Social Media Analytics - Italian Referendum - Temporal Analysis and Spread of Influence

Vigèr Durand Azimedem Tsafack, Pratuat Amatya

Malick Alexandre N. Sarr

University of Rome "La Sapienza"

July 14, 2019

Abstract

Social media has completely transformed the way people interact with each other, having an equally significant impact on businesses as well. There are now billions of unsolicited posts directly from consumers that brands, agencies and other organizations can use to better understand their target customer, industry landscape or brand perception. [1] In this experiment, as an illustration of how useful social media could be, we analyzed two data-sets, a network representing the relationships among twitter profiles and a massive quantity of date containing the stream of tweets across the Italian referendum day (4th of December). On those data-sets, we investigated the temporal distribution by performing a temporal analysis before Identifying potential Yes/No supporters. Additionally, we performed spread of influence analysis to evaluate the spread of the different parties over the network.

I. Introduction

Recent research on social media analytic has emphasized the need to adopt a business intelligence based approach to collecting, analyzing and interpreting social media data. [2] Social media presents a promising, albeit challenging, source of data for business intelligence. Customers voluntarily discuss products and companies, giving a real-time pulse of brand sentiment and adoption.[3]

Social media is one of the most important tools for marketers in the rapidly evolving media landscape. Firms have created specialized positions to handle their social media marketing. These arguments are in line with the literature on social media marketing that suggest that social media activities are interrelated and influence each other.

In this experiment, as an illustration of how useful social media can be, we analyzed twitter data across the Italian 2016 constitutional referendum date in order to evaluate the coherence between our results and the actual outcome of that referendum.

II. Methods

i. Data

The provided data-set contained a generated network consisting of over 450.193,0 twitter IDs organized in around 22.649.482,0 of edges (connections among twitter profiles). Each edge was represented on a line as follows: $source < tab > target < tab > weight$.

Additionally, We were provided a massive quantity of data containing the stream across the 4th of December divided by day, each file containing 10.000,0 tweets. The overall dimen-

sion of available data was around 10 Gigabytes.

ii. Temporal Analysis

ii.1 Tweet distribution over time

To perform analysis on the files, we needed to find a way to store, access and query the information provided within the data. For that purpose, we created 3 main indexes, related to the temporal analysis. All the text, containing relevant information have been cleaned before being saved. The built-in *Lucene* Italian analyzer has been used along with a list of Italian stop-words taken online to omit words containing little information. All the Tweet text has been stemmed, and stripped of not needed symbols such as *https*, *rt* , etc.

The first index created was the tweet index, which contained all information about the tweet. That includes but is not limited to the tweet date, users, scree-name, text, hash-tag, mentioned people. Few additional functionalities such as filtering, searching and getting tweet frequency were added to manipulate the data-set.

The next index to be created was the politician index. We first scrapped a Wikipedia page (Endorsements in the 2016 Italian constitutional referendum) containing the list of Italian politicians along with their affiliation (Yes or No) during the 2016 Italian referendum. Once that list was obtained from the web, we created a twitter look-up function that would associate the twitter user name to the politician name. In order for a politician to be considered in this analysis, he was supposed to have at least 5000 followers.

Finally we separated our tweet index between politicians having voted yes and politicians having voted no. Upon doing that we then plotted the number of tweet generated by each camp over time.

ii.2 Time series representation of tweets

In this part we performed SAX analysis on the politicians tweet for each camp (yes/no). We can think of SAX (Symbolic Aggregate approximation) as a symbolic representation of time series. The basic idea behind SAX is the fact that it allows a time series of arbitrary length n to be reduced to a string of arbitrary length w, ($w < n$, typically $w << n$). The alphabet size is also an arbitrary integer a, where $a > 2$. The algorithm consist of two steps: (i) it transforms the original time-series into the PAA representation and (ii) it converts the PAA data into a string. The use of PAA brings advantages of a simple and efficient dimensional reduction while providing the important lower bounding property. The actual conversion of PAA coefficients into letters by using a look-up table is also computationally efficient and the contractive property of symbolic distance was proven by Lin et al. [9]

To apply the SAX analysis for each camp, we first pulled up the earliest and latest date in our index in order to have a proper range for the event. Then, we extracted to top 1000 more frequent words tweeted by the politicians. Having the most frequent terms, we counted the occurrence of each term over time with an interval of 12 hours obtaining a temporal distribution of the terms by each party (Yes/No) with a grain of 12 hours. This temporal distribution has then been converted into a SAX string using the SAX package for Java. The alphabet size we used was 20. Finally we performed a unsupervised classification of the above SAX string applying the K-means clustering algorithm on the Time-Series.

The K-means algorithm is a clustering method that aims to find k centroids, one for each cluster, that minimize the sum of distance of each data point from its respective cluster centroid. It solves, for $xi \in X$: $argmin_{C_1,...,C_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} d(x_i, c_j)$ where $(C1, ..., Ck)$ are k non-overlapping clusters, c_j is the representative of clusterC_j , and d is a distance function. [10]

We chose our initial centroids by picking random points withing our cluster. The distance measure used is the Jaro Winkler distance, mainly because it gave us more balanced cluster. But the clustering was tried as well

with the Euclidean distance and the Levenshtein distance. For each sax string, we calculated the distance to each centroid and we added it to the cluster belonging to the centroid with the lowest distance. Once that was done, the centroids for each cluster is updated, by averaging each char c at position n out of N of the data-points within the cluster. We repeated the same procedure until there were no changes or we reached the maximum iteration flag.

ii.3 Time series co-occurrence graph

The co-occurrence analysis was performed through 3 main steps. We first converted the terms within the cluster to a graph, having as nodes the terms, and as edges their weight, which correspond to a normalized frequency measure obtained by extracting the frequency of occurrence of the two words. This weight will correspond to the total number of tweets that contained both term. We then applied a threshold value t which correspond to the highest frequency between the two term multiplied by a probability value p. $t = argmax(t1, t2) * p$. Finally an edge between two nodes/terms is added if their weight is above to the threshold value.

Once the graph has been created, we then proceeded in extracting the Largest Connected Components within the graph. A connected component of an undirected graph is a maximal set of nodes such that each pair of nodes is connected by a path.[13] For each cluster, its largest connected component is saved, this is done with a helper graph java library called G.

Our final step consisted of extracting the k-cores of the graph. The k-core of a graph G is the maximal induced sub-graph H G such that (G) k. Thus all vertices of H are adjacent to at least k other vertices in H.[12] The k-core was introduced by Steven B. Seidman in a 1983 paper entitled Network structure and minimum degree. In other words, k-core of a graph G is the largest induced sub-graph of G in which every vertex has degree at least k. The coreness of a vertex v in G is the largest value of k such that there is a k-core of G containing v. [11]

ii.4 Time series comparison

In order to compare the various time series, we saved the LCC and the Kcores for each cluster and for each camp(yes/no politicians). We those graph, we then went ahead and extracted the number of occurrences of each node (term) in an interval of 3 hours, then we graphically compared the behavior in order to observe if their were any similarity in the way those term occur over time.

iii. Identify mentions of candidates and Yes/No supporters

iii.1 Largest Connected Components

Largest Connected Component (LCC) of a graph is the largest possible subset of the original graph with all the vertices connected through a path of any possible length. Real world networks often comprise of multiple disjoint component of nodes. The good majority of the nodes from original graph often forms the Largest Connected Component and hence analysis done on LCC generalizes well for whole of the graph giving the advantage of reduced computational need due to smaller number of nodes and avoidance of complexity due to highly clustered or disjoint graph components. An example is shown in red on Figure 1

iii.2 HITS analysis

HITS (Hyperlink Induced Topic Search) is a algorithm developed by Jon Kleinberg in IBM Almaden which is used to identify important group of nodes within the graph based upon their hubness and authority. Originally developed to represent a network of web pages where an authority is a node with high indegree of link representing its authoritative content while a hub represents a node which work as an index with many out-links to other authority pages. The same philosophy of authority and hub is widely applicable in real

world directed graphs as in social network graphs. An illustration of an authority node (A) is shown on Figure 2, while a hub node (A) is shown on Figure 3.

iii.3 KPP-NEG algorithm

Key Player Problem - Negative (KPP-NEG) is a greedy algorithm used to compute a prestige measure of node being a broker between different component of a graph. A broker is a node that connects multiple segments of clusters of nodes to each other and their removal causes reduction of graph connectivity to greater extent compared to other nodes. Brokers are also called key separators in literature. Figure 4 illustrates a broker node.

iv. Spread of influence

A social network—the graph of relationships and interactions within a group of individuals—plays a fundamental role as a medium for the spread of information, ideas, and influence among its members.[5] In this study, one of our main goals was to measure the spread of Yes/No over the provided network. To achieve this goal we focused on two main algorithms: $K-means$ and LPA (Label Propagation Algorithm).

iv.1 Spread of Yes/No: K-means

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of the K groups based on the features that are provided. Data points are clustered based on feature similarity.[4] The following is a pseudo-description of the K-means implementation we performed:

Initialization step: set K = 2, initialize the K clusters using the previously identified Yes/No seed nodes and randomly choose K nodes in the K initial clusters to be the initial centroids for the graph.

Iterative steps:

1. Associate every node to the centroid whose travel distance to the node is minimum, thus creating K clusters of nodes. The metric chosen to associate a node to a centroid can be understood by thinking that every node wants to belong with the centroid that can reach that node in the fastest way and having a similar neighborhood Yes/No labels with the node.

$$d(c,n) = \frac{1}{cosSim(c,n)} + (1 - lSim(c,n)) \tag{1}$$

   Where:

   - $d(c,n)$: is the distance between the centroid c and the node n
   - $cosSim(c,n)$: is the cosine similarity between the centroid c and the node n
   - $lSim(c,n)$: The percentage of neighbors of the node n having the same label(Yes/No) as the centroid c

2. Now that we have clusters of nodes we have to pick the best node to be the centroid of every cluster. The optimal node is the one that can reach all the other nodes with the lowest travel cost. The travel cost being the sum of the travel distances to every other node in the cluster.

Convergence: After a certain number of iterations, the centroids will not change anymore and the algorithm will have converged. If this does not happen, the algorithm will stop at a given maximum number of iterations.

iv.2 Spread of Yes/No: LPA

Label Propagation is a semi-supervised machine learning algorithm that assigns labels to previously unlabeled data points. At the start of the algorithm, a (generally small) subset of the data points have labels (or classifications). These labels are propagated to the unlabeled

points throughout the course of the algorithm. [6] Label propagation is an algorithm for finding communities. [7]

In comparison with other algorithms (for example with K-means) label propagation has advantages in its running time [8] and amount of a priori information needed about the network structure (no parameter is required to be known beforehand). The disadvantage is that it produces no unique solution, but an aggregate of many solutions.

In this experiment, We adopted two different strategies while using the *LabelPropagationAlgorithm*.

- The first strategy was to modify the original *LPA* in order to be able to evaluate the spread of only Yes/No labels in the network. In particular, the following are the steps we relied on:

    1. **Initialization of the labels**: for each node, we set the label to be -1 if the node is already known to be a *Yes* supporter, -2 if the node is known to be a *No* supporter and a unique integer in the interval [0, +∞[for the remaining nodes.

    2. **Update of the labels**: for each node in the graph, the new label is the most popular (frequent) among all the Yes/No neighbors. if the node does not have Yes/No neighbors, we kept the previous label. This was to ensure the propagation of only Yes/No (-1/-2) labels.

- The second strategy was to run the original *LPA* 10 times and obtaining the final community for a given node by averaging the 10 different outputs. Additionally we illustrated the importance of the Normalized Mutual Information measure (NMI - a common measure of goodness of a community detection algorithm) by computing it over the 10 *LPA* runs.

III. Results and Discussions

i. Temporal Analysis

i.1 Tweet distribution over time

From the 26 of November 2016 to the 6 of December 2016, the number of tweets generated in our stream at any given time in a span of 12 hour ranged in around a million. We observed that there were more tweets during the day and the we had a peak of tweets during voting day at about 1.4 million tweets. This already shows us the importance tweeter has as a social network. Figure 5 shows us the distribution of tweets between our dates of interest at any given time. The next step will be to focus our attention to politics and specifically the Italian referendum of 2016.

For the temporal analysis, we worked with a total of 290 politicians. Those politicians were divided into two camps, the politicians promoting the vote of YES, referred in this paper as the Yes camp and the politicians promoting the vote of No, referred as the No camp. The Yes camp counted 157 members and the no camp counted 133 members. Figure 6 shows the total tweet of each party over time in an interval of 12 hours.

Some interesting fact to note here is the fact that both party tweet approximately the same amount of tweets with the people from the yes camp tweeting slightly more the days leading to the vote. During and after the voting day, we observed an increase in the amount of tweets of the no politicians.

One way we could improve was by including additional politicians basing their vote on their respective a party affiliation, this will therefor increase the amount of data we are working on. In addition to that, we could have added some popular influencers that had a say in the referendum and that worked closely with a particular party. Those include but are not limited to journalist, TV personality, athletes, etc.

i.2 Time series representation of tweets

The 20 most popular terms in our index are: *no*, *solo*, *sempr*, *quand*, *oggi*, *me*, *cosa*, *graz*, *video*, *via*, *ora*, *fa*, *te*, *cos*, *ital*, *renzi*, *fare*, *mai*, *poi*, *giorn*. We could notice see some popular terms used by many politicians such as: *thankyou*, *today*, *do*, *always*. And also the name of a politician: *Renzi*. This suggests that he was getting into a lot of headlines for his name to be mentioned so frequently.

With time series representation of the tweet, we divided each party into 20 clusters, each term was represented by a sax string of 20 characters. The representation of 1 cluster can be seen in Table 1. The rest of them are available within the files attached to this paper.

One trend that we realized tend to happen is that most of the words end up clustered in 2 or 3 clusters, ergo we end up having 2-3 cluster with the bulk of the elements while the rest have very few.

A future change of improvement would be on the kmeans algorithm. Indeed we are using the very basic kmeans algorithm which has its limitation such as those unbalanced clusters. A better version was proposed by Saeid Soheily-Khah. [10] addresses those limitations.

The data and the clusters along with the sax representation of the words are available in the attached code under *ressoures/yesCluster* and *resources/noClusters*.

i.3 Time series co-occurrence graph

Since the clusters were very unbalanced, one thing that we realized is that not all clusters will have LCC or Kcores. Indeed for the No camp about 1/3 of the clusters did not have LCC and KCores whose weight would have been significant for analysis. And for the yes camp about 4/5 of the clusters did not have any significant co-occurrence. Another observation is that the LCC graphs are slightly bigger that the Kcores one in terms of nodes.

The co-occurence full result can be found in the resource folder accompanying this file under *resources/cooc*.

Looking at some of the co-occurrence graphs and their distribution over time with a granularity of 3hours, we realize that most of the words within the kcore and LCC share a some what similar temporal behavior. For example, looking at figure 7, we we can see that there is a somewhat same temporal behavior between Renzi and Graz. We can here assuming that he's been using that word quite often and people have been thanking him as well. We can see as well the word "support" behaving the same way.

We can see as well similar type of behavior for the LCC. In this example we show the LCC of cluster 0 for the Yes camp in figure 8. In this case, we see, as most visible relation, approximately the same temporal behavior between the term analysis and contract. We can easily assume that the were taking about analysing certain contract, probably bills to be passed. Additionally, contract has somewhat the same temporal behavior as the term signature. Ergo this cluster somewhat shows a tendency for the yes camp to talk and push a lot the signature of new bills/contracts.

Additional files are available in the folder accompanying this file under *ressources/pics*. in this folder, it's possible to find all the temporal behavior for every single LCC and Kcore graph for both party. Those temporal behavior may show what each party was focusing on talking about on twitter and may reflect the impact it had on the referendum.

ii. Identify mentions of candidates and Yes/No supporters

ii.1 Identifying tweets mentioning politicians

In this section we analyzed the network of twitter users who have expressed their opinions regarding politicians by mentioning them directly in tweets. We made use of politician index generated in the first section to list the politicians in a combined and grouped csv files. Using the list of politicians, a Lucene query was built that checked the inclusion of any of the politician names in the 'mentioned' field of

the original tweet index. The query structure is outlined as following.

$$mentioned : pol\_1\ OR\ mentioned : pol\_2... \quad (2)$$

All the resulting documents for the query were indexed into new Lucene index, called USER TWEET INDEX. A list of unique user identifier was created from the above query results hence giving us the set of base users (M) and the resulting index of tweets T(M). We uniquely identified 409,043 number of tweets mentioning politicians made by total of 70,684 unique users. The resulting set of users was saved to ***all_users.csv*** file. Table 2 shows a subset of the resulting set of users.

ii.2 Sub-graph Analysis

The first step of the graph analysis involved generating the $\boldsymbol{M}$ user sub-graph from the the root graph originally provided. We leveraged existing java graphing library $\boldsymbol{G}$ to extract sub-graph for set of users $\boldsymbol{M}$ and then further on extracted the largest connected component of the sub-graph. Lastly we ran HITS analysis on that sub-graph to identify 2000 highest ranked (Authority) users (results sorted by score in file all_users_top_authorities.csv). Table 3 shows the top 10 authority users.

ii.3 Partitioning of users in YES/NO group

The task of this section was to classify the user set M into YES and NO vote users taking into consideration the candidates they mention. For the computational convenience we initially built a Lucene index USER POLITICIAN INDEX with the documents comprising of just two fields ***userScreenName*** and ***politicianScreenName***. The document entry was created for each single mention of a politician by a user in the USER TWEET INDEX. Then on for each unique user in set $\boldsymbol{M}$, we counted number of mentions of both ***YES*** and ***NO*** politicians. The tally count for total number of mentions, and mentions of either group of politicians by unique users was generated and saved into ***data/resources/user_tweet_count.csv***. Depending upon the frequency of mentions for YES and NO group, whichever is greater, we classify the user to that group, i.e,

$$if\ n_y > n_n\ then\ u\ \epsilon\ YES \quad (3)$$

$$if\ n_y < n_n\ then\ u\ \epsilon\ NO \quad (4)$$

Using above classification measure, user set $\boldsymbol{M}$ was classified into two groups and HITS analysis was performed onto respective largest connected components of the sub-graphs to obtain 1000 authority users for each group, namely $\boldsymbol{M'}$. The sub-graph extraction and HITS analysis was performed using $\boldsymbol{G}$ library and resulting 1000 authority users for each group was saved into files ***yes_users_top_authorities.csv*** and ***no_users_top_authorities.csv***. Table 4 and Table 5 show respectively the top authority among *Yes* user and *No* users.

The top authority users include *Matteorenzi*, former Italian prime-minister who advocated for the referendum, *Republica.it* news provider and *Sky_tg*24 broadcasting. Due to very high number of follower base, these user accounts certainly commands for high authority within the network. As for NO votes, top authority users are *BeppeGrillo*, co-founder of *FiveStar* Movement, *VirginiaRaggi*, current mayor of Rome and a representative of *FiveStar* Movement and *Mov*5*Stelle*, the official twitter user for the *FiveStar* Movement itself. The movement's stance was against the referendum and hence with high follower base, they stood out as authorities for NO vote.

ii.4 KPP-NEG analysis

Lastly in this section we used KPP-NEG algorithm on the user sub-graph $\boldsymbol{S(M)}$ to identify key players. Because of high computational cost we applied varying threshold value for each node's out-degree centrality to reduce the graph size. The resulting users alongside their scores for each YES and NO group were saved to ***yes_users_500KPP.csv*** and ***no_users_500KPP.csv***. Table 6 and 7

show the KPP-NEG top users respectively for the Yes and No groups.

iii. Spread of influence

iii.1 Spread of Yes/No: K-means

The K-means algorithm major drawback is the running time. This algorithm is indeed computationally quite expensive. In this experiment we faced that issue by trying to cluster the $S(M)$ graph having 34,879 nodes. The algorithm was so slow and was spending 2 hours per iteration. To address this issue, we decided to run the K-means algorithm on a reduced version of the graph by keeping only the nodes with in or out degree greater than 99.

Running the modified K-means algorithm (explained in section iv.1) on the resulting reduced graph, we obtained the the results displayed in Table 8, Table 9 and Table 10 respectively using as seed nodes K-playes, the M set of users and M'. In general, the results showed that the label spreading the must over the network was the *No* label having bigger cluster sizes (Cluster ID = 2). The fact that this result is coherent with the actual output of the 2016 referendum (the No victory) shows us how useful social medias like twitter could be in inferring on real life events.

iii.2 Spread of Yes/No: LPA

The Label Propagation Algorithm (LPA) is far better than the K-means algorithm in terms of computational coast an time. In fact, we didn't have to make any size reduction on the full graph before running it. However, one of the drawbacks of this method is its random component making it possible to get different results in 2 different runs of the algorithm. The 2 strategies we used were (*i*) to run the modified LPA described in section *iv*.2 on the full graph and (*ii*) to run the original LPA 10 times on the S(M) and choose the final community by averaging among the 10 different outputs.

The output of the first strategy can be found on Table 11. And it's also available in the following location data/resources/Spread_Of_Influence_Output/Modified_LPA/Modifed_LPA_Output_K_Players.csv. The results show that the party spreading the most over the full network is *Yes* with 427854 elements against 914 for the *No*.

The output (the top 30 communities in terms of number of elements) of the second strategy can be found on Figure 10. And it's also available in data/resources/LPA_Final_Output.csv.

We implemented the NMI measure and used it to evaluate the correlation among the 10 LPA runs. Figure 9 shows a heat-map of the NMI over the 10 LPA runs. The causality in the NMI values shows that the 10 LPA runs are independent due to the random component of the LPA. We can also observe that the values are around 0.8. this suggests that even tho the algorithm is casual it still yields similar results among runs.

References

[1] Crimson Hexagon The Fundamentals of Social Media Analytics: http://www.upa.it/static/upload/the/the-fundamentals-of-social-media-analytics.pdf

[2] Umar Ruhi (2014), "Social Media Analytics as a Business Intelligence Practice: Current Landscape Future Prospects", Journal of Internet Social Networking Virtual Communities, Vol. 2014 (2014), Article ID 920553, DOI: 10.5171/2014.920553

[3] Lu, Y., Wang, F., Maciejewski, R. (January 01, 2014). Business intelligence from social media: a study from the VAST Box Office Challenge. IEEE Computer Graphics and Applications, 34, 5.)

[4] Andrea Trevino, Introduction to K-means Clustering; https://www.datascience.com/blog/k-means-clustering.

[5] David Kempe, Jon Kleinberg, Éva Tardos, Published April 22, 2015 Maximizing the Spread of Influence through a Social Network

[6] Zhu, Xiaojin, Learning From Labeled and Unlabeled Data With Label Propagation.

[7] U.N.Raghavan – R. Albert – S. Kumara, Near linear time algorithm to detect community structures in large-scale networks.

[8] M. E. J. Newman, Detecting community structure in networks.

[9] Jessica Lin Eamonn Keogh Stefano Lonardi Bill Chiu A Symbolic Representation of Time Series, with Implications for Streaming Algorithms.

[10] Saeid Soheily-Khah. Generalized k-means based clustering for temporal data under time warp. Artificial Intelligence [cs.AI].

[11] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, Alex Thomo K-Core Decomposition of Large Networks on a Single PC.

[12] Allan Bickle The k-Cores of a Graph.

[13] Massimo Franceschet Network Connected Components.

IV. Appendices

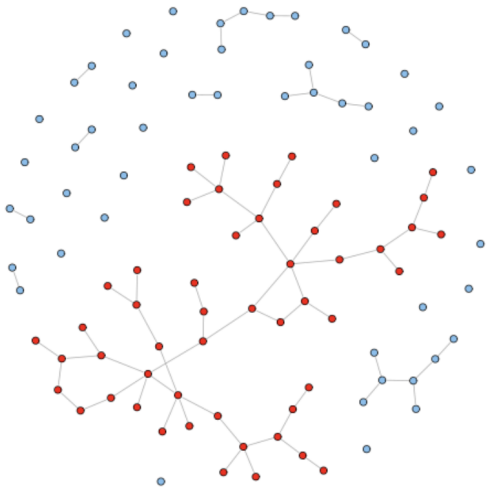

Figure 1: *Largest connected component in a graph.*

Term	SAX Representation.
risultat	ejojeeeeeoejeeerjrtoe
bilanc	eeetrjoootojjeeeeeeje
punto	ffrrfftffrrrfffllfff
social	tfotiiloosdilffddfddd
battagl	ffjfffjftjtrffrffrfff
tropp	jjjdtpjtdtjdjjddjtdjd
staser	jjptjftfjfjjmfffjffff
megl	pjjpjjjjdjtttdddjjddd
via	idcoilsiottllcidodocc
senator	rejjoetoeerjteeeeejee
propr	jjddjjdojjjdtdootdjjd
smontat	gggggkrogkgktgggggggg
pubblic	ejelljlptpeneejeejgje
civil	ftffrrrrflfrfffffflff
mille	eeeejtjtjejjjeeestee
camer	jjjjtrrtejjeeejeeeeee
comun	jjjjjppddtjtddddjjjjd
parlamentar	tjjjjjjdjpjttjddddjjd

Table 1: *Sample of SAX representation for Cluster 1 of No Politicians*

ID	Screen Name
803590719432695808	mr_ranadeus
524372178	nicoladeredita
1513455144	davide_bedini
237816428	vitochiariello

Table 2: ***M*** *set of users.*

ID	Screen Name	Score
18762875	matteorenzi	0.141567
18935802	repubblicait	0.128491
5893702	SkyTG24	0.123204
150725695	Agenzia_Ansa	0.114537
15072569	$\text{Agenzia}_{A}nsa$	0.114537
395218906	Corriere	0.112361
14060262	RaiNews	0.110574
29416653	LaStampa	0.109715
420351046	sole24ore	0.107311
19067940	$\text{beppe}_{g}rillo$	0.101121
963938472	$\text{Palazzo}_{C}higi$	0.099910

Table 3: *Top 10 authority users for* ***M****.*

ID	Screen Name	Score
18762875	matteorenzi	0.182002
18935802	repubblicait	0.160015
5893702	SkyTG24	0.139088
150725695	Agenzia_Ansa	0.135655
963938472	Palazzo_Chigi	0.135041

Table 4: *Top authority from YES users.*

ID	Screen Name	Score
19067940	beppe_grillo	0.157597
1530798872	virginiaraggi	0.133566
289400495	Mov5Stelle	0.122574
376218450	AndreaScanzi	0.120515

Table 5: *Top authority from NO users.*

ID	Screen Name	Score
1386632492	miaudit	5358.0
437337751	Esercito	4287.0
2226802248	SquilibrataS	4286.0
16402819	andreadelogu	3214.0
18935802	repubblicait	3214.0

Table 6: *KPP-NEG top users for YES group.*

ID	Screen Name	Score
434839840	lauracesaretti1	2552.0
2651374166	ScifoGiuseppe	2552.0
16476536	ultimenotizie	1912.0
376218450	AndreaScanzi	1912.0
2455915345	tiz_mic	1912.0

Table 7: *KPP-NEG top users for NO group.*

Cluster ID	Number of elements
1	1446
2	1375

Table 8: *Modified K-means results - K players seeds.*

Cluster ID	Number of elements
1	1385
2	1452

Table 9: *Modified K-means results - M seeds.*

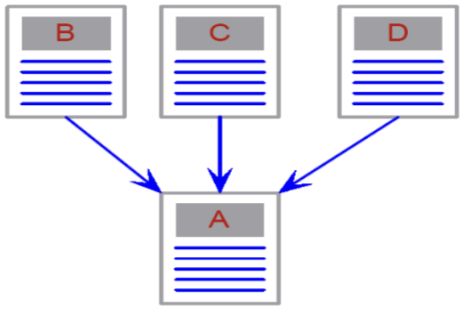

Figure 2: *An authority node with high in-degree centrality.*

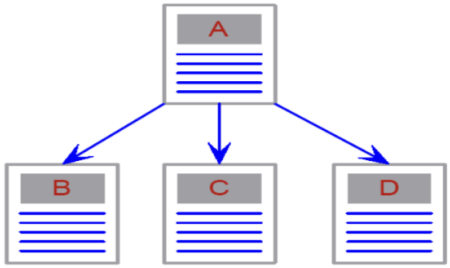

Figure 3: *A hub node with high out-degree centrality.*

Cluster ID	Number of elements
1	1418
2	1404

Table 10: *Modified K-means results - M′ seeds*

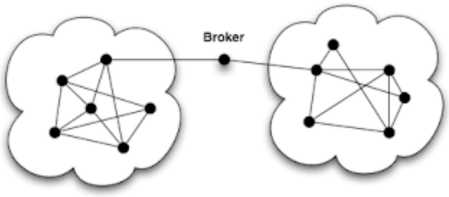

Figure 4: *A broker node as key separator between two clusters.*

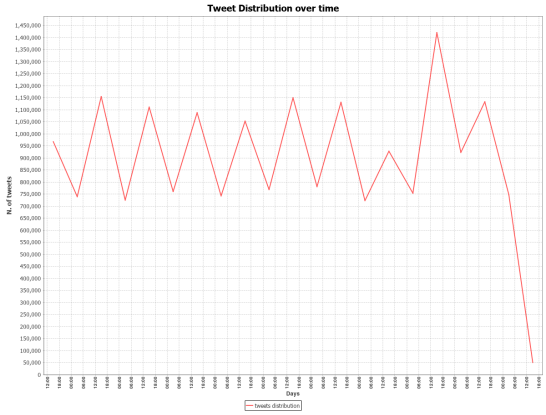

Figure 5: *Tweet distribution over time for all tweets*

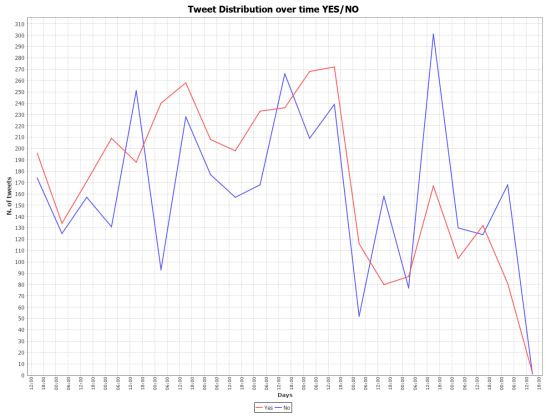

Figure 6: *Tweet distribution over time for YES/NO Politicians*

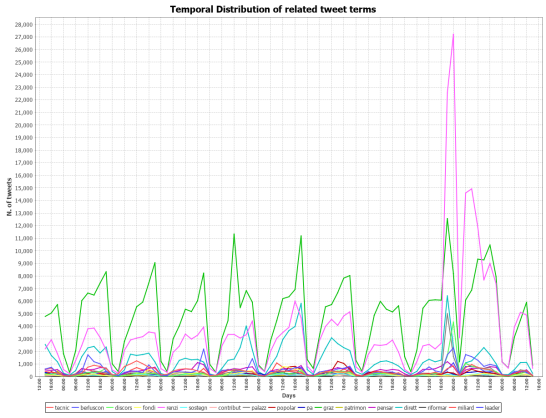

Figure 7: *Kcore temporal representation for Cluster 0 No camp*

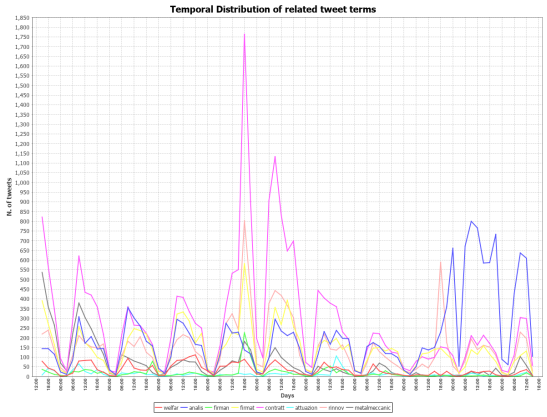

Figure 8: *LCC temporal representation for Cluster 0 YES camp*

Community Label	Number of elements
YES	427854
NO	914
UNKNOWN	21426

Table 11: *Modified LPA results - K players seeds*

Community Label	Number of elements
YES	427854
NO	914
UNKNOWN	21426

Table 12: *LPA 10 runs final results*

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
Run 1	1	0.84	0.84	0.84	0.83	0.81	0.88	0.84	0.86	0.863
Run 2	0.84	1	0.82	0.82	0.81	0.79	0.85	0.83	0.84	0.836
Run 3	0.84	0.82	1	0.85	0.82	0.8	0.87	0.83	0.84	0.84
Run 4	0.84	0.82	0.85	1	0.83	0.8	0.87	0.84	0.84	0.85
Run 5	0.83	0.81	0.82	0.83	1	0.79	0.84	0.82	0.81	0.831
Run 6	0.81	0.79	0.8	0.8	0.79	1	0.82	0.81	0.82	0.818
Run 7	0.88	0.85	0.87	0.87	0.84	0.82	1	0.87	0.89	0.875
Run 8	0.84	0.83	0.83	0.84	0.82	0.81	0.87	1	0.84	0.851
Run 9	0.86	0.84	0.84	0.84	0.81	0.82	0.89	0.84	1	0.848
Run 10	0.86	0.84	0.84	0.85	0.83	0.82	0.87	0.85	0.85	1

Figure 9: *NMI heatmap over 10 runs*

Community ID	Number of elements
19	33719
21	62
18	59
22	58
20	52
17	42
16	36
25	25
24	24
23	24
26	17
27	15
28	15
36	8
15	8
30	7
38	6
31	6
44	6
34	6
37	6
39	6
69	5
35	5
33	5
29	5
71	5
68	5
54	4

Figure 10: *The top 30 communities in terms of number of elements*