Project 2018: Italian Referendum

Davide Aureli 1651051

Valerio Guarrasi 1643257

Federico Siciliano 1604124

General Notes

Most of of our functions are implemented in the file **Load_tweets.java**.

In **Algorithms.java**: we put all of the algorithms implemented by us.

The plot implementation can be found in **Plots_SBN.py**.

To make the project work, you must have in the data directory all of the folders initialized, the stream directory, the two .json files, the graph file and the plotici2.csv file.

Temporal Analysis

1) To create our dataset we used 3 websites.

- https://social-media-expert.net/2017/12/elenco-profili-twitter-dei-deputati-italiani-xvii-legislatura/
- https://social-media-expert.net/2017/12/elenco-profili-twitter-dei-senatori-italiani/
- http://www.rainews.it/referendum/votazioni-in-parlamento.html

Scraping.java: The first two websites gave us the twitter account screen names of the deputies and senators that use twitter. From these websites we extracted the info with web scraping. Instead the third one gives us the last vote that each deputy and senator expressed on the Referendum. Here we got the data from the two files: **Votazione_5.json**, **Votazione_6.json**. To make one unique dataset we matched the two sources by using the last name as id. We created a file called **politici.csv** formed by 4 columns:

- Last Name
- First Name
- Twitter Account
- Vote (Y/N): if a politician decided to not vote, we classified him as an N. We decided to use this strategy since we saw that many politicians in the last vote

decided not to express their opinion and because abstaining from voting in this Referendum is like voting No.

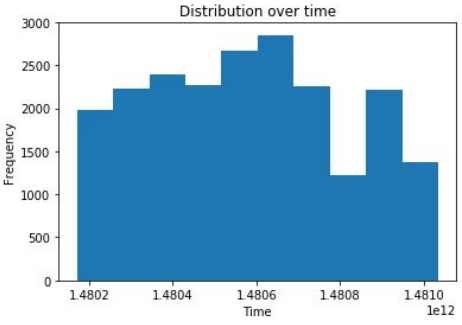
Since these websites aren't perfectly made (and some politicians have the same last name), we had to do some manual modifications to make our data consistent. We saved our modified version in **politici2.csv**.
We get 693 politicians (394 for Yes and 299 for No).

FilterTweets0.java: Rereading the csv file and the stream of tweets for each folder(day), we created two Lucene Indexes (one for the accounts that voted Yes and one for the accounts that voted No), by checking for each tweet if there is a match on the screenname. Each tweet has these fields:

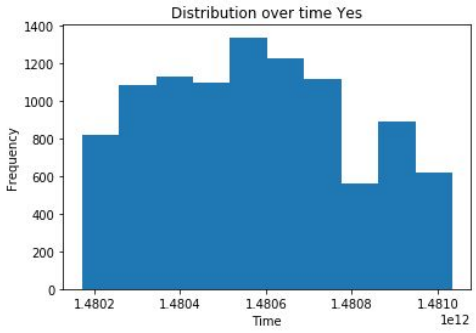
- screeenname: String Field - the screenname of the user
- id: LongField - id of the tweet
- createdAt: Long Field - the epochs time in which the tweet was published
- text: TextField - parsed text of the tweet
- vote: String Field - N if the user voted No, Y if the user voted Yes

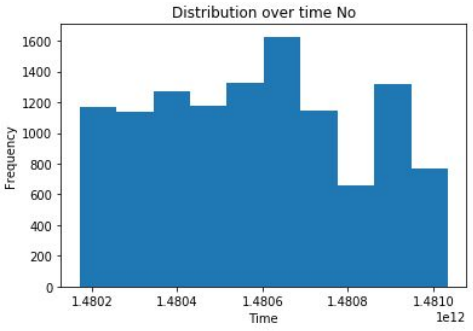
We saved this index in the folders **/tweets_politici_Y** and **/tweets_politici_N**. From the screen outputs, we can notice that some twitter accounts tend to tweet posts one after the other.

Stat_0.java: Here we answer the requested questions combining the two datasets. The total of number accounts that are both in our list and in the stream are 538 (303 for Yes and 235 for No). We have 77% of the Yes party and 79% of the No party. Instead the number of total tweets recovered are 21484 (9879 for Yes and 11605 for No). To see the distribution over time of the tweets we used python. The distributions are saved in the files **distribution_Y.csv** and **distribution_N.csv.**



We can see that there is increase in the frequency of posting in the last days before the Referendum, the day of the Referendum the number of tweets decreases noticeably and right after it there is a new peak. After that there is an ulterior drop because the topic isn't hot anymore. Observing the two parties separately, we can notice that the trend is very similar but the peak for the Yes party is 3 days before the Referendum and for the No party 2 days before.





2) **Sax_create.java**: For each group (Yes and No), we found the top 1000 words in frequency. We can see that they are almost all relative to the Referendum. So, we decided to remove some terms that are not relevant to this topic (4 words: https, rt, t.co, t.c).

The top-5 words are:

Yes

1. bastaunsi
2. iovotosi
3. si
4. referendum
5. renzi

No

1. no
2. iovotono
3. renzi
4. referendum
5. riforma

We queried the tweets for each term, so that we can get the normalized time series of when they were created (12 h grain) and then transform them in a Sax string using an alphabet of 15 letters(to extract more details). We saved these maps in **saxN.csv** and **saxY.csv**.

The first algorithm that we used is K-Means between the Sax strings of the time-series of the top 1000 frequent words for each group. We used as distance between the strings the MinDist. We used k = 2, so we get four subgroups: ***t_1(Y), t_2(Y), t_1(N) , t_2(N)***. To use SAX efficiently, we created a table with the precomputed distances between each character. The clusters are saved in the folder **cluster_words/** divided in a folder for the group Y and the other for the group N.

3) **Co_occurence_graph.java**: For each group of token in ***t_i(Y)*** and in ***t_j(N),*** we load the files that we created and build the co-occurrence graph of them (two word ***t_1 , t_2*** have an edge **e**, if they appear in the same document; the weight ***w*** of **e** is equal to the number of documents where both tokens appear); To get these information we used a query AND to find the tweets that are in common between two

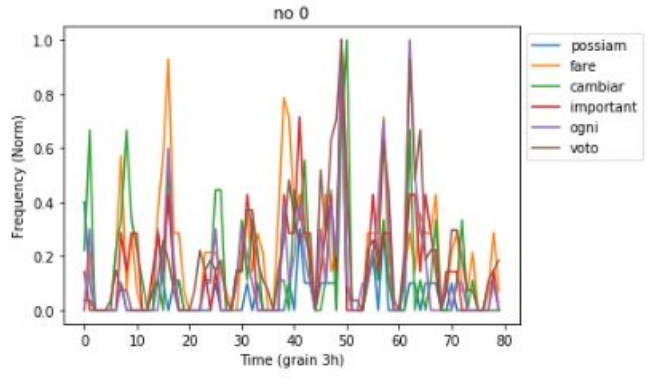
words. As threshold of the edges weights we used 15, in this way the output terms of the next part, are all words relative to the referendum. After a few tries, our best technique to extract these terms is:

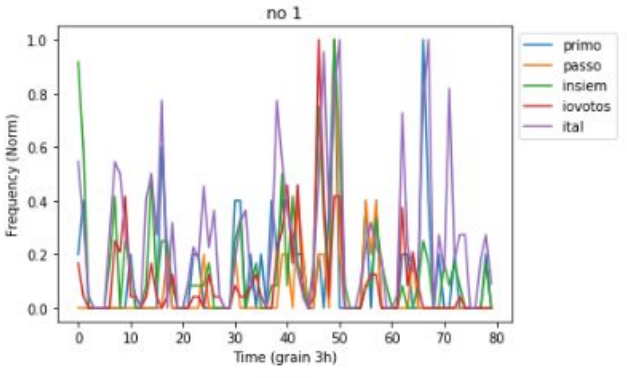
- identify the biggest Connected Components ***CC*** (we can see that the smaller ones are groups of other topics)
- From this we extract the innermost core (***K-Core***) producing a subset of terms for each cluster: ***t_1'(Y), t_2'(Y), t_1'(N) , t_2'(N)***.

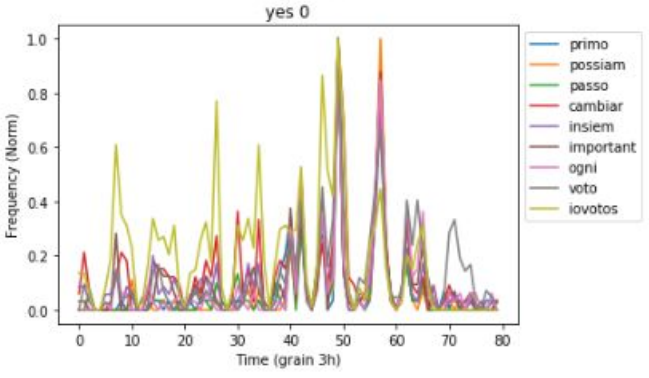
We decided not to consider all the Connected components since the only one with words related to the topic was the biggest one. We would have expected this, since the tweets that are analyzed are all from the week of the referendum and written by politicians. The combination of these two algorithms gives us the most important words that represent the Referendum, we can see them in the folder **/cores_words** in the directory of the Yes group and in the No group. We even put the related time-series with grain 3h for the next point.

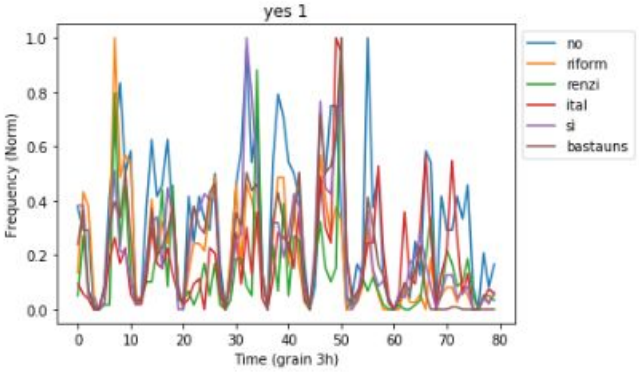
We decided to use 15 as threshold for the subgraph, because if we have chosen a too small value we would have also considered words that aren't relevant, instead if we would have chosen a too big of a threshold we would have gotten only the most frequent words. So the best way to get the most relevant words is to get a connected component of medium size and then get from it the innermost core nodes.

4) For comparing the time-series of each group (Y/N) we used python. Here we show for each group (Y/N) and for each cluster the time-series of each word extracted from the previous point:

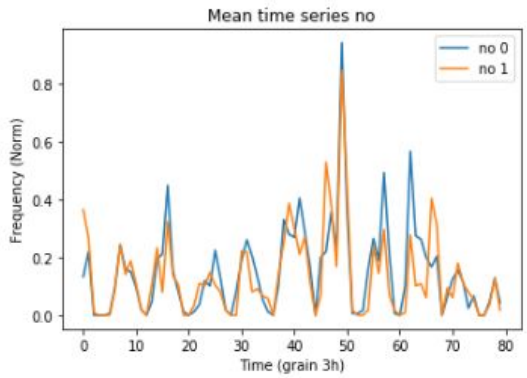


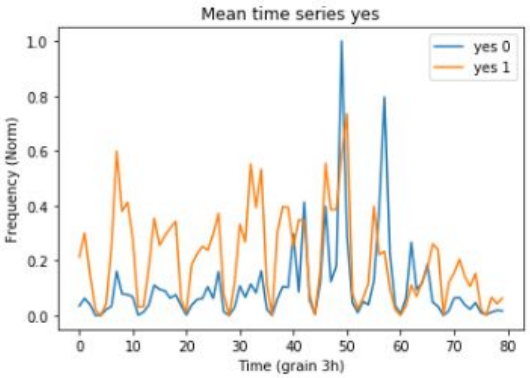






To check out the trend of each cluster, we computed the mean of the time-series of the selected words for each cluster of each group.





We can see that each cluster in each group have similar trend, in particular between the clusters in group No. We can see that during the days close to the Referendum the frequency reaches the maximum. We can notice that, they day after the Referendum the two parties have a different reaction: the Yes party, since they lost, decided to tweet on the topic. In particular we can see that the yes party, maybe thanks to some prior investigation, decided to change strategy and in the days close to the Referendum posted more on the matter. We can also observe with this grain that during the night, barely nobody tweets, in particular these words. We can also see that not all of the words are consistent with the party they are part of; by checking the tweets we saw that a lot of users of the No party decided to be sarcastic on the lost of the Yes.

Identify mentions of candidates or YES/NO supporter

1) **FilterTweets1.java**:From the entire tweets dataset, to create our set of users **M** we reiterated on all the tweets:

- if the tweet is from a politician we extract the tweet, and the user.
    - The politician gets a score of 25 for each tweet he posts
- if the tweet is not from a politician we compute the score of Y/N party.
    - if the text of the tweet has one of the official #hashtags (Y/N), for each hashtag the user gets a score of 2 on the relative party score(Y/N).
    - if the text of the tweet mentions one of the politicians, like before the party score is upgraded by 3
    - to be sure that these users are really talking about the Referendum, we decided to keep the user only if his total score (combination Y/N) is higher than 6.

We decided not to use the ***t_i'(Y)***, ***t_j'(N)*** groups of words, since they are too common words and the extracted people may not be interested in the Referendum. Of course, the party that the user will vote, for us, will be the one with the highest score. In case of a draw, we decide to give as default No. We saved our set of users M in the file **M_score.csv**, where we have the user screenname, user id and the score for each party (Y/N). Like we did before, we save these tweets in a Lucene Index with the same fields as before but this time we also put the id of the user, so that we can do a match with the given graph later. This index is saved in the folder **/tweets_correlati**.

Stat1.java: like before we answer a few questions. We get a total of 3293 users with 27928 tweets.

2) **Create_subgraph.java**: by reading the graph given to us, we create the induced subgraph relative to the selected profiles, by choosing the edges between them. We saved our subgraph in the file **subgraph.gz**. To know, to which node a profile corresponds to, we save on **node_mapper.csv** the node number with his user id.

3-4) **HITS_Graph.java**: From this subgraph, we found the biggest connected component and on that we applied the HITS. We are interested in Authorities score, you can find the results in the folder **/TopAuth**, where there are 2 files **TopAuthYes.csv** and **TopAuthNo.csv** where we have the userid, score(Y/N) of the party he votes for and the Authorities score. To re-rank these two files we combined these two measures by simply multiplying them. In this way the resulting score takes into account how much a user is active on the topic of the referendum and how much this user is an influencer in the Twitter graph. We saved these results in **TopCombYes.csv** and **TopCombNo.csv.** On this same connected component we applied KPP-NEG algorithm and as before we got the top-k both for Y/N. The results

are saved in the same folder in the files **TopKYes.csv** and **TopKNo.csv**. We can see that the top ranked users for this measure are both politicians and not. These users that aren't polititians are influencers in the political sphere, since they are active with a lot of political posts.

Top Users for Authorities:

Yes

1. angelerrimo (not politician)
2. PuglisiPD (politician)
3. paolocaccamo68 (not politician)

No

1. AntonellaGramig (not politician)
2. 57_mimmo (not politician)
3. 2ndDemocracy (not politician)

Top Users for Combined:

Yes

1. s_margiotta (politician) (2nd in M_score)
2. antgentile (politician) (1st in M_score)
3. erealacci (politician) (7th inM_score)

No

1. renatapolverini (politician) (4th in M_score)
2. MagdaCulotta (politician) (5th in M_score)
3. renatobrunetta (politician) (1st in M_score)

Top Users for KPP-NEG:

Yes

1. flavagno (politician)
2. AnnaAbagnale5 (not politician)
3. SteGiannini (politician)

No

1. iovotono (political page)
2. AugustoMinzolin (politician)
3. Mov5Stelle (political page)

Spread of Influence

1) **KMeans_seeds.java**: We created modified version of the K-Means algorithm that works with S(M). Every node(user) has as information its relative row of the adjacency matrix. The membership function is combination of the Cosine-Similarity on the adjacency matrix row and the strength of the vote. We computed as new centroids a real node of the graph, by choosing the node with minimum distance from it. In this way we can consider the information on which party he is on, the two nodes on which we are calculating the distance get a distance equal to 0.5*(distance between the adjacency)+ 0.5*(distance between votes). Distance between votes = Abs(vote_1 - vote_2)/2, where vote_i = (M_score_Y-M_scoreN) / (M_score_Y+M_scoreN). So 1 if the user is fully Yes, -1 if he is fully No and positive if his idea is closer to Yes and negative if it closer to No. The algorithm is initialised with two clusters (K=2) containing all the SEED nodes identified (for every party). We do this with three different Seeds:

- Using k-Players ***K***
- Using ***M***
- Using only the ***M'***

The results are shown in the folder **/clusters**. We can see that the number of iterations is really low, thanks to the seed initialization. We can see that we get for each initialization 2 clusters and the correspondence with our ground truth is pretty consistent: the users that voted yes are almost all in the same cluster, same thing for the users that voted no. The implementation of this modified k-means is in **Algorithms.java**. We can notice for all the implementations that the final centroids aren't politicians but normal people:

- PallePanzane (vote = 0.83) → Yes party
- QualeFuturo (vote = -0.76) → No party

The score Y/N of these users is uncertain in respect of the ones of the politicians which are more convinced on their choice.

Addendum

LPA.java: our modification of the LPA algorithm simply consisted in modifying the initialization: instead of assigning a different label to each node, we assign to the Y K-players the label 1 and for the N K-players the label 0. The modification of this algorithm can be found in **ModComunityLPA.java**.We implemented the algorithm ten times, so for each node we get ten different label propagation results. Our function to decide which of these labels should be the winning one is simply by getting the most common label between the ten results. The assigned label is shown in **LPA_labels.csv**. Finally we calculated the NMI Normalized Mutual Information measure between the 10 LPA runs, we can observe the resulting matrix saved in **NMI_matrix.csv**. We can see that each clustering results are around 70% or more similar to each other. So we can conclude that almost 30% or less of the twitter population doesn't have a clear political side because of their lack of connectivity, while around the 6-10% of the population fall in the middle of the graph and doesn't have a clear political side because they are connected to both Yes and No users.