs()**

getUsersIDsFromToken() has as:

- *input*:Long **start_mills** starting date of the window, Long **end_mills** ending date of the window **String token**
- *output*: ArrayList<String> **users_ids** contains the IDs of users that have posted a tweet containing the given token in the current window

writeUsers_Gt_graphs() has as:

- *input* : String **windowsDirpath** the directory where are stored the windows
- *output*: write the all G^m graphs to different file inside the clusters directory " $< WINDOW\_DIR > /User\_clusterGraphs/ < .gzip\_file >$ "

This method first retrieves tweets containing a given token for each token in every cluster. From these tweets calculates the IDs of users and iterating on users ID's pair by pair we build the weighted graph. In this way we find group of users that have posted a tweet containing the same token for each cluster, All this group will be merged by window to build the final graph

```
 package Main.networkAnalysis;

public class users_Gt_retriever {

    // return all user mention a given token in a window

    public static ArrayList<String> getUsersIDsFromToken(Long start_mills ,
        Long end_mills, String token) throws IOException {
```

```
        ArrayList<String> users_ids = new ArrayList<String>();
        .
        .
        .
        }

        return users_ids;
    }

    // write to the all Gm graphs to different file inside the directory
        "<WINDOW_DIR>/User_clusterGraphs/<.gzip_file>"

    public static void writeUsers_Gm_graphs(String windowsDirpath) throws
        IOException {
        .
        .
        .

    }

    public static void main(String[] args) throws IOException, ParseException {
        writeUsers_Gm_graphs("src/main/java/Main/timeseries_windowsByDay");
    }
}
```

2.3 Graph analysis

Find the largest connected component CC on G^t And G^m and compute

- HITS
- LPA

2.3.1 G^m Analysis

We have implemented this task in the notebook **NetworkAnalysis_Gm.ipynb**, at first we read the G^m graph of the window and we compute largest connected components. On the generating graph we apply the HITS algorithm and the label propagation algorithm identifying different kind of community.

Largest connected component

```python
##read the graph from .zip file
def graph_from_zip(file):
    return G

# Drawing Function
def draw(graph, node_color='lightblue', edge_color='black'):


## Describe the Graph
def describe(original, largest_cc):
    original_shape = " orginal_shape: (" + str(len(original.nodes)) + ',' +
        str(len(original.edges)) + ')'
    largest_cc_shape = " cc_shape: (" + str(len(largest_cc.nodes)) + ',' +
        str(len(largest_cc.edges)) + ')'
    print(original_shape + ", " + largest_cc_shape)


# Generate the largest connected component of all windows (in this case we will
    show it just for the window "21Jan_30Jan")
windows = []
curr_dir = "../socialMining/src/main/java/Main/timeseries_windowsByDay"
for subdir, dirs, files in os.walk(curr_dir):
    for dir in dirs:
        filename = curr_dir + "/" + dir + "/Users_Gm_Graphs/Gm_graph.gzip"
        if(os.path.isfile(filename)):
            graphs = []
            G = graph_from_zip(filename)
            graphs.append(G)

G = graphs[0]
largest_cc = largestCC(G)
describe(G, largest_cc)

draw(largest_cc)
```

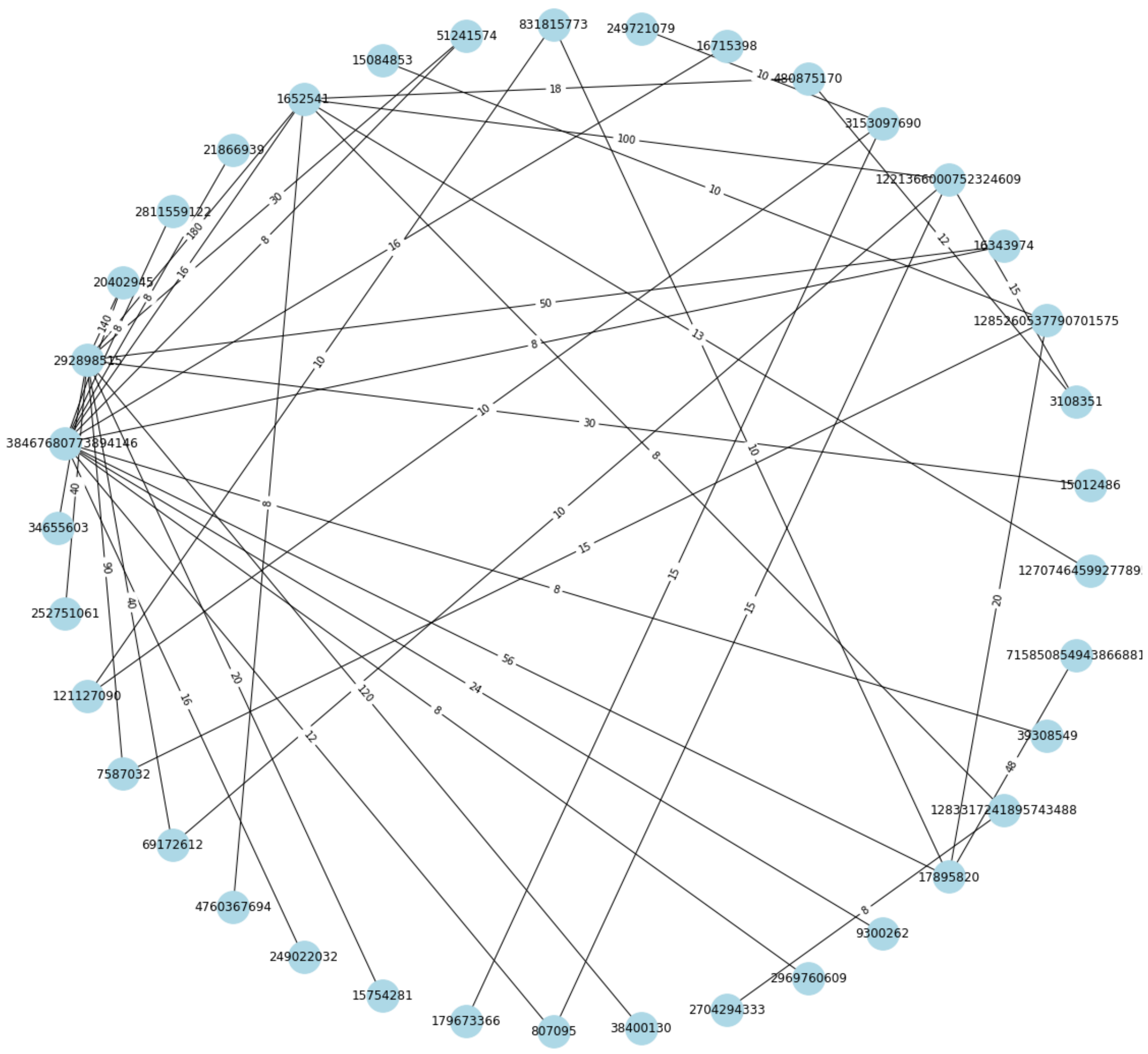


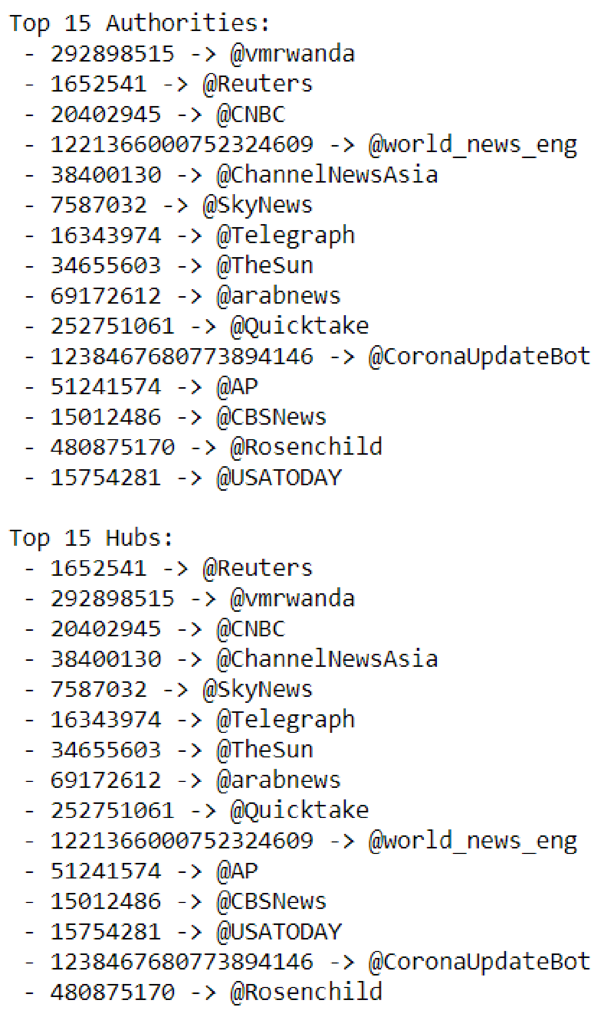
Figure 7: Largest connected component for the window "21Jan_30Jan"

HITS algorithm

Running this algorithm on the identified largest connected component we print out the id and name of the top 15 authority users and the top 15 hub users,

```
h, a = nx.hits(largest_cc)
```

Output of HITS algorithm



```
Top 15 Authorities:
 - 292898515 -> @vmrwanda
 - 1652541 -> @Reuters
 - 20402945 -> @CNBC
 - 1221366000752324609 -> @world_news_eng
 - 38400130 -> @ChannelNewsAsia
 - 7587032 -> @SkyNews
 - 16343974 -> @Telegraph
 - 34655603 -> @TheSun
 - 69172612 -> @arabnews
 - 252751061 -> @Quicktake
 - 1238467680773894146 -> @CoronaUpdateBot
 - 51241574 -> @AP
 - 15012486 -> @CBSNews
 - 480875170 -> @Rosenchild
 - 15754281 -> @USATODAY

Top 15 Hubs:
 - 1652541 -> @Reuters
 - 292898515 -> @vmrwanda
 - 20402945 -> @CNBC
 - 38400130 -> @ChannelNewsAsia
 - 7587032 -> @SkyNews
 - 16343974 -> @Telegraph
 - 34655603 -> @TheSun
 - 69172612 -> @arabnews
 - 252751061 -> @Quicktake
 - 1221366000752324609 -> @world_news_eng
 - 51241574 -> @AP
 - 15012486 -> @CBSNews
 - 15754281 -> @USATODAY
 - 1238467680773894146 -> @CoronaUpdateBot
 - 480875170 -> @Rosenchild
```

Figure 8: top 15 authorities and hub (id->name) for the window "21Jan_30Jan"

Label propagation algorithm

Running the LPA we have assigned to each node a community, the resultis represented as follow:

```
communities = nxcom.label_propagation_communities(largest_cc)
```

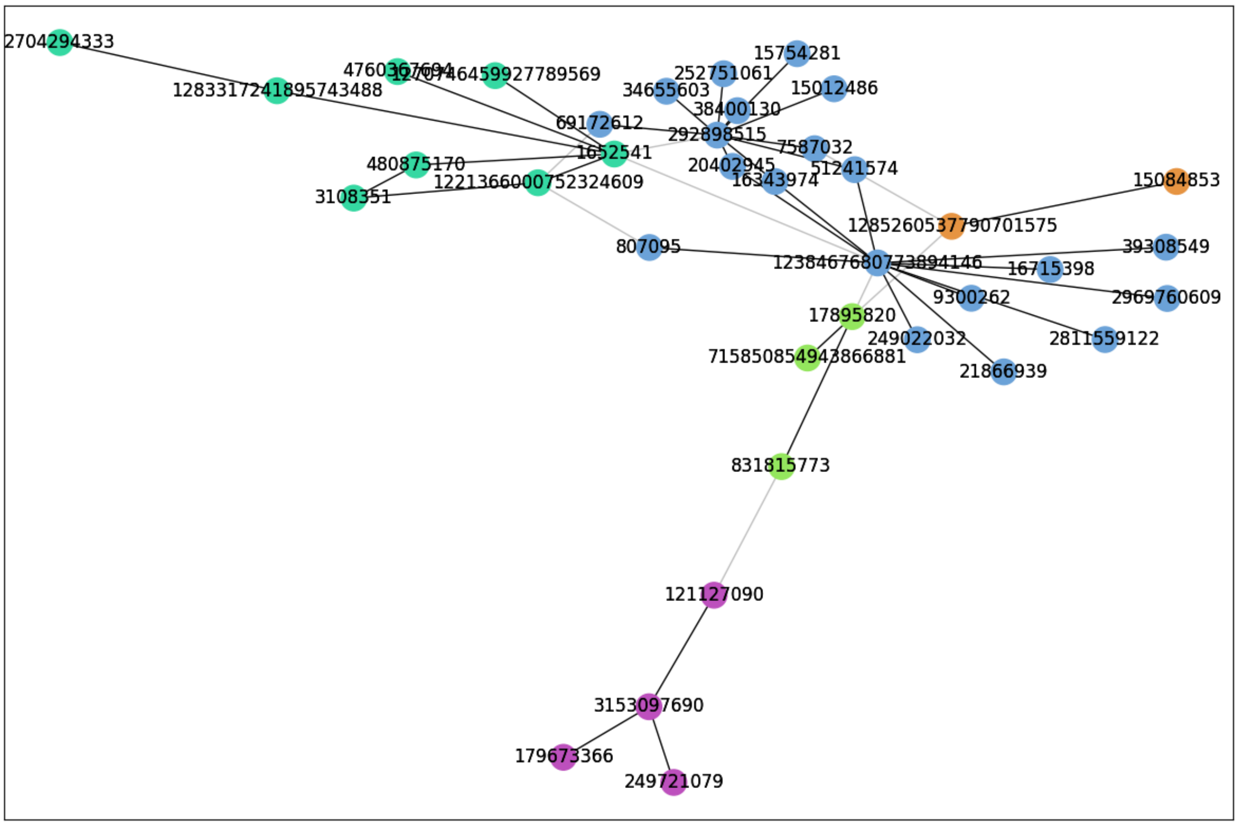
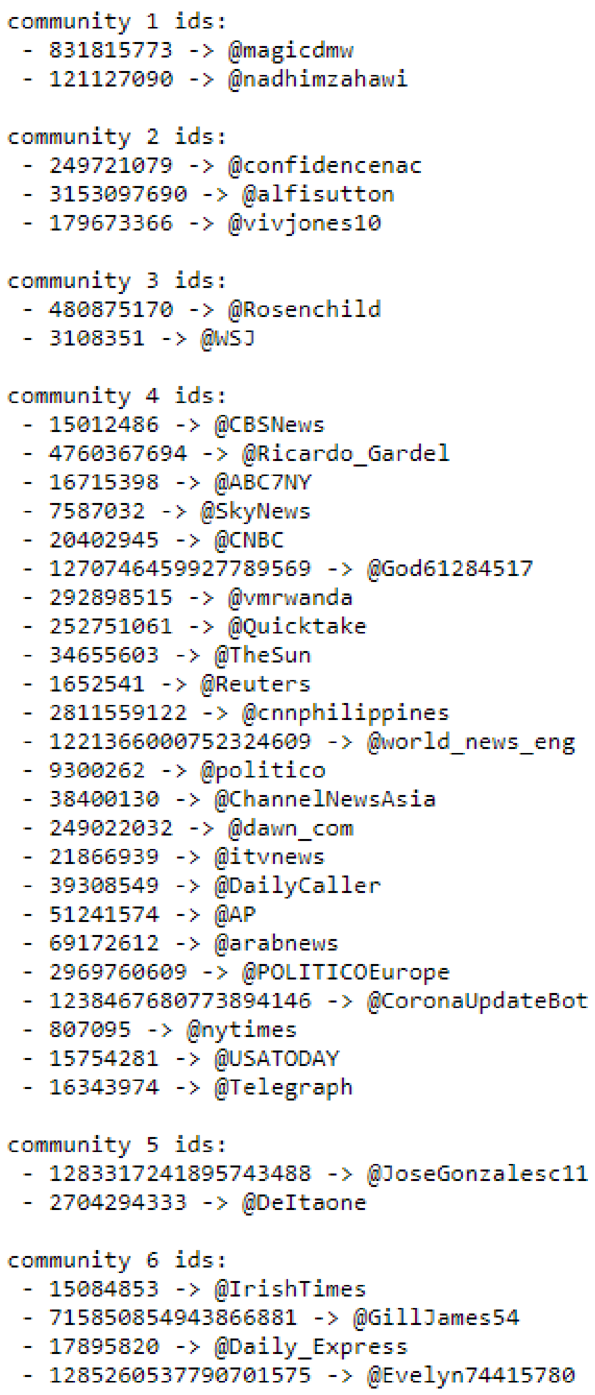


Figure 9: Plot of the output of LPA, each different color represent a different community



```
community 1 ids:
 - 831815773 -> @magicdmw
 - 121127090 -> @nadhimzahawi

community 2 ids:
 - 249721079 -> @confidencenac
 - 3153097690 -> @alfisutton
 - 179673366 -> @vivjones10

community 3 ids:
 - 480875170 -> @Rosenchild
 - 3108351 -> @WSJ

community 4 ids:
 - 15012486 -> @CBSNews
 - 4760367694 -> @Ricardo_Gardel
 - 16715398 -> @ABC7NY
 - 7587032 -> @SkyNews
 - 20402945 -> @CNBC
 - 1270746459927789569 -> @God61284517
 - 292898515 -> @vmrwanda
 - 252751061 -> @Quicktake
 - 34655603 -> @TheSun
 - 1652541 -> @Reuters
 - 2811559122 -> @cnnphilippines
 - 1221366000752324609 -> @world_news_eng
 - 9300262 -> @politico
 - 38400130 -> @ChannelNewsAsia
 - 249022032 -> @dawn_com
 - 21866939 -> @itvnews
 - 39308549 -> @DailyCaller
 - 51241574 -> @AP
 - 69172612 -> @arabnews
 - 2969760609 -> @POLITICOEurope
 - 1238467680773894146 -> @CoronaUpdateBot
 - 807095 -> @nytimes
 - 15754281 -> @USATODAY
 - 16343974 -> @Telegraph

community 5 ids:
 - 1283317241895743488 -> @JoseGonzalesc11
 - 2704294333 -> @DeItaone

community 6 ids:
 - 15084853 -> @IrishTimes
 - 715850854943866881 -> @GillJames54
 - 17895820 -> @Daily_Express
 - 1285260537790701575 -> @Evelyn74415780
```

Figure 10: Community users

2.3.2 G^t Analysis

We have implemented this task in the notebook **NetworkAnalysis_Gt.ipynb**, at first we read the G^t for each cluster of the window, and we marge them to build the requested G^t graph.
Now is computed the largest connected component and are applied the following algorithm: HITS, LPA

```
##read the graph from .zip file
def graph_from_zip(file):
    return G

# Drawing Function
def draw(graph, node_color='lightblue', edge_color='black'):



    # get largest connected components of G
def largestCC(G):
    return s

def mergeClusterGraphs(graphs, weight_LOW_threshold=0,
    weight_UP_threshold=10000):
    return Gt

def describe(original, largest_cc):

# Generate subgraphs associated with every cluster of every window (in this
    case we will show it just for the window "21Jan_30Jan")
windows = []
curr_dir = "../socialMining/src/main/java/Main/timeseries_windowsByDay"
for subdir, dirs, files in os.walk(curr_dir):
    for dir in dirs:
        clusters_dir = curr_dir + "/" + dir + "/Users_Graphs"
        if os.path.isdir(clusters_dir):
            print("-"*50 + dir + "-"*50 + '\n')
            graphs = []
            for filename in os.listdir(clusters_dir):
                G = graph_from_zip(clusters_dir + "/" + filename)
                graphs.append(G)

# generate Gt graph by merging all cluster associated graphs
```

```
LOW_threshold = 60

# generate Gt graph by merging all cluster associated graphs
# here we define an upped bound to because we had identified twitter’s bot with
    high weights on arcs
LOW_threshold = 45
UP_threshold = 55

Gt = mergeClusterGraphs(graphs, LOW_threshold, UP_threshold)

largest_cc = largestCC(Gt)
describe(Gt, largest_cc)
draw(largest_cc)
```

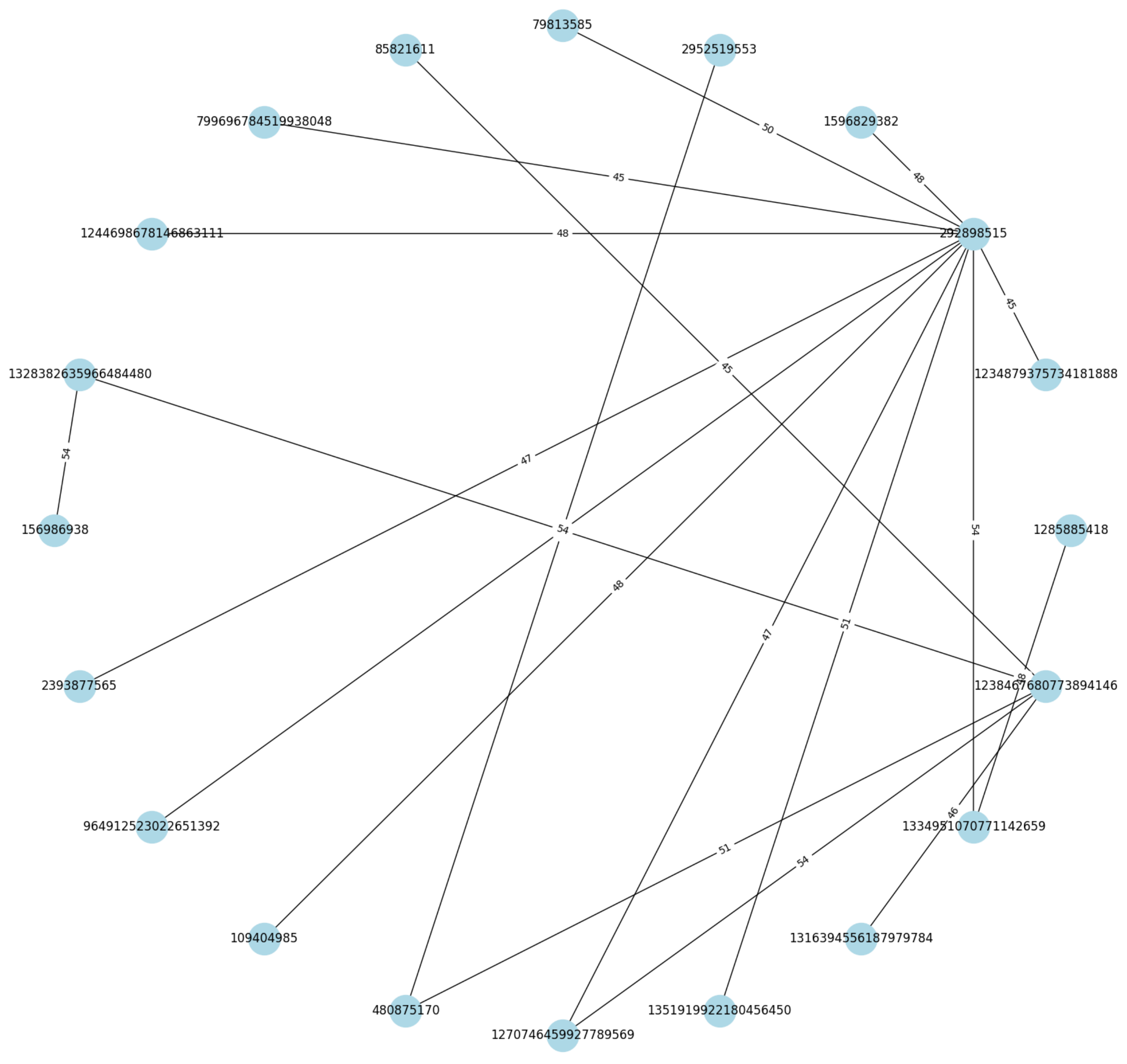


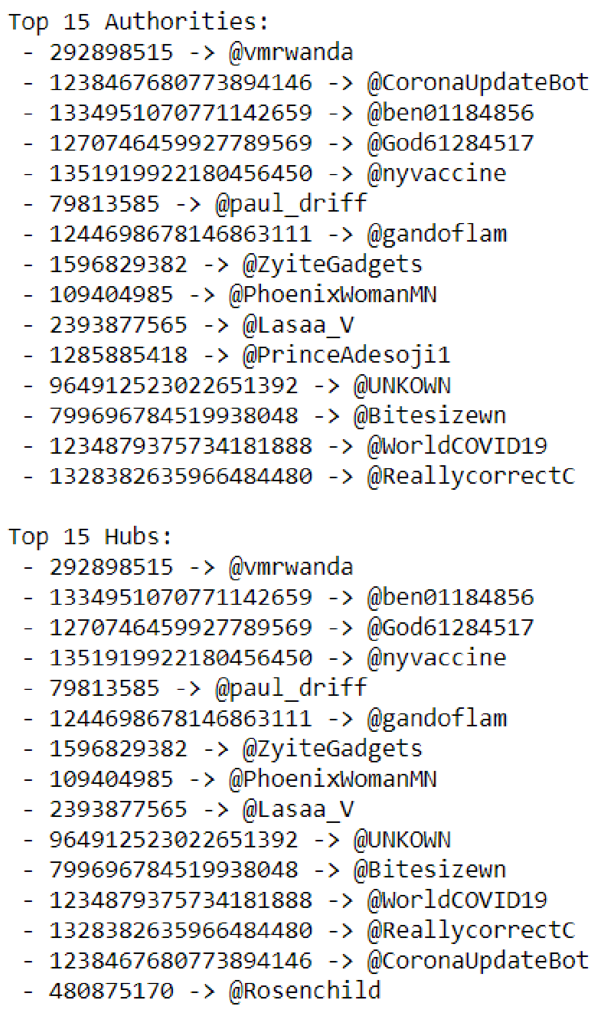
Figure 11: Largest connected component for the window "21Jan_30Jan"

HITS algorithm

Running this algorithm on the identified largest connected component we print out the id and name of the top 15 authority users and the top 15 hub users,

```
h, a = nx.hits(largest_cc)
```

Output of HITS algorithm



```
Top 15 Authorities:
 - 292898515 -> @vmrwanda
 - 1238467680773894146 -> @CoronaUpdateBot
 - 1334951070771142659 -> @ben01184856
 - 1270746459927789569 -> @God61284517
 - 1351919922180456450 -> @nyvaccine
 - 79813585 -> @paul_driff
 - 1244698678146863111 -> @gandoflam
 - 1596829382 -> @ZyiteGadgets
 - 109404985 -> @PhoenixWomanMN
 - 2393877565 -> @Lasaa_V
 - 1285885418 -> @PrinceAdesoji1
 - 964912523022651392 -> @UNKOWN
 - 799696784519938048 -> @Bitesizewn
 - 1234879375734181888 -> @WorldCOVID19
 - 1328382635966484480 -> @ReallycorrectC

Top 15 Hubs:
 - 292898515 -> @vmrwanda
 - 1334951070771142659 -> @ben01184856
 - 1270746459927789569 -> @God61284517
 - 1351919922180456450 -> @nyvaccine
 - 79813585 -> @paul_driff
 - 1244698678146863111 -> @gandoflam
 - 1596829382 -> @ZyiteGadgets
 - 109404985 -> @PhoenixWomanMN
 - 2393877565 -> @Lasaa_V
 - 964912523022651392 -> @UNKOWN
 - 799696784519938048 -> @Bitesizewn
 - 1234879375734181888 -> @WorldCOVID19
 - 1328382635966484480 -> @ReallycorrectC
 - 1238467680773894146 -> @CoronaUpdateBot
 - 480875170 -> @Rosenchild
```

Figure 12: top 15 authorities and hub (id->name) for the window "21Jan_30Jan"

Label propagation algorithm

Running the LPA we have assigned to each node a community, the result is represented as follow:

```
communities = nxcom.label_propagation_communities(largest_cc)
```

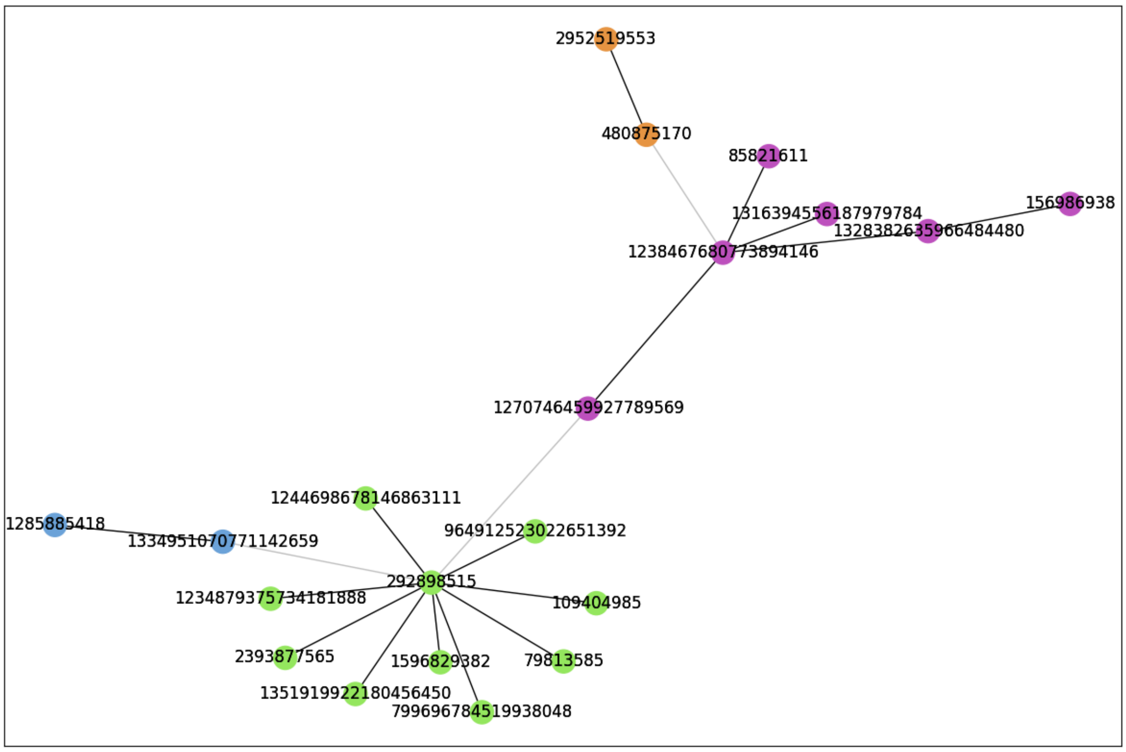
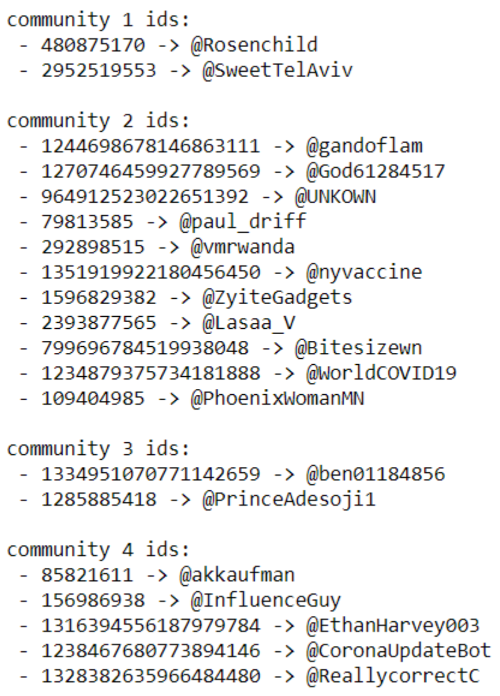


Figure 13: Plot of the output of LPA, each different color represent a different community



```
community 1 ids:
 - 480875170 -> @Rosenchild
 - 2952519553 -> @SweetTelAviv

community 2 ids:
 - 1244698678146863111 -> @gandoflam
 - 1270746459927789569 -> @God61284517
 - 964912523022651392 -> @UNKOWN
 - 79813585 -> @paul_driff
 - 292898515 -> @vmrwanda
 - 1351919922180456450 -> @nyvaccine
 - 1596829382 -> @ZyiteGadgets
 - 2393877565 -> @Lasaa_V
 - 799696784519938048 -> @Bitesizewn
 - 1234879375734181888 -> @WorldCOVID19
 - 109404985 -> @PhoenixWomanMN

community 3 ids:
 - 1334951070771142659 -> @ben01184856
 - 1285885418 -> @PrinceAdesoji1

community 4 ids:
 - 85821611 -> @akkaufman
 - 156986938 -> @InfluenceGuy
 - 1316394556187979784 -> @EthanHarvey003
 - 1238467680773894146 -> @CoronaUpdateBot
 - 1328382635966484480 -> @ReallycorrectC
```

Figure 14: Community users

2.4 Top 15 users

based on the results of Hits algorithm we have done an analysis on the top 15 Users on graph G^t and graph G^m.

2.4.1

G^t top 15 users We can mostly identify three type of users :

- **bots that provide information about the pandemic**
  - https://twitter.com/coronaupdatebot
  - https://twitter.com/nyvaccine
- **Profiles of newspaper**
  - https://twitter.com/WorldCOVID19
  - https://twitter.com/Bitesizewn
- **Very active users in the dissemination of news regarding politics and health**
  - https://twitter.com/PrinceAdesoji1
  - https://twitter.com/God61284517

2.4.2

G^m top 15 users The main difference between the top 15 users on G^m and G^t is that on G^t there were more users correlated to the pandemic, while users in G^m are simply news-pages but not directly related to covid.
This happen because their tweets are the most retweeted or the page has a lot of mentions. the main reason why top G^m users are not related to covid is because G^m is not build upon interesting tokens, but only by correlation of users. We can mostly identify one type of users :

- **Profiles of newspaper**
  - https://twitter.com/quicktake
  - https://twitter.com/Reuters
  - https://twitter.com/CNBC
  - https://twitter.com/ChannelNewsAsia

2.5 Community analysis

$G^m graph$
We have identified some correlation between members of the same community.

- **Community 1** : contains two member, a covid scientist and a woman interest in science and covid, both living in England.
- **Community 2**: contains three members, a photographer from Suffolk, and two mothers of family interested in the same topics (one of them came from Suffolk)
- **Community 3** : contains only news-pages
- **Community 5** : contains two guy interested in crypto-currencies

$G^t graph$
We have identified some correlation between members of the same community.

- **Community 1** : Can't find correlation
- **Community 2**: contains only covid news related users
- **Community 3** : contains two users that interact a lot with SkyNews
- **Community 5** : contains users that provide news

3 How to run code

The directory "socialMining" is the Intellij project and "SM_project_python" contains the python code.

Follows the right way of execution all the code described so far.

NB: There are steps that will take a lot to be executed.

(1) To index the dataset run the main method in:
"*src/main/java/Main/Index/DatasetIndexer.java*"

(2) To write timeseries of each window, of their best 5000 token, in their associated directory run: "*src/main/java/Main/timeSeriesRetriever/main.java*";

- it creates the files containing a a json of the form: {<token>: <Timeserie>} into
"*src/main/java/Main/ < WindowsDir > / < Single_windowDir > /Timeseries.gzip*"

(3) Run the file "*src/main/java/Main/saxGenerator/main.java*" in order to:
- write to file "*src/main/java/Main/ < WindowsDir > / < Single_windowDir > /saxRepresentations_alpha_2.gzip*" the json {"Token": <TOKEN>, "sax": <SAX>} for each timeserie, associated with the token, in the file:
"*src/main/java/Main/ < WindowsDir > / < Single_windowDir >> /Timeseries.gzip*"
- write to the file "*src/main/java/Main/ < WindowsDir > / < Single_windowDir > /collectiveAttention_tokens_alpha2.gzip*" the json { "Token": <TOKEN>, "sax": <SAX> } for each token which timeserie match the pattern of collective attention

(4) Run*SM_project_python/k_means_clusterer.ipynb*" in ordeer to clusterize sax strings of the tokens of collective attention for each cluster of every window
- it creates the file "*src/main/java/Main/ < WindowsDir > / < Single_windowDir > /clusters_i.gzip*" in which each line contains the top tokens of the i_th cluster

(5) Run the main inside "*src/main/java/Main/clusters/coOccurenceGraph_generator.java*" in order to create the co-occurence graphs associated with clustered tokens.
- it created the file with the graph in json form at:
"*src/main/java/Main/ < WindowsDir > / < Single_windowDir > /clustersGraphs/ < clusterGraphName > .gzip*"

(6) Run the python code "*SM_project_python/clustersAnalizer.ipynb*" to analize clusters and visualize their related graphs and tokens timeseries

(7) Run "*src/main/java/Main/networkAnalysis/users_Gm_retriever.java*" to build GM graph associated with each window
- it create the file:
"*src/main/java/Main/ < WindowsDir > / < Single_windowDir > /Users_Gm_Graphs /Gm_graph.gzip*"

(8) Run*src/main/java/Main/networkAnalysis/users_Gt_retriever.java*" to build a cluster-splitted version of the GT graph (that the python code will merge later on)
- it creates all graphs inside a gzip in the directory "*src/main/java/Main/ < WindowsDir > / < Single_windowDir > /Users_Graphs_i.gzip*"

(9) Run "$SM\_project\_python/NetworkAnalysis\_Gm.ipynb$" and "$SM\_project\_python/$ $NetworkAnalysis\_Gt.ipynb$" to analyze the resulting graphs and the application of ranking algorithms on them