Date: 6th September 2018

SBN Project

Italian Referendum V2

Siddhant Tandon- 1771650

Sibghat Ullah – 1772576

Introduction

We present a work of 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. A total of 510 Politicians were collected , mostly by scraping from the Open Parlamento website.

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 , Jsoup, Lucene , SAX , G Stilo Library.

Temporal Analysis

1. Gathering Data

The names of the politicians were scraped from the Open Parliamento website and some of them were manually added later. Then these names were divided into Yes Opinions and No opinions based on some prior belief, the political parties they belonged to, and some website pages mentioning their vote.

Then, the corresponding twitterIDs of these politicians were collected using Twitter4j API. The relevant twitter account was selected by keeping a constraint on number of followers. We kept a threshold of 1000 minimum followers. In the end we got a total of 510 Politicians with their twitter IDs, and after dividing them into Yes and No groups the number was 278 for the Yes group and 232 for the No group.

After getting a definite list of twitter IDs of these politicians, the next step was to obtain their tweets from the given Dataset of twitter stream. To do this, we built a Lucene Inverted Index in which each document had the following fields : **Date, Twitter profile ID , Name, Screen name , Tweet, Retweet Screen name , Retweet User name , Retweet Twitter Profile ID.** All through the analysis mostly Date, Profile ID, Name and Tweet fields were used. Also we kept three separate Lucene inverted indexes Yes Index directory containing only the documents related to Yes Politicians , No Index directory containing documents related to No Politicians and finally one complete Index directory on the whole dataset. We did this because at some points we needed to specifically index only the Yes or No directories.

Total number of Yes tweets found were **68341.** Total number of No tweets found were **97356.** Even though the number of No users 46 less than the Yes users, total number of no tweets were far greater.

Next , to obtain the distribution of tweets we just counted how many tweets were done by both the groups everyday from 27th November to 6th December. Following is the graph obtained :

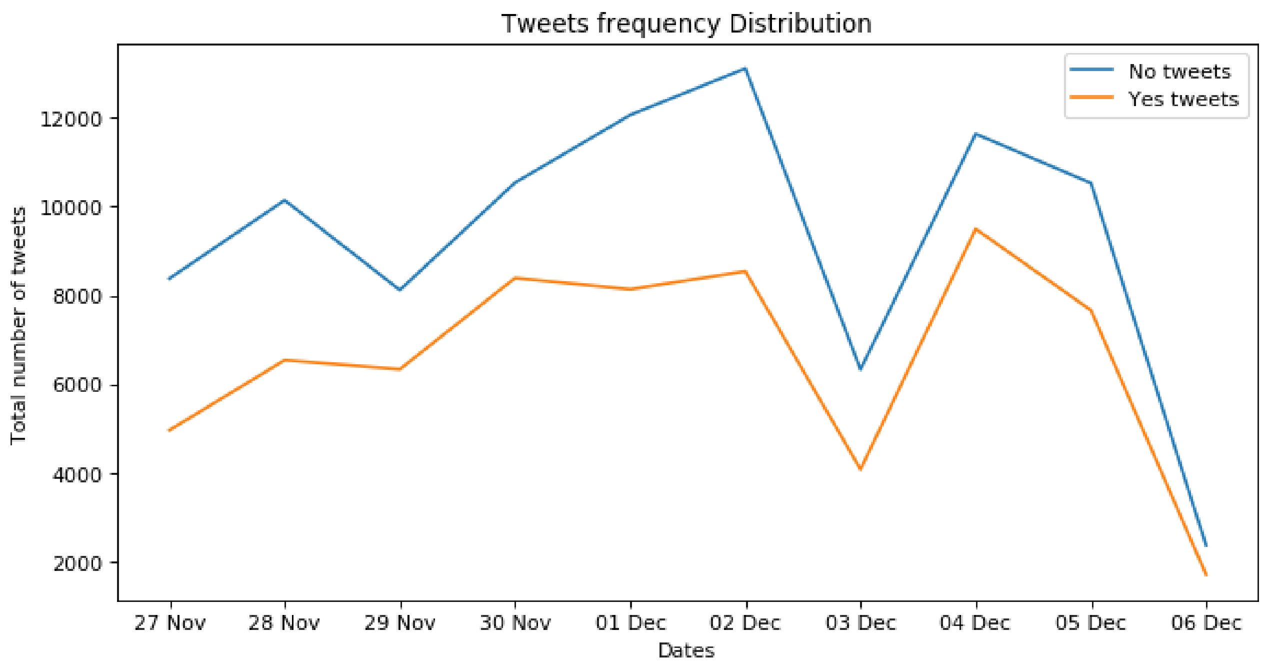

Fig 1. Figure showing distribution of tweets across 10 days.

2. Clustering SAX Strings

In the previous step we had created two separate directories to index both Yes and No documents separately. For both these directories we collected the 1000 most frequent terms and built a timeseries by counting the occurrences of these words in the tweets every 12 hours period starting from 27th November 3:27 am. Then for all the timeseries of these frequencies, SAX strings were built.

After this we built a class to implement K-Means algorithm in order to cluster the SAX strings obtained. To implement the K- Means we followed the following pseudocode :

```
1. Initialize Random centroids.
2. Iterate until not Converged :
    a. Iter ++
    b. Assign labels for Current Centroids.
    c. Calculate centroids again and update current centroids
    d. Old centroids = Current Centroids.
```

Explanation :

1. Random Centroids means we randomly assign any of the SAX string to be the cluster centroid.
2. Convergence criteria was such that we measured the RMS distance between the old and current centroids, if this value was less than a threshold of 0.001 we stop the iteration.
3. To assign the labels , the SAX distance between a member (SAX string for a particular Term) , and every centroid was calculated. Finally it was added to the group with minimum distance.

   This distance between two SAX strings was computed using the library function for SAX.

   new SAXProcessor().strDistance(sax.toCharArray(), center.toCharArray()).

4. Then after label assignment, new Centroids were calculated by taking the average of the cluster.

   The average of two SAX strings are defined as average of their corresponding Numeric value. Ex : "aaaa" Numerical value : 10,10,10,10

   "bbbb" Numerical value : 11,11,11,11

   Average : 10+11/2 , 10+11/2,10+11/2,10+11/2

   Then the equivalent string was found by using Character.forDigit() method.

The total number of clusters were decided using the **Elbow Method** which is such that, K-Means is run for several values of total clusters ***k***, and for every run the Sum of Squared errors is calculated between the cluster element and its centroid. Finally this value is summed over all the clusters present. We found that SSE is very high for $0< k < 5$ then it decreases with increase in k and again starts increasing around 10. That is where the elbow is located hence the total number of clusters were 10. We have compiled the results in the text file **yescluster.txt & nocluster.txt.**

3. Building Co-occurrence graph

To build a Co-occurrence graph for Yes group, we had to see if two terms of a cluster occur together in a document or not. To do this we implemented the following solution.

For each of the clusters present in a group (Yes or No group) , all the cluster members were put in a set. For all the terms of this set, we calculated if a pair of terms occur together in a document or not. This was done by building the MUST Boolean query like this :

```
BooleanQuery booleanQuery = new BooleanQuery();

booleanQuery.add(queryterm, BooleanClause.Occur.MUST);

booleanQuery.add(queryterm2, BooleanClause.Occur.MUST);
```

Finally the total number of documents having these two terms together , was assigned as the edge weight between these two terms occurring together. This process was repeated for all the 10 clusters. The results were saved in files **yesOccgraph.txt & noOccgraph.txt.**

These are some results from the Yes-Co occurrence graph.

```
leader;matteorenz;12;0        difficil;lira;112;0        giampaologall;populism;13;0

giampaologall;matteorenz;7;0        bella;pdnetwork;27;0        testimon;matteorenz;49;0
```

Largest Connected Component and K-Core

Next, we built the Largest Connected Component and K-Core graphs by using the functions from the G library.

To find the largest connected component we applied this method. For each of the co-occurrence graph, we built a weighted Undirected graph with normalized edge weights. Then we kept a threshold of **0.05** to keep only those edges which had some weight. This would have removed the terms which were occurring together in less number of documents. This number was decided by finding the minimum edge weight and average edge weight in the whole graph. Then we just randomly chose a value between these two values. We used this resulting graph to extract both the Largest Connected Component and K-Core. The files were saved as **yes_cc_005.txt, yes_kc_005.txt, no_cc_005.txt , no_kc_005.txt**.

The only significant difference in the results we could spot was the size of both these graphs obtained. For example the total number of edges in No Connected Components are 4158 and for the No-K core the number is 184, which is significantly smaller. And this makes sense because K-core is one of the connected components found out by deleting vertices with degree less than K.

The following were some results from the No-K core about Matteo Salvini :

matteosalvinim	vincenzofolin	0.5231692677070828	2
matteosalvinim	mater	0.42262222222222223	2
matteosalvinim	ricord	0.3301204819277108	2
matteosalvinim	rispond	0.21642619311875694	2
matteosalvinim	domenic	0.49645390070921985	2

And following from Yes Connected Component :

bella	matteorenz	0.16535433070866143	0
bella	pdnetwork	0.06725380304243395	0

The third column represents the edge weight and the 4th represent the cluster id. We think that the words for Matteo Renzi a, Pd network are positively expressed in Yes Connected Components.

4. Time Series Comparison

To compare timeseries for the both Yes and No groups of both K-core and LCC, we built a timeseries for each of the words in a graph by counting the frequency in a period of 3 hours.

This was done for all the four graphs obtained for every cluster present. The timeseries were stored in the files yes_cc_0,yes_cc_1….., and yes_kc_0,yes_kc_1 … similarly for the no groups.

We also plotted the timeseries for the top 4 terms in a cluster ordered by edge weight. Following are some plots obtained :

Timeseries Plot for words in 0th Cluster of No-Kcore

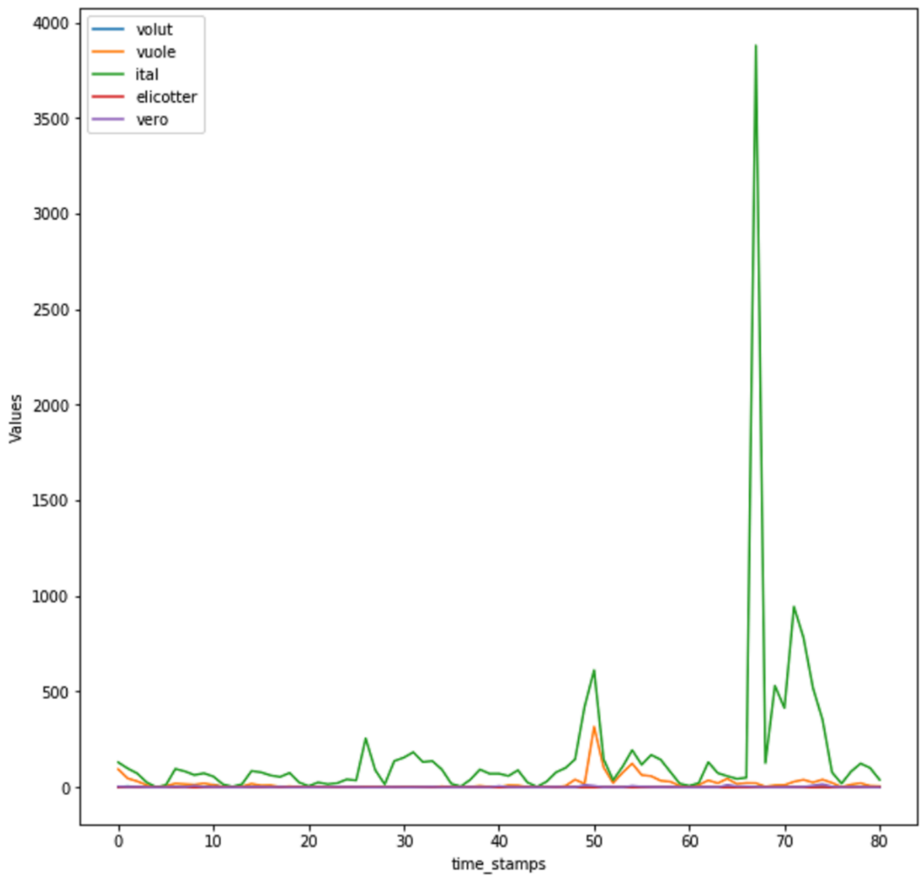

Fig2. TimeSeries comparison for the group No-Kcore cluster 0. The x axis represent the dates starting from 3:29 PM of 27th December and having a period of 3 hours.

We do observe some similarities in the timeseries pattern for the words”ital” and “vuole” which sort of explains both of them belonging to the same cluster.

Timeseries Plot for words in 0th Cluster of Yes-Kcore

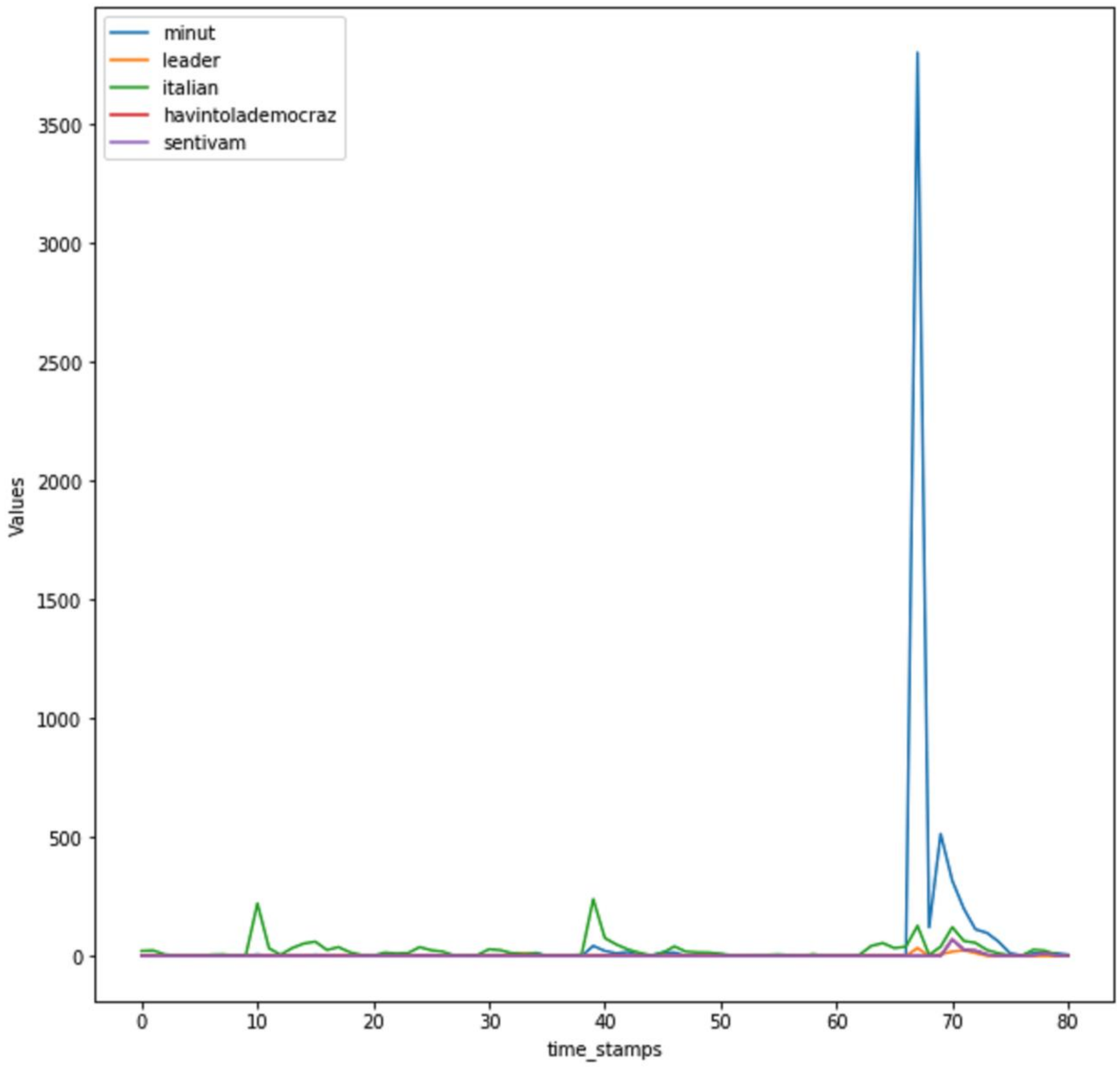

Fig2. TimeSeries comparison for the group Yes-Kcore cluster 0. The x axis represent the dates starting from 3:29 PM of 27th December and having a period of 3 hours.

Similarly in this plot “minut”,”Italian” and “sentiviam” have quite common patterns across the days.

Identifying Mentions of candidates or Yes/No supporter

1. Identifying tweets of users that mention politicians who support Yes or No

In this task we retrieve those user IDs who have mentioned a politician who could support Yes group or No group. Our intuition was that we could use the set of words occurring in the Yes-Kcore graph and the Yes Politicians list, by building a AND query between them. Similarly doing this for the No group. Following is the Boolean Query we came up with :

Word Set : [Set of all terms occurring in K-Core graph analysis]

Politician List : [Set of all Politician twitterids/ Names which we collected]

To identify Yes Tweets :

(Politician1 OR Politician2 OR Politician2….Politician N) AND (Term1 OR Term2 OR Term3…..Term N)

After obtaining the resulting documents from the above query we stored the list of user names, their user IDs , tweets, frequency (the number of times a user appears throughout the documents). Specifically we use one file yes_idtoname.txt/no_idtoname.txt mostly in the subsequent steps and it has twitterID to name Mappings.

These are some stats obtained :

No of Yes users : 17570

No of No users : 25798

No of yes user tweets :45256

No of no user tweets : 127520

Total tweets : 172776

2. Finding highest ranked (Authority Users)

After loading the given graph , we extracted the subgraph of nodes which are the M users found in the previous step. From the resulting graph the largest connected component was extracted. Then we computed Authority scores on this resulting graph. The file was saved as **ranked_authorities.txt**. The 5 of the top 1000 ranked (Authority) users are as follows :

TwitterID	Authority Score
292417838;	0.03458809527082806
2278958635;	0.03407098661371533
517934877;	0.03401182496050117
201412456;	0.03285402801266916
1412500075;	0.03200352072099691
749248856;	0.031894622932535525
1318108844;	0.03087635121626473

These scores were obtained on the complete list of M users found in the previous step which had both Yes and No users included. So we just traversed the whole list of ranked authorities and added the ranked authority to Yes group if the authority was in Yes group of M users. Similarly for No group. In this way we obtained the following stats :

Top Authorities present in yes groups: **247**

Top Authorities present in no groups: **323**

Top Authorities who mentioned both yes and no groups: **430**

Following are top 5 authorities for each of the groups :

Yes group:

martinorosario2 gioicaro FranceCasini PaoloScilinguo PietroSalvatori ALisimberti gennaromigliore siperilsud1

No group:

Babylonboss leonardo1ao luigigiordano87 KikaKahshan OliverMcPic mazzanti_giu ilcorvo20 Ambrogio2001

Users who mentioned both Yes and No groups

GASPARECARLINI 57_mimmo AurelioToro Didi1648 soniabetz1 Giangiagainst antonio_ansp danielecina

It turns out that these users have very high number of tweets and moderate amount of followers. For example the user **gioicaro** has 23,000 tweets and 4000 followers. Similarly another user from No group **Babylonboss** has 64000 tweets.

3. Identifying highly Central Users

From the previously obtained M users, those users were selected that had a frequency of appearing in the document greater than 2. Then we followed the same initial steps as in the previous question, that is loading the graph, selecting subgraph, and extracting the largest connected component. The nodes on which the subgraph was extracted were highly frequent users.

On the resulting graph we computed both the Hubness and Authority scores. Our next task was to define a combined measure to select the top user IDs. This is essentially a Top-K retrieval problem. Hubness and Authority are two different metrics which has given us a ranked list of user IDs. Of course a Naïve solution to this problem would have been just combining both the Hubness and Authority scores arithmetically by taking the average of both scores. But this metric does not really reflect the true information. Hence to solve this problem we implemented **Fagin's Algorithm.**

Fagin's Algorithm :

0. Sort in descending order both the lists of scores (i.e. Hubness and Authority scores) .
1. Do sorted access in parallel to each of the m sorted lists Li.
   a. Stop when there are at least k objects, each of which have been seen in all the lists. In our case k means 500, that is top 500 elements.
2. For each object R that has been seen: •
   a. Retrieve all of its list scores by random access.
   b. Compute F(R) = F(x1,...,xm) i.e. the aggregate of all the scores.

3. Return the top k answers.

In this way we obtained a combined metric of Hubness and Authority scores that give us the top ranked users. Again , because this was run on all the set of users, they were divided into Yes and No groups just by traversing the whole list of highly frequent **M** users. The final authorities are in text files **yestop500.txt** and **notop500.txt.**

Following are some users from the Yes and No groups:

Yes group	No Group
Name Score	Name Score
ricdil1 0.13629732055785815	claudio301065:0.09937161137266498
MarcoFer1975 0.1288077387042009	serebellardinel:0.09900222315411415
Francetomm 0.1154788706511519	carlucci_cc:0.09814902258600677
antonio_bordin 0.11438870508235993	sevenseasmarina:0.09551960361758255
civati 0.1123782449216246	SCUOLATUTTIUNIT:0.09360801837302468
	laltraguanciama:0.0929817241721128
	AngeloTofalo:0.09091900936612612

4. Identifying top 500-K users using KPP-NEG

The first step to use this algorithm was to reduce our graph size as this is very expensive to compute. We selected the subgraph by extracting the nodes of the **M** group of users obtained in part 1.1. Then again we extracted the largest connected component graph. This graph was further pruned by selecting only high degree nodes, i.e. the nodes with neighbors more than 10.

This task was computationally expensive and often we faced out of memory error. Then we had to explicitly mention how much heapspace we need. This can be done in Netbeans in the following way :

1. Right click your Project and open Properties.
2. Click on Run option and enter the following in the VM options text box : -Xmx4g

   Here 4g means we are demanding 4 Gigs of Heap space from java compiler.

After working around this, the results were obtained from the KPP-NEG and saved them in the file **yeskpptop500.txt and nokpptop500.txt.** Of course there were some overlapping results because in our original M users that we obtained from the query , they were not entirely disjoint sets. Hence some intersection in the subsequent results was expected.

Yes Group Kpp top ranked users :	**No group of Kpp users:**
GabrieleMuccino:6580.0	**501874833:s_margiotta:6580.0**
AngeloTofalo:6580.0	**1594040222:marobe997:4386.0**
michelecmassari:6580.0	**2456846400:VEN_VIT52:4386.0**
ErbaSelvatica:6580.0	**3405599800:5StelleOvunque:4386.0**
VBenetello:6576.0	**589418956:giuslit:4386.0**

Spread of Influence

After understanding the Label Propagation algorithm , we modified the algorithm by changing the **initList()** function in the G library. Specifically we made a copy of the ComunityLPA.java file and changed the initList() function by initializing the nodes of the graph which occur in either of the groups by a number. For Yes nodes they were assigned the number 1 and , for no group the number -1, and for the other Undecided nodes we kept the number 0.

We also passed the set of Yes and No nodes from outside through the **communityLPA.compute()** method into the constructor of the class. This way the value of Yes and No nodes were fixed. We also had to pass the LongIntDict LID into the constructor so that we could write the set of labels to disk. As asked in the exercise we ran Label Propagation for top KPP players, top M players and top M' players.

We assigned the Yes and No nodes as the users obtained for the top K players using KPP-NEG. Before running LPA, we extracted the largest connected component of the graph and ran Label propagation on this resulting graph. We did this because we believe that since labels are spread through edges , it is better to just run this on the largest connected component rather than running on the complete graph which might have huge number of small disconnected groups, and running LPA on it would just increase the computational cost rather than generating some important information. We did this for all the set of players.

Following were the results obtained for LPA on top K – Kpp Neg players :

Initial size of Yes Community : 500

Initial size of No Community : 500

Initial size of Unknown community : 16999000

Final size of Yes Community : 71

Final size of No Community : 707

Final size of Unknown community : 16999222

In 8 iterations , LPA converged for given set of nodes.

Remark : The size of unknown community is so high because initially you have to give a big value to read the Given graph. Even though you further extract the subgraph, the resulting size is the same as the one you gave initially. So if we look at the net movement of labels, we could say No labels propagated quite much than the Yes labels. The net amount of labels that went to the unknown group were 222.

The results are stored in the files **yes_comm_kpp.txt , no_comm_kpp.txt and unknown_comm_kpp.txt.**

Following were the results obtained for LPA on M' top players :

Initial size of Yes Community : 500

Initial size of No Community : 500

Initial size of Unknown community : 16999000

Final size of Yes Community : 68

Final size of No Community : 681

Final size of Unknown community : 16999251

It took 12 iterations for LPA to converge.

Again the reason for a high value of Unknown community is same as described above.

The results are stored in the files **yes_comm_MM.txt , no_comm_MM.txt and unknown_comm_MM.txt.**

We faced quite some problems when we were running LPA on the **M** set of users probably because the size of Yes and No top M players was quite high. Thus the resulting largest connected component graph was a big graph too.

The size of the Yes top M players were 17,557 and for No it was 25,774 . We ran it several times but it was just taking a lot of time. However we wrote a function that would write, at every 10th iteration , the Yes , No and Unknown community members to disk. The algorithm was stopped around 200th Iteration.

Before doing all of this, we faced another very unusual error. In the Compute() function, the execution of the program was stuck at **CountDownlatch latch.await()** function. The control was not exiting out of this statement. Probably something was messed up with the CountDownlatch as it has to count down the latch to exit from waiting. So we found there was an argument that could be passed to the await() function to end waiting time hence we modified this line to **latch.await(30, TimeUnit.SECONDS)** and it worked.

Following were the results obtained for the set of M players :

Initial size of Yes Community : 17,557

Initial size of No Community : 25,774

Initial size of Unknown community : 16956669

Final size of Yes Community : 1565

Final size of No Community : 19808

Final size of Unknown community : 16978627

The results are stored in the files **yes_comm_M_iter200.txt and no_comm_M_iter200.txt.**

It can be seen from the text files that many of the No labels have been assigned to the unknown community. Overall it can be commented that the final size of No labels has always been bigger than the Yes labels. This hypothesis supports the results of the referendum as No voters won the referendum.