Introduction:

This report encompasses social and behavioral analysis about the Constitutional Referendum of Italy that took place on 4th of December 2016. The referendum proposed new constitutional laws that could change the power of the Parliament of Italy. The voters of the state had to vote either Yes or No to indicate whether they favor the referendum or not. The main source of data for this analysis has been from twitter, where we analyzed tweets of Italian Politicians and some Italian users, who were very important influencers for either of the Yes or No opinions.

We also had access to a dataset containing all the tweets across 4th of December, worth of 10 Gigs of compressed tweets. So, the volume of dataset was enormous. This dataset was used to collect the tweets of the list of politicians we collected. We were also provided a graph of sampled network having nodes as twitter ids and edges between nodes representing if two nodes are related or not.

This project involves the fundamentals of Temporal Analysis and Graph analysis to find the most important Politicians and Users. The whole project has been implemented in Java and the libraries used were Twitter4j, Maven, Lucene, SAX, G Stilo Library.

Temporal Analysis:

Theoretical background:

Temporal statistical analysis enables you to examine and model the behavior of a variable in a data set over time (e.g., to determine whether and how concentrations are changing over time.). In our case we do temporal analysis to analyze the change and spread of opinion about the referendum over the period of 10 days through the tweets data and list of politicians as a starting point.

Analysis:

1.1) I started the analysis from the list of politicians or journalists provided in the senato.csv and camera.csv files, merged the people mentioned in both these files for ease of access and analysis. I did some data cleaning to filter out the people who doesn't have complete information like twitter id, party supported etc.., From the filtered list I basically did a segregation of Yes and No politicians based on the party they are supporting, specifically if a politician is supporting one of AP, DS-CD, PD, SCPI, AL-A, AUT he/she is categorized as a YES politician, if he/she is supporting one of FI-PDL, FDI, LEGA, M5S, SI-SEL, COR, GAL, MISTO he is categorized as a NO politician. Once we get the list of YES and NO politicians we separate out the tweets of these people from the main tweet indexes and create 2 types of indexes holding YES and NO politician tweets.

1.2) We then analyze the words mentioned in these 2 tweets sets and extract top 1000 words (both from the YES and NO tweets) based on their frequency. We also build a SAX string for every term with grain = 12h for doing temporal analysis. After this, we cluster these words using k-means algorithm I put together to do a clustering of the words which expose similar temporal behavior. This clustering is done on the 2 sets of top 1000 words from YES and NO tweets. I used elbow method to obtain optimal cluster count for our analysis.

1.3) For every cluster obtained in the previous step we built a co-occurrence graph based on the number of documents(tweets) a pair of terms occur together and how similar are their SAX strings (temporal occurrence) divided by respective term frequency which gives us 2 scores, we consider the maximum of these 2 scores as edge weight between these 2 words or terms in the graph. We build a co-occurrence graphs like this for each cluster of top-1000 YES or NO terms (or words) that we got in the previous step. We then find the connected component and k-core for each of these graphs to obtain sub-set of words for each cluster of words from the previous step. so, we focus our attention towards the words that really matter for our analysis of referendum. so, we defined a score function to determine the edge weights in the graphs that we created for the co-occurrence graph

With this we conclude the temporal analysis of the available tweet data superimposed with the list of politicians or journalists that we started with. Below are some results that I obtained in this analysis (in the results section)

Code analysis:

We used the following files to create indexes and plots,
For plots I made use of Python and code is present in **plot.py** file.

TweetIndexManager.java (extended from base abstract class IndexManager.java) and helper classes TweetIndexBuilder.java (extended from base abstract class IndexBuilder.java) to actually build the tweet indexes. - this is for building all the tweet related indexes like core tweets stream, yes tweets and no tweets.

Similarly, we used PoliticianIndexManager.java and PoliticianIndexBuilder.java to manage YES or NO politician indexes for ease of access. This sort of hierarchical code division aided me in clear separation of concerns and better code reusability and code maintenance for further edits.

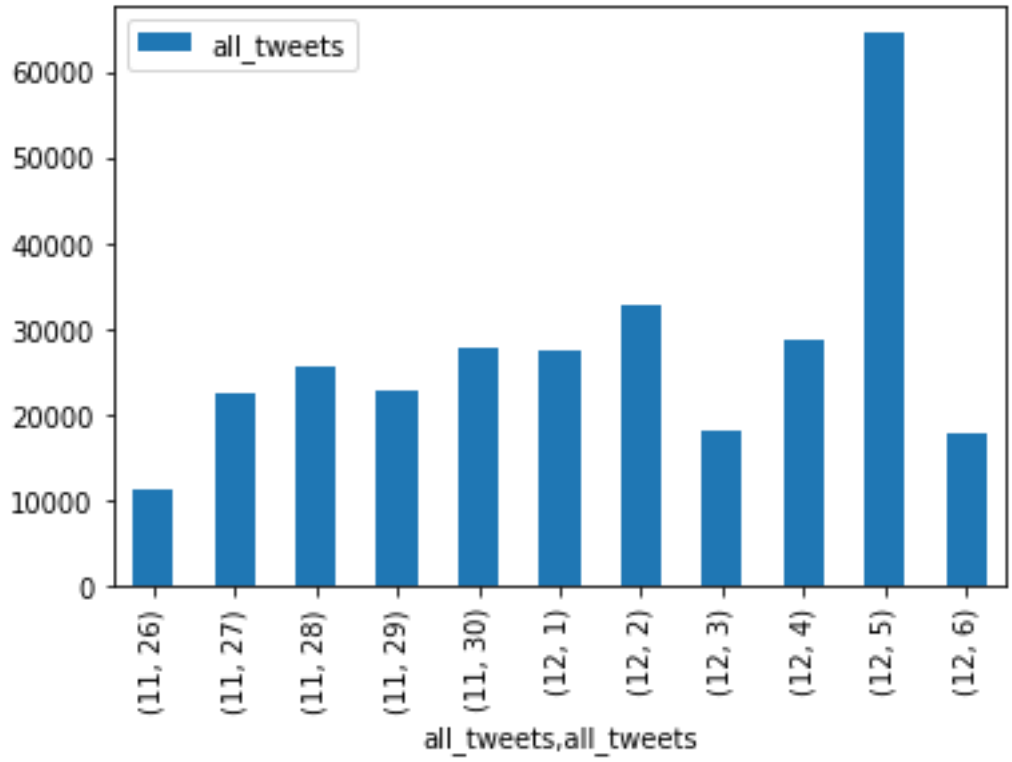
ClusterGraph.java which makes use of quite a few libraries from G library is used to build co-occurrence graphs from the clustered term sets. ClusterGraphFactory.java is a mediator class to manage group of Cluster graphs easily without any code duplication.

TweetTerm.java is used to hold the SAX and binary representation for the frequent terms from the tweet indexes for temporal analysis and better code separation and object manipulation

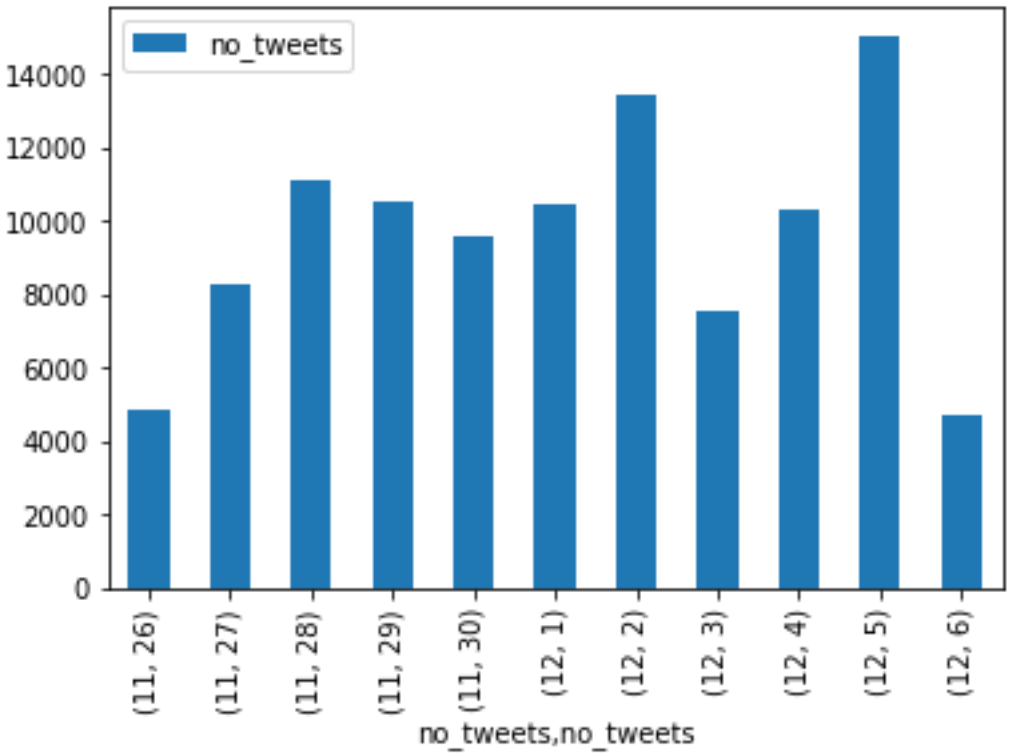
Results:

1.1) Initially based on the twitter stream data (from 26/11/2016 to 6/12/2016) that was provided, I created indexes for each tweet using Lucene. In total **18850000** tweets were indexed. Following information is stored for every tweet index, userId, date, name, screenName(this is the twitter id which is unique for every user), tweetText, hashtags, mentioned, followers, rtScreenName, rtUserId(this holds re-tweet information). After this, as mentioned in the analysis section we did some data cleaning to put together list of politicians (and journalists) for our analysis of referendum, we got **299** YES politicians and **256** NO politicians. Based on the politician's screen names (twitter ids) we queried the original tweet data set and we got **289229** matching tweets out of which **188373** are from YES politicians and remaining **100856** are from NO politicians. Initially I started with considering only the actual tweets tweeted by either YES/NO politicians, but it yielded me very small sample of **20000** tweets, so I considered the retweet data as well by applying a OR Boolean query on Lucene which kind of increased my tweet sample by **10** times. I used python to visualize the results of tweets over the period of 10 days (from 26/11/2016 to 6/12/2016)

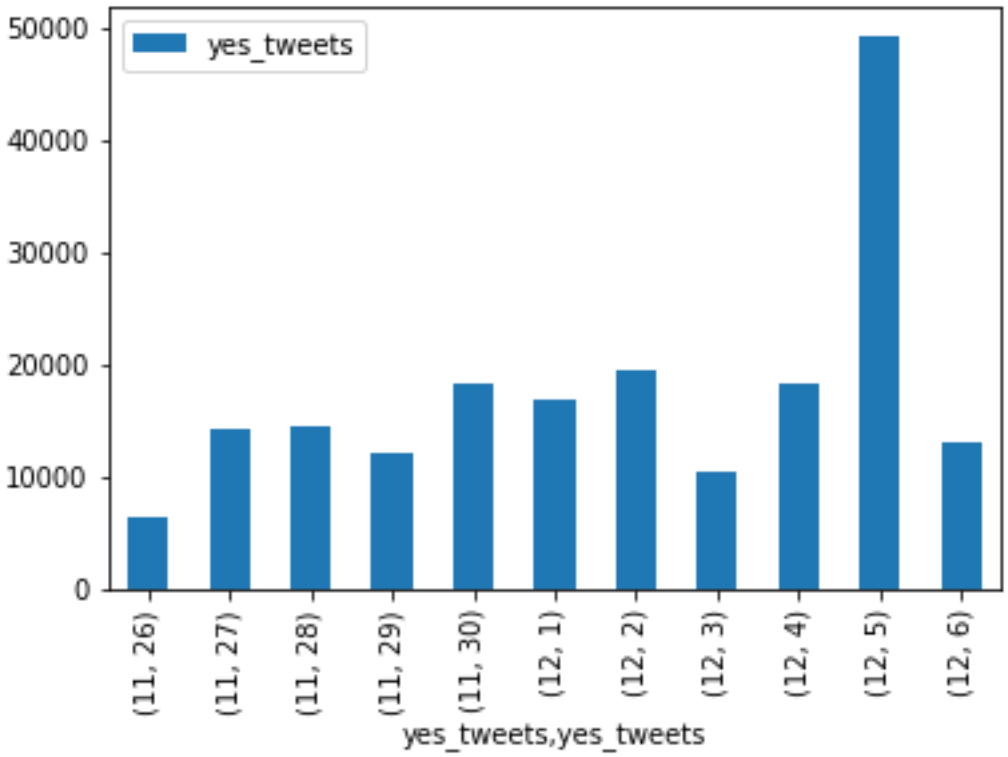
ALL TWEETS



NO TWEETS:



YES TWEETS:



Note: Time stamps of both YES and NO tweets are updated in yes_tt.csv and no_tt.csv files inside the output folder.

1.2) We analyzed these **188373** tweets from YES/NO politicians and extracted top 1000 words from both YES and NO tweets (text is extracted from 2 fields, tweetText and hashtags) based on their frequency. Then from every word of these 2000 thousand words we generated a SAX string based on its frequency over the 10 days period under consideration with the grain value of 12 hours so as to know the temporal behavior of each word over the period. We use this SAX representation to cluster similar words using trivial k-means algorithm. In other words, we group words in to clusters based on their temporal behavior. Top 1000 YES and NO words and their respective frequencies can be found in **output/yesWords.json** and **output/noWords.json**.

Note: I ensured that there are no STOP words in the top 1000 words from both the sets by the comparing each term with stop words updated in **input/stopwords.txt**

Below are the word clouds generated for YES and NO words –

YES words word cloud -

NO words word cloud -

For k-means, with the elbow method we arrived at **clusters=10** to be the optimal cluster terms for our list of words. We considered number of **epochs/iterations to be 1000** for the clustering so if the centroid calculation reaches saturation then the algorithm terminates without any further centroid readjustment.

1.3) From the clusters of words (20 clusters, 10 each for YES and NO words) obtained in the previous step, we built co-occurrence graph for each cluster of words and generated the graph through a threshold from a score function we put together.

The score function goes like this,

```
double div1 = intersection / uFreq;
double div2 = intersection / vFreq;

// Get the max of them
float maxRelFreq = max((float) div1, (float) div2);

// If this quantity is higher than a threshold add the edge between the nodes
if (maxRelFreq > 0.0001) {
    g.add(i, j, 1);
}
```

So, if the score of a pair of words is greater than 0.001 we add the edge to the co-occurrence graph otherwise we skip the connection between 2 terms.

Once the co-occurrence graphs are built for every cluster then we generate Connected Components and k-core for every cluster to extract the sub-set of words which are highly connected (and there by very related) for every cluster so as to deep dive in to our analysis on the words that really matter for the referendum analysis.

Results for Connected Components of all the clusters can be found in:
output/relComps.json (with separate arrays for clusters from YES and NO words)
output/ccWords.json - contains all the subset of YES and NO words from CC components

similarly, we updated the results of k-core in **output/relCores.json** and **output/coreWords.json**

Note: After all the analysis of YES and NO supporter and Spread of influence we found that K-Core yields better results in terms of number of hubs and authorities created for our specific analysis than Connected components results.

Identify mentions of candidates or YES/NO supporter:

Analysis:

2.1) Initially to create the supporters indexes I created a new Lucene directory with each index having fields such as name, id, yesPolsMentioned, noPolsMentioned, yesConstructionsUsed, noConstructionsUsed, yesExpressionsUsed, noExpressionsUsed, isAYesPol, isANoPol, vote. So, a lot of metadata has been added to each supporter index document for ease of analysis and correlation. Initially we added all the YES and NO politicians from the previous step as supporters by updating their corresponding meta data (isAYesPol and isANoPol).

For deciding whether the current user is a YES/NO supporter we used different level of information by iterating through all the available tweet stream. To start with we used information in the mentioned attribute of every tweet, so if a YES politician is mentioned then yesPolsMentioned is updated with 1 else if NO politician is mentioned then noPolsMentioned is updated with 1 for that particular index. If both are mentioned, then both the fields are updated to 1. Then we made use of the Top 1000 terms (both YES and NO) extracted from the previous step and accordingly update the yesConstructionsUsed and noConstructionsUsed fields.

Next, we made use of pre-defined YES and NO expressions to further segregate YES and NO supporters list.

Below are the YES and NO expressions we used,

YES Expressions:

#iovotosi, # iovotosì, #iodicosi, #iodicosì, #iohovotatosi, #iohovotatosì, #votasi, #votosi, #votosì, #bastaunsi, #bufaledelno, #bufaladelno, #si, #sì

NO Expressions:

#iovotono, #iodicono, #iohovotatono, #votano, #votono, #bastaunno, #bufaledelsi, #bufaledelsì, #no, #noivotiamono, #ragionidelno, # unitixilno, #votiamono

Then we implemented a score function to update the vote of each supporter as mentioned in the following pseudo code,

```
 float yesScore = (float) (supporter.getYesPolsMentioned()
                                + 0.5 * supporter.getYesCostructionsUsed()
                                + 3 * supporter.getYesExpressionsUsed() );

 float noScore = (float) (supporter.getNoPolsMentioned()
                               + 0.5 *supporter.getNoCostructionsUsed()
                               + 3 * supporter.getNoExpressionsUsed());
// If the sum of the score is at least 8
if (yesScore + noScore > 8) {
    finalScore  = yesScore / noScore;
} else {
   finalScore  = 1;
}

if (finalScore > 1.45) {
       vote = "yes";
} else if (finalScore < 0.7) {
       vote = "no";
} else {
       vote = "-";
}
```

2.2) In this step, Using the provided Graph and the library G, we generated the subgraph induced by users S(M) from the previous step (here users = supporters created before). Then we found the largest connected component CC and computed HITS on this subgraph of M users.

2.3) By performing HITS on the subgraph created on the previous step, we could compute the authority and hub scores of each node sorted in descending order. Now we basically extract the node ids from the graph and get their support (YES/NO) already computed in the step 2.1 (using the score function). We extract top 1000 (for both YES and NO supporters) Authorities and HUBS from the HITS result using functions provided in G library for Hubs and Authorities (using HubnessAuthority algorithm). All the results are mentioned in the Results sub section below.

Code analysis:

The workflow for Supporter Index creation is more or less same as both Tweets and Politician indexes with some additional utilities added for score functions and other requirements.
SupporterIndexManager.java (extended from base abstract class **IndexManager.java**) and helper classes **SupporterIndexBuilder.java** (extended from base abstract class IndexBuilder.java) to actually build the supporter indexes. All the supporter indexes have format mentioned in **Supporter.java**, we use this class to manipulate supporter indexes once we retrieve them.

Results:

2.1) From the analysis we did, we obtained **84237** users with **5944505** tweets since we considered both tweets and retweets data for building original tweets data set.

Out of the **84237** supporters,
502 - from YES and NO politicians list
45884 - from mentioned field in the tweet stream
31569 - from YES and NO expressions mentioned above
6372 - From list of words from the k-core algorithm

2.2) Provided graph file has been correlated with above **84237** users and the resultant subgraph has been saved to **output/ccsg.txt** file.

We could obtain this in the created subgraph,
Number of YES supporters: **11549**
Number of NO supporters: **18985**

2.3) We then computed HITS on the above generated sub-graph. Below are the results we obtained,

YES AUTHORITIES: **1000**
NO AUTHORITIES: **1000**
UNCLASSIFIED AUTHORITIES: **1586**

YES HUBS: **500**
NO HUBS: **500**

Above results are saved to following files,

output/yesSup.txt – all the list of supporters or users who voted YES
output/noSup.txt – all the list of supporters or users who voted NO
output/authorities.txt – contains all the authority scores of all the supporters (84237**)**
output/yesAuthorities.txt – contains top 1000 YES users sorted by authority scores
output/noAuthorities.txt – contains top 1000 NO users sorted by authority scores
output/unclassifiedAuthorities.txt – contains the authority scores of unclassified supporters
output/hubs.txt - contains all the hub scores of all the supporters (84237)
output/yesHubs.txt – contains top 1000 YES users sorted by hub scores
output/noHubs.txt - contains top 1000 NO users sorted by hub scores

Spread of Influence:

Analysis:

To do the analysis for spread of influence over the period of 10 days we created a wrapper for Label propagation algorithm which not only does the clustering of all the nodes of the graph provided but also adds a YES or NO label on top of the clustering, so we can analyze how the spread of influence about the opinion (YES or NO) on the referendum throughout the network in a nutshell. This analysis kind of gives us an additional view point so that we can be very clear about the actual opinion of the supporters and all the nodes in the network in one place.

3.1) For the list of YES and NO supporters we gathered in the previous step (from files output/yesSup.txt and output/noSup.txt) we ran a customized version of LPA we prepared and while initializing the labels for the algorithm we pushed 1 for YES support and 2 for NO support for a corresponding graph node from the yesSup.txt and noSup.txt files so that we can easily differentiate the YES and NO influence of all the labels in the graph (using the label 1 and 2 with which we initialized the graph).

3.2) We did a similar analysis for the nodes in output/yesAuthorities.txt and output/noAuthorities.txt as well as output/yesHubs.txt and output/noHubs.txt so that we can have a holistic analysis of spread of influence for both authorities and hubs there by removing any inconsistencies in the intermediate analysis.

Code analysis:

We created a customized version of Label propagation algorithm to account for additional labels on the individual nodes. The code is present in the CommunityLPA.java. We initialize all the nodes in the graph with zero and then add labels to the corresponding supporters extracted from the files mentioned above (yesSup.txt, noSup.txt, yesHubs.txt, noHubs.txt, yesAuthorities.txt, noAuthorities.txt)

Results:

We applied LPA algorithm to 3 sets of data, YES/NO supporters, YES/NO authorities, YES/NO hubs. Below are the results obtained,

SUPPORTERS (M):
YES: **22406**, NO: **370405**, UNCLASSIFIED: **57383**

HUBS (M'):
YES: **4321**, NO: **387367**, UNCLASSIFIED: **58506**

AUTHORITIES (M''):
YES: **155734**, NO: **236805**, UNCLASSIFIED: **57655**

As we can see from the above results, there is clear spread of influence of NO supporters across the graph for all the supporters, authorities and hubs data sets.

Summary and Conclusion:

To summarize the analysis that was performed, we started with **299** YES politicians and **256** NO politicians. We extracted **18850000** tweets from the tweet stream dataset (from 26/11/2016 to 05/12/2016) and based on the YES and NO politicians screen names we created a subset of **289229** tweets with **188373** tweets from YES politicians (lets refer them as YES tweets) and **100856** from NO politicians (lets refer them as NO tweets). We then extracted top 1000 words each from YES tweets and NO tweets based on the tweet frequency and temporal behavior of the words over the period of 10 days. We then applied trivial k-means to these 2 sets of top 1000 YES and NO words obtaining 20 different clusters of words (since we got optimal clustering count as 10 from the elbow method). Now we created Cluster graphs to each of these clusters and calculated both Connected component and K-means to get subset of words from each of these 20 clusters. So, we tried to take in to account only the terms which are highly correlated temporally.

We then tried to find all the users using different criteria like whether they mention any of YES or NO politician in their tweets, whether they use any predefined expressions that clearly signifies their support, do their tweets have any of the words from the clusters that we obtained in the previous step. Considering all these metadata we created a score function to obtain the support (YES/NO) of a particular user. In total we obtained **84237** supporters with **11549** YES supporters and **18985** NO supporters which clearly signifies that there is more influence for NO. We then applied HITS algorithm on the list of supporters to get the top 1000 YES and NO authorities and top 1000 YES and NO hubs (here Authorities and Hubs are just normal supporters but with more influence).

To try to analyze the spread of influence, we applied a variant of Label propagation algorithm which is more or less a customized version of k-means with consideration for labels of individual nodes for clustering or categorization.

To conclude, even if we started with more YES politician samples and there by more YES politician tweet samples, the support for NO is clearly visible with the supporters list and their corresponding spread of influence of opinion throughout the network