*Datas :*

The dataset is saved in several json files which are used to build a graph.

The two main json file are :

- **json\_tweet.json** which is a list of dictionaries (one per tweet), each dictionary has the keys  'tweet\_id','tweet\_text','user\_id'.

- **json\_users.json**  is also a list of dictionaries (one for each user that has a tweet in json\_tweets.json)

with the keys 'user\_id', 'friends', 'statuses\_count' (friends is a list with the id of users followed by the user, statuses\_count is the number of tweets he has published)

from these two json files, the following json file has been created. This file uses the informations from json\_users.json and json\_tweets.json to have all the information necessary to build the graph in one place.

- **json\_users\_for\_graph.json** which is the same as **json\_users.json**  but has two added informations :

\* polarity, which is the average of the polarities (a number between -1 and 1 representing the positive or negative aspect of a text) of the tweets of this user, using the get\_sentiment function from textblob\_analysis.py (textblob module)

\* vec, which is a 300 dimensional vector representation of the tweets of the user. This representation is made by obtaining a representation of each word from a word2vec pretrained model, and averaging theses vectors over the sentence with the tf-idf weights

(using the embed\_data function from tweet\_to\_vec.py)

Note : Several similarity measures have been constructed from different features. Even though we only use the vector representation of tweets in our final graph, attempts have been made with features such as the number of tweets published by the users, the nulber f tweets published by its friendsi, or a polarity of sentiment measure from the python textblob module.

*Tweet cleaning*

The tweets have been cleaned for processing using the functions from the tweet\_cleaner.py file (suppression of non alphanumeric characters, handling of hastags, lowercasing…)

*Tweet embedding*

The vector representation of the tweets has been done with the function embed\_data from the tweet\_to\_vec.py file. It uses word embeddings from fasttext and a tf-idf weightage of the words to get an embedding of a tweet.

*Graph*

The function get\_graph from the network creation.py file takes the json file **json\_users\_for\_graph.json** and uses its infos to return a directed weighted graph.

- each node is a user, the edges represent the follow relationship

- the weights are, in the version in which it is, where

And vec1, vec2 are the vectors of the two users involved.

*Clustering*

The idea was to find two communities and see what criterion was characteristic of each.

For this, we used a spectral clustering algortihm.

The function clustering2 from clustering.py does this.

It takes a graph G as given by the function get\_graph, and returns a numpy array that represents the labels of each node in the graph.

The clustering function, also in clustering.py, has been used in a previous attempt to detect community. It is based on a generalization of the minimum cut search from the following reference :

[M. Meil˘a, W. Pentney, Clustering by weighted cuts in directed graphs, in: SDM ’07: Proceedings of the 2007 SIAM International Conference on Data Mining, 2007]

*Analysis*

The function cluster\_analysis from cluster analysis.py takes as input the labels as returned by the clustering2 function and returns a list containing for each class, the 5 users with the highest in degree, and their tweets.

The function get\_high\_ranked takes a networkx graph G as input and returns the most important nodes according to the pagerank algorithm ran on G