Italian Referendum Analysis

Gabriel Nespoli[1], Sergio Ballesteros[2]

Introduction

In this present work, we performed a social and text analysis of the Italian constitutional referendum held in Italy on Sunday 4 December 2016. In this referendum, voters were asked whether they approve a constitutional law that amends the Italian Constitution to reform the composition and powers of the Parliament of Italy, as well as the division of powers between the State, the regions, and administrative entities. Voters had two options in the poll: yes and no.

The data that we analyze comes from Twitter. The aim of the analysis were to extract information about the content of the tweets as well as the users relationship with each other. The initial point of the study was to select 150 influencers of each group that clearly support the yes or the no in the poll. These influencers were politicians or former politicians, candidates, journalists, actors or actresses.

The dataset contains 1.9 million tweets (76 GB of data) obtained from the Twitter stream from the 26/11/2016 until the 06/12/2016. The programming language used was Java and some of the technologies used were Twitter4J, Maven, Apache Lucene, SAX and G graph library. It was also provided a directed graph created through the tweets between 26/11/2016 a 06/12/2016, where in the tail of an edge is a user that mentioned another user in the head of the edge. This graph is called *original graph* in this report.

Here below we present our methodology and analysis of the results. In the Appendix it is possible find the steps about how to run our code in order to replicate our results.

0. Temporal Analysis

1. Data collection

We gathered from the Internet 456 politicians and influencers for the option yes and 390 influencers that supported the no in the referendum. For simplicity we will refer to this group of users simply as politicians. The supporters names were found in the OpenParlamento website[3] and in Wikipedia[4].

[1] mesquitanespoli.1743585@studenti.uniroma1.it
[2] sergio@ballesteros.me
[3] https://parlamento17.openpolis.it/lista-dei-parlamentari-in-carica/camera/nome/asc
[4] https://en.wikipedia.org/wiki/Endorsements_in_the_Italian_constitutional_referendum,_2016

It was developed a method that reads a name and search on Twitter this user. After that, it is retrieved the most related Twitter account of this name. Not all these politicians had a Twitter account, or at least we could not find it in an automatic way their Twitter account with the method mentioned above. A subset of 158 and 134 people were found for the yes and no group, respectively.

A code was developed in order to get a full name, connect to the Twitter via Twitter4J API, search the name and retrieve the most related user of the response. Thus, we read the Twitter account of this user and saved it into a file.

We counted the number of times that one of the politicians tweeted and also kept the timestamp of the tweet to show a temporal distribution. The politicians defending the No have tweeted 9403 times, while the Yes group have tweeted in the same period of time 5754 times.

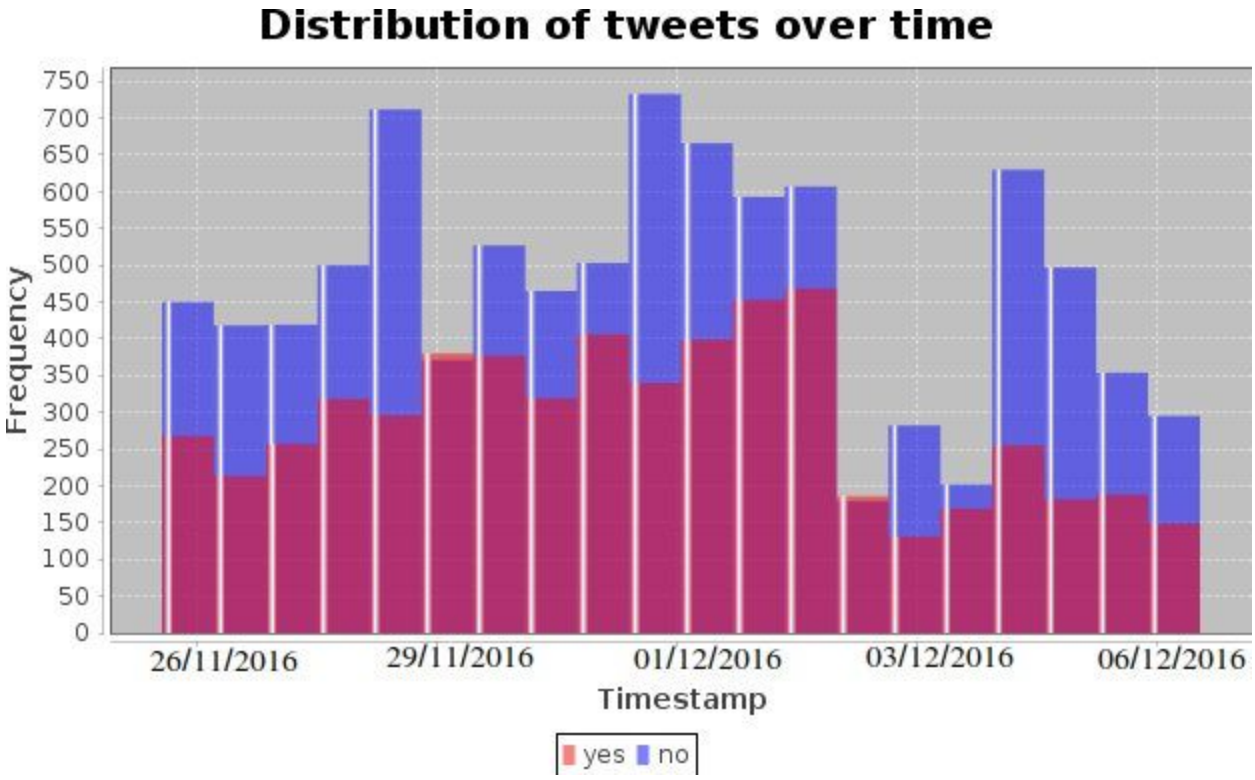


Figure 1. Distribution of tweets over time

2. Clustering of similar pattern words

The first step of this section was to create the Lucene index. Thus, we have went through through the entire collection of tweets adding to an index the tweets written by, or replied by, an influencer. If the politician votes for No, the document was added to /no_index, otherwise it was added to /yes_index.

After creating the index, it was retrieved the top 1000 most frequent terms used by each group of politicians. After that, we used each index to create a timeseries of the frequency of all

the 1000 terms (frequency of term X = number of uses of X / time interval). The granularity used was 12 hours.

Later, we represented the time series in a sequence of characters using the SAX algorithm. In order to obtain the terms that exposed a similar temporal behaviour, we implemented a k-means algorithm that to calculate the centroid between two SAX strings follows the steps described below with an example:

Input *= {s1 = [a,b,b,d], s2 = [c, e, g, z]}*

1. *Retrieve s1[i] and s2[i], which are a and c, respectively*
2. *Get the ASCII value of the characters and calculate the mean*
   (ascii(a) + ascii(c)) / 2
   (97 + 99) / 2 = 98
3. *Map back from decimal to character*
   char(98) = b
4. *Increment i and repeat from step 1*

The ***output*** *of this example would be s3 = [b,d,e,o]*

3. Co-occurrence graph

We built an undirected graph for each cluster and in each graph the nodes are the different words existing within the cluster. One node is connected with another node with an edge if the words that represent the nodes appear in the same tweet. The weight of the edges are the number of times that the two words appear in the same tweet. We used Lucene for this task. We normalized the weights of the edges dividing by the maximum weight of the greatest edge and kept only the edges that had a weight greater than 0.07. Then, for each graph, we found the Largest Connected Component (LCC) and also found the innermost core via k-Core. Both tasks were done using the library G[5].

We found that the subgraphs extracted using k-Core were significantly smaller than the ones extracted using LCC, on the average 5 times smaller.

Those words can be found in the following resource files:

yes_largestcc.txt
no_largestcc.txt
yes_kcore.txt
No_kcore.txt

Where the files contain three columns, the first two ones are nodes (words) and the third one is the cluster ID where they belong. Here we show a sample of the cluster ID 6 for the Yes group using the Kcore:

matteorenz forza

[5] https://github.com/giovanni-stilo/G

matteorenz graz
matteorenz ital
matteorenz matte
matteorenz seren
matteorenz viva
matte forza
matte graz

The words appear trimmed because of the method used during the inverted index generation. In this cluster the words clearly show a positive sentiment towards Matteo Renzi, ex Prime Minister of Italy who promulgated the referendum.

4. Time series clustering

We read the the tokens of each cluster calculated in the section 0.3 using both strategies (k-Core and LCC) and traced the time series with granularity of 3h of each term. We saved a graph comparing the time series of terms of each cluster. All the figures can be found in the resources folder under the subfolder called images, here we show three examples.

In Figure 2, it can be observe the similar temporal behaviour of the terms *iodicono* and *iovotono*

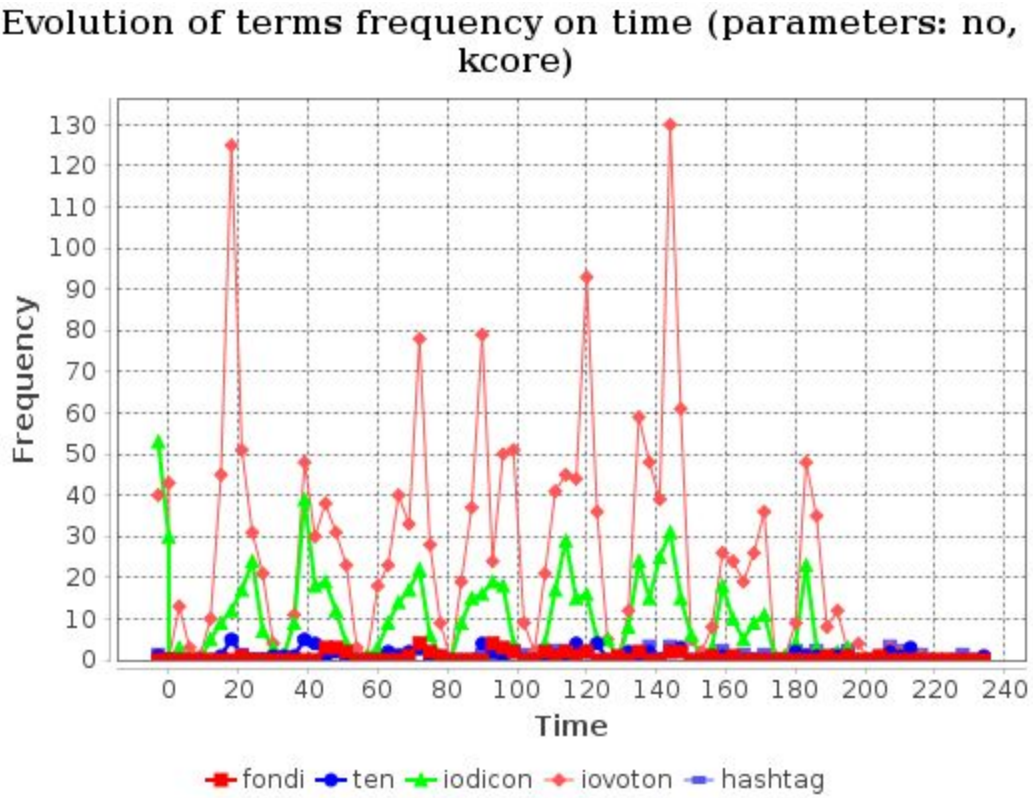


Figure 2. Frequency of terms of cluster 0 of ‘no’ group

In Figure 3, it can be seen other very correlated terms with the referendum.

Figure 3. Frequency of terms of cluster 6 of ‘no’ group

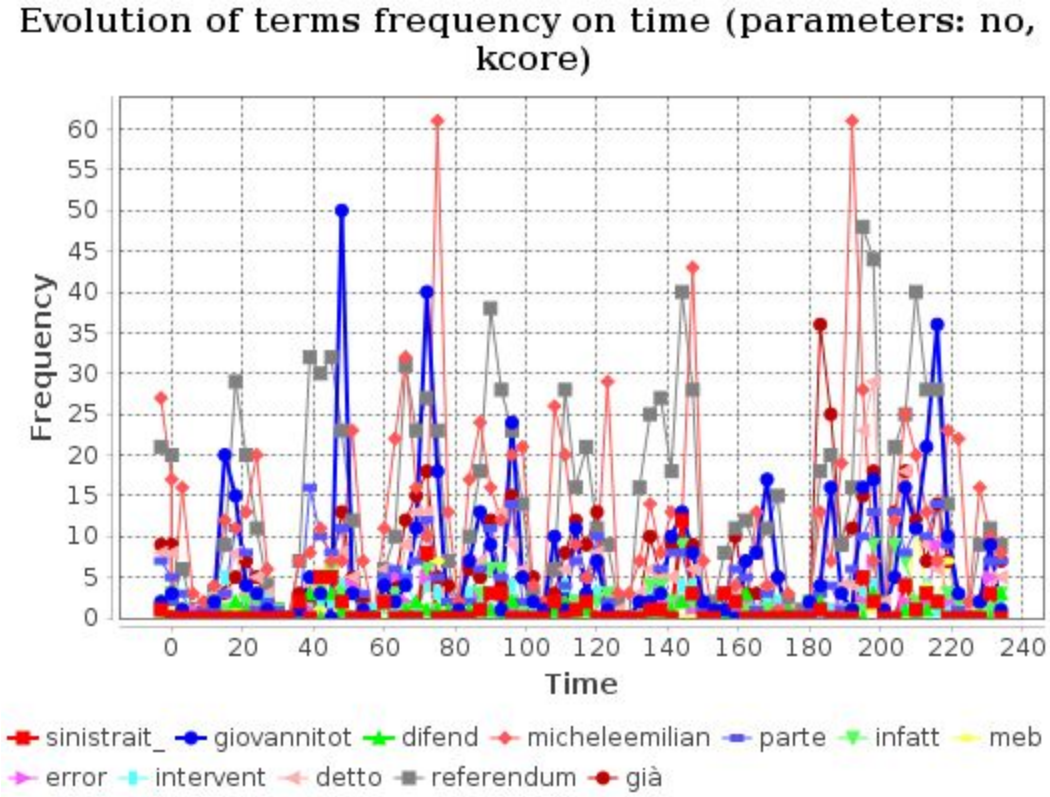
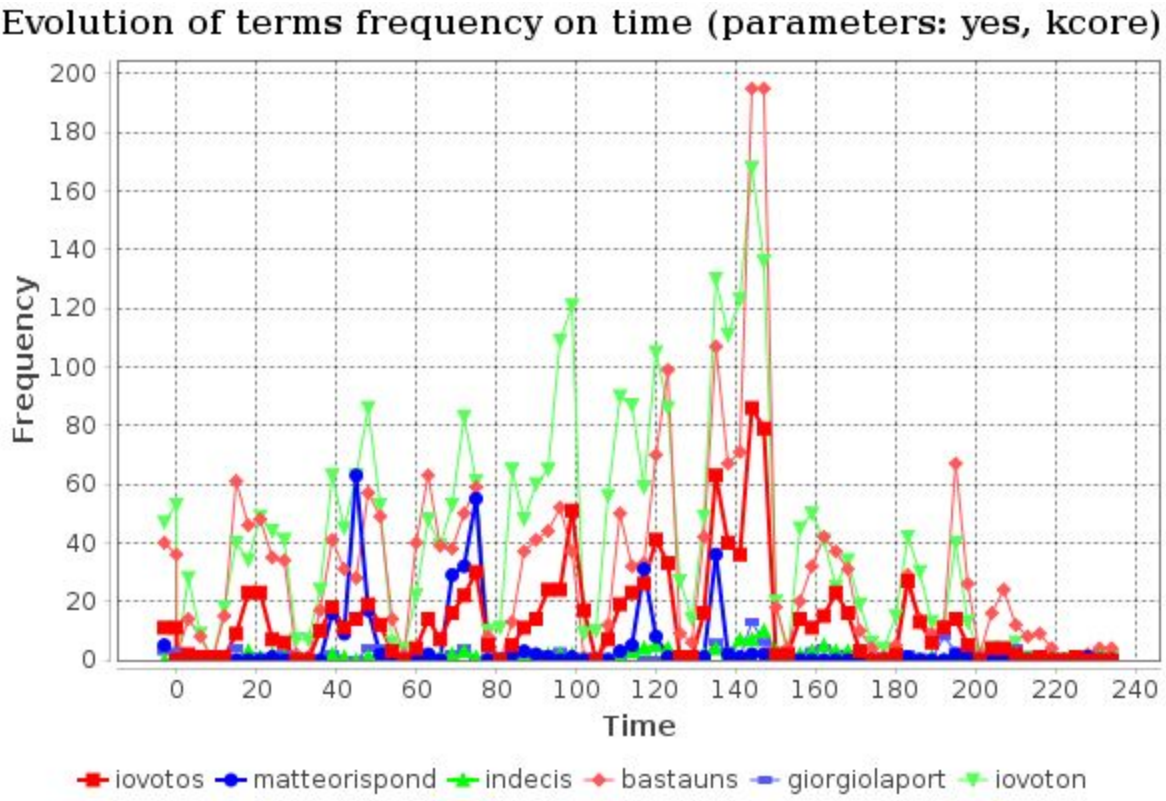


Figure 4. Frequency of terms of cluster 19 of 'yes' group



In Figure 4, *iovotosì*, *matteorisponde*, *bastaunsì* are put together in the same cluster and are example of terms related to the 'yes' group. It is interesting to noticed that *iovotono* has a strong similarity with the aforementioned words. A possible explanation is that the no-supporters may be reacting to the video when Matteo Renzi was addressing about the referendum on live online. The same explanation could be given to the presence of *giorgiolaporta*, the Twitter name of Giogio La Porta, a writer and founder of Centro-Destra.it, an Italian newspaper that supported the No during the referendum.

1. Identifying Yes/No supporters

1. Identifying users that mention the politicians and keywords

We created an inverted index using Lucene for the whole collection of tweets. Then we found the set of users $\mathcal{U}$ that mentioned at least one politician. To do that, we ran the Lucene query for both Yes politicians and No politicians for the content of the tweets:

Politician1 OR Politician2 OR ... OR PoliticianN

And kept the set of users $\mathcal{U}$ from the resulting documents. Also we used the words from the clusters created in point 0.3 and ran the following query for the Yes and No clusters:

Word1 OR Word2 OR ... OR WordN

We stored the users from the list of results. Since the users can mention Yes and No politicians at the same time, we decided that a user is a Yes or No supporter in the following way:

Using users from set U, we calculated using Lucene how many times each user mentioned a Yes politician (n_{YP}), a No politician (n_{NP}), a Yes keyword from the previously created clusters (n_{YW}) and a No keyword (n_{NW}). Then, a user is a Yes supporter if at least one of the two following condition holds:

- $n_{YP} > n_{NP}$ AND $n_{YW} > n_{NW}$
- $n_{YP} < n_{NP}$ AND $n_{YW} > n_{NW}$

Similarly, the user is a No supporter if at least one of the two following conditions holds:

- $n_{YP} > n_{NP}$ AND $n_{YW} < n_{NW}$
- $n_{YP} < n_{NP}$ AND $n_{YW} < n_{NW}$

We used this method because it takes into account that an user can mention one politician in a positive or a negative way. This leads to 55 K Yes supporters and 190K No supporters. We can clearly see an imbalance where the number No supporters is much greater . The total number of tweets for each group of supporters is approximately 0.3M and 3.5M, respectively. We calculated this number taking a sample of 1000 users for each group and finding the number of tweets that those users created. We stored the usernames in the files yes_M.txt and no_M.txt.

2. Identifying the top 1000 highest authority ranked users

We selected all the users retrieved from the previous step, unifying both groups of supporters and extracted from the original graph just the subgraph induced by the users that mentioned a politician. Later we extracted the largest connected component of this induced subgraph. After this, it was possible to get the top 1000 authorities through the method *HubnessAuthority.compute* present in the G library. This list was saved in the file *top_authorities.txt* in the folder of the resources of the project in the following format:

<node_rank_1>,<authority_value>
<node_rank_2>,<authority_value>
...
<node_rank_1000>,<authority_value>

The output of our code results in:

837 of the top authorities mentioned the Yes supporters
853 of the top authorities mentioned the No supporters
147 of the top authorities mentioned just the Yes supporters
163 of the top authorities mentioned just the No supporters

*Table1: Top authorities that mentioned just the **Yes/No** supporters*

	Yes	**No**
1	fannicanelles	spinozait
2	lgatto4	ilpost
3	rotanicolas	welikechopin

The 6 authorities exemplified above there are an influent professor of the University of Bologna with 141K followers, 2 social media writers, a blogger, an online newspaper and a TV show.

3. Identifying the top 500 most central users

We followed two methods:

a. Using the extracted subgraph

Using the previously extracted subgraph described in part 1.2 we ran the HITS algorithm from the G library to calculate the authority and hubness scores for all the nodes. In order to keep only users who mention frequently the politicians, we kept only the users who mentioned 3 times or more either the Yes or No politicians. We found only 15 yes supporters and 75 no supporters. In the next table it is possible to see the top 3 for each group:

Table 2. Top 3 most central accounts of Yes/No group using the subgraph HITS

	Yes supporters	**No supporters**
1	pdnetwork	nellina99
2	zazagab2	repubblica
3	enzo958	ardovig

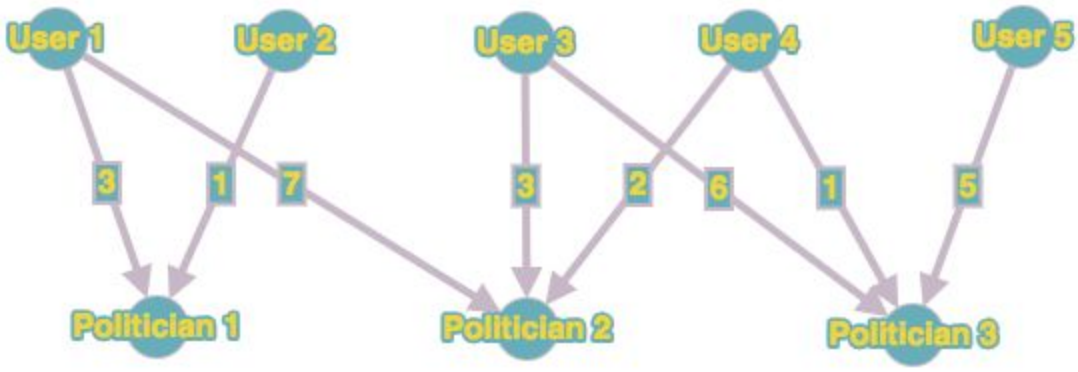
The Twitter ID of the top users is stored under the files:

top_hubness_authorities_no.txt
top_hubness_authorities_yes.txt

A manual inspection to the top 3 users found in each category confirmed us that those users are heavily involved in politics, with a bast number .

b. Creating a new graph

Instead of using the provided graph, we built a directed graph where the nodes with outgoing edges are the users, and the nodes with the incoming edges are the politicians. We traced edges between nodes when an user mentioned a politician. Therefore this graph instead of being based on who follows whom, is based on the Twitter mentions. The weight of the edges correspond to the total number of mentions of the politician by the user. Since the size of the graph was only about 247K nodes, we decided to not use any threshold and keep all the edges in this step. Again, we used Lucene to find all that information.



Concept of the directed graph built: The User 1 mentioned 3 times the Politician 1 and 7 times the Politician 7.

This graph is almost a bipartite graph. This is because a politician can mention another politician or a normal supporter, but this happens very rarely compared to the volume of mentions for normal users to politicians. Therefore HITS will give more weight in the hubness values to the Users and more authority to the politicians.

Using this graph, we ran the HITS algorithm from the G library, to calculate the Hubness and Authority scores of all the nodes in the graph. After that, we computed for each node the final centrality score using average Hubness and Authority scores. We sorted the nodes from most central (highest final score) to least central. We took our previous results from point 1.1

(sets of Yes and No supporters), and kept the top 500 most central users for the Yes supporters and the top 500 most central users for the No supporters.

This method lead to the following top users for each group:

Table 3. Top 3 most central accounts of Yes/No group using the almost bipartite graph and HITS

	Yes supporters	**No supporters**
1	masdepasca2	UnicronPlanetM
2	MarcoCav	SteccaAlessio
3	Be_Strega	G_ianluc_a

As expected, this users have a huge amount of writen tweets, in the order of thousands, but they have less followers that the users found with the method described in a).

4. Identifying the top 500 k-Players

In the step 1.2 it was saved in an intermediate step the graph composed by the largest connected component of the subgraph induced by the M group (created in 1.1), *graph_largest_cc_of_M.gz*.

We start 1.4 loading this subgraph and From the list of Twitter usernames created in the step 1.1 on the *yes_M.txt* and *no_M.txt* files, we retrieved their respective Twitter ID (long format) calling the method *IndexSearcher.searchByField*. We then mapped the long value to int using an object of the class *LongIntDict*. The int value represented the respective position of this user in the graph. We ended up with an array of ints. The following steps represent the conversions made:

list of Twitter names → Twitter name to Twitter ID (Long) → long to int → array of int

We extracted a subgraph from the original graph composed by the nodes of this array of int (the M users), we filtered some nodes that had a low degree, then retrieved the k-players of this remaining graph through the method *KppNeg.searchBroker.*

The top 500 players are saved in the files yes*_top_k_players.txt and* no*_top_k_players.txt*.

Below in Table 1, it can be seen the top 3 players of each group. Again, searching for these users on Twitter, we could prove their importance in the referendum. They are accounts of a major political party that supported the Yes, an openly supporter of Matteo Renzi, a writer, a blogger and a politician.

Table 4. Top 3 players of Yes/No group of supporters according to KppNeg algorithm

	Yes supporters	**No supporters**
1	pdnetwork - 664	mlabriola73,135776
2	renzinews - 399	1_starrynight,124362
3	caprozz - 375	spinozait,116343

It is worth to comment that due to the unbalance in the size of the groups, we could not find 500 important players in the Yes group.

2. Spread of influence

Label Propagation Algorithm

We decided to implement our own Java method for the LPA, which can be found in the class *analysis.LPA*. Our method uses the whole network, and sets the labels in the following manner:

- Initially, for all the Yes supporters sets the “yes” label
- Initially, for all the No supporters sets the “no” label
- Initially, for the rest of the users of the subgraph, sets one unique label for each user. We call here this set of labels “unknown”
- After this initial modification of the standard LPA algorithm, we implement the algorithm as described in the original article[6] with the exception that only “yes” and “no” labels can propagate, “unknown” labels do not propagate.
- We assume that the Yes and No seed nodes can change their label too.
- Since our graph is a directed graph, we consider that the neighbours of one nodes are the ones that are connected to this one with an incoming edge. For example:

$$A \rightarrow B$$
$$C \rightarrow B$$
$$B \rightarrow D$$

[6] Xing, Yan & Meng, Fanrong & Yong, Zhou & Zhu, Mu & Shi, Mengyu & Sun, Guibin. (2014). A Node Influence Based Label Propagation Algorithm for Community Detection in Networks. TheScientificWorldJournal. 2014. 627581. 10.1155/2014/627581.

Here A and C follow B, so when B tweets, its information will flow to A's and C's timelines, but not to D's.

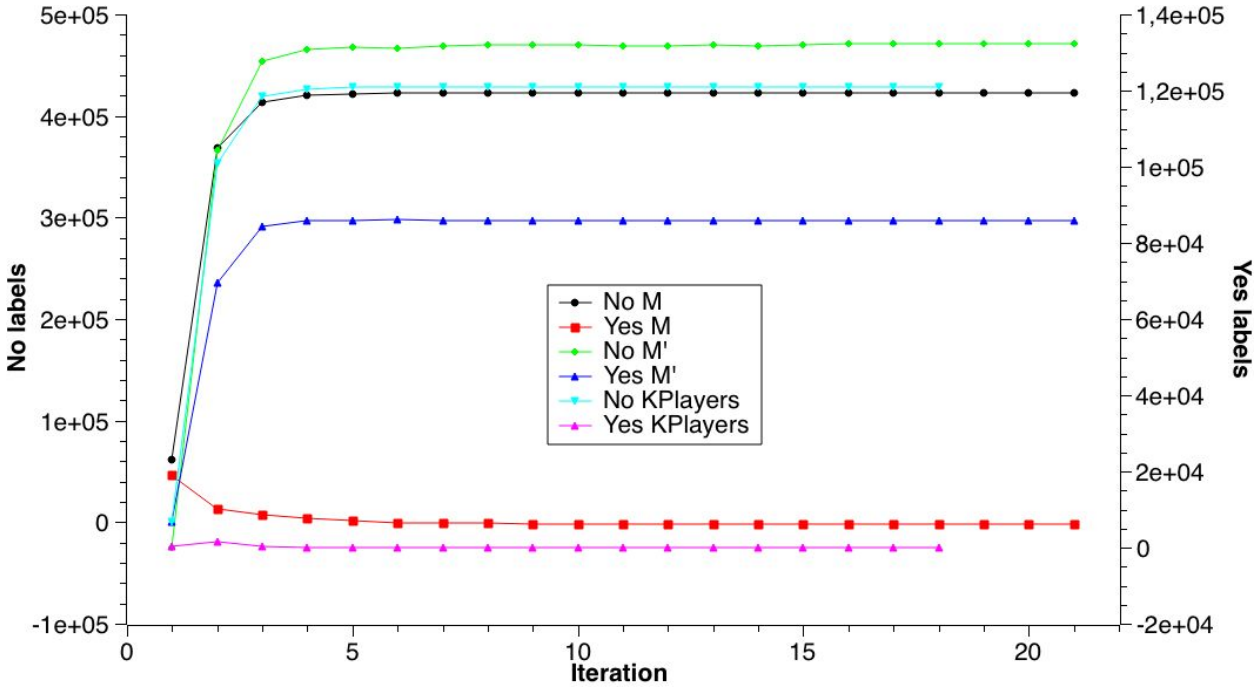
- It stops iterating when all no nodes change their labels in one iteration, or a maximum number of iterations is reached. We introduced this last condition because the system can take too long to converge due to small fluctuations.

We found the next number of nodes with the following labels:

Table 4: results of the LPA modified algorithm on the whole network

Label	Initial count M	Final count M	Initial count M'	Final count M'	Initial count KPlayers	Final count KPlayers
yes	18949	6201	143	132340	233	23
no	62084	422951	139	296429	498	428745
unknown	16778457	16386782	16815729	16387165	16815203	16387166

Also, we recorded the number of "yes", "no" and "unknown" labels on each step of the iteration:



The algorithm converges in less than 20 steps. In all cases the No label propagates much more than the Yes label. In fact, using as seed nodes the top Kplayers and M leads to just a few remaining nodes with Yes labels. This can also be cause by the heavy unbalance of users of different groups. On the other hand it seems that the opinion of the Yes M' users manages to take off and propagate on the network several orders of magnitude more.

Appendix

In this section we show how to run the code to replicate our results. The code is available at the following link:

https://github.com/gabrielnespoli/italianreferendumanalytics

The steps are the following to run the software are the following

1. Open a terminal and use git to clone our repository:

git clone https://github.com/gabrielnespoli/italianreferendumanalytics

2. Navigate into the resources folder

cd italianreferendumanalytics

3. Copy the Twitter stream data into the resources folder

mkdir -p src/main/resources/sbn-data/stream;
cp -r <path to the data>/stream src/main/resources/sbn-data/stream

4. Copy the original graph into the resources folder

cp <path to the data>/Official_SBN-ITA-2016-Net.gz src/main/resources/

5. Clean built and execute the Main method with Maven. This should take about 15 hours.

mvn clean install exec:java -Dexec.mainClass="Main"

6. All the generated files will be under src/main/resources/