Predicting Political Orientation with Twitter

Group E:
Stefano Campese, Davide Ghiotto, Darko Ivanovksi,
Diego Mazzalovo, Alessandro Mazzoni

20 January 2020

Contents

Chapter 1

Abstract

Nowadays information that a person shares with their social networks profile can tell a lot about that person, and knowing and managing those pieces of information can be such a huge source of interest and income.
In this project, we aim to collect data from Twitter profiles of a user and understand their political orientation by studying those data.
The choice to use Twitter as a social network (instead of using other socials like Facebook and Instagram that are maybe even more popular than Twitter in Italy) is just because in other social networks the contents are private, while in Twitter all the contents are public and there isn't any privacy problem.
At the very beginning, the point is to differentiate between predictors and outcomes:

- predictors (or independent variables) are the data that allow us to predict a certain topic. In our case, predictors will be all written contents (tweets and retweets) of a certain Twitter profile;

- outcomes (or dependent variables) are the results of the previsions made on the base of predictors. In our case, outcomes will be the political orientation (right or left) of a person related to a certain Twitter profile.

The first step is to collect data, in particular, to collect Twitter profiles, and from each profile obtain the political orientation they expressed. In this way, we have a solid base from where we can start our analysis.
Once the data collection is ended, the second step is to analyze the data we have collected, building a machine learning algorithm with the capability to express if a certain user has a right or left political thought.
All the Python scripts written for this work are in the final Appendix.

Chapter 2

Phase 1: Data Collection

To perform a good data collection, the idea is to distinguish between three different methods, trying in this way to avoid the limitations that are typical of each method. The three methods (or tasks) are the following:

1. The first task has to collect data from an online questionnaire. In particular, the questionnaire is composed both by direct and indirect questions;

2. The second task has to collect data directly from Twitter accounts. In particular, task 2 has to analyze Twitter profiles randomly chosen from the follower of the most popular five parties in Italy (Partito Democratico, Movimento 5 Stelle, Forza Italia, Lega-Salvini Premier and Fratelli d'Italia) and manually tell the political orientation of that profile;

3. The third task has to collect data in the same way of the second task, but instead of working on profiles picked up from official Twitter accounts of those five parties, the third task works on Twitter profiles randomly chosen all over Twitter.

Our group (group E) has been assigned to the first task, so the first task is the only task treated in this section.

2.1 Structure

The questionnaire [1] is thought to be completed by an adult (whose age is bigger than 18) Italian citizens.
In the first page, there are questions about demographic data (age, gender,

and educational level achieved), except the name because this survey is supposed to be completed anonymously.
In the second page, all the questions are about personality features (reservedness, laziness, sociality, etc.), to discover the user nature, not directly asking about politics.
In the third page, there are 25 sentences taken from official social accounts and declarations of politicians related to the five biggest parties in Italy (Partito Democratico, Movimento 5 Stelle, Forza Italia, Lega-Salvini Premier and Fratelli d'Italia), divided by five big topics (illegal immigration, taxation, election law, ius soli, and living will-euthanasia), and for each topic, there's one sentence related to one party. The participant has to choose the sentence he agrees more between the five sentences related to each topic.
In the fourth page, the participant has to choose a number between 0 and 10 representing his political orientation (0 is for extreme left and 10 is for extremely right), and then has to specify which of the five big parties specified above he is closer to (or say "Other" if he doesn't recognize himself in any of those parties).
In the last page (the fifth) there is a space to insert participant's Twitter username ("None" if the participant hasn't any Twitter account). This survey is also open for people who don't use Twitter because it could be interesting to study the relationship between personality features and closeness to a political party.

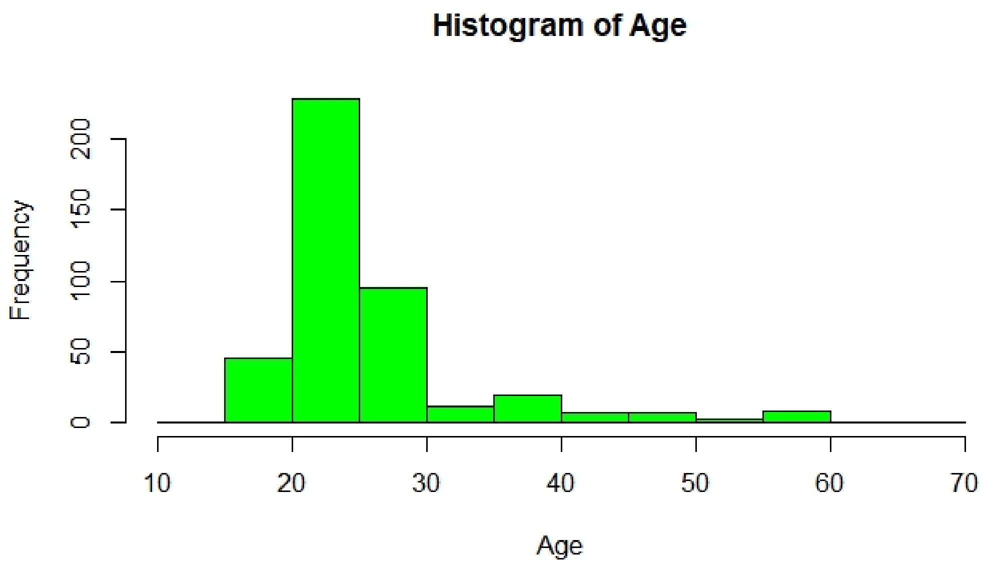
2.2 Limitations

A characteristic problem of each survey is that we don't know if the participant tells the truth while he's answering the questions. There is some bias in recognizing in himself a not socially accepted feature (for example is quite rare to get a positive answer if you ask directly a racist person about his racism). Another problem of a survey like this can be the difficulty of a participant to declare his or her political orientation, above all his or her reference party.
The idea to try to solve these problems is to avoid direct questions and try to obtain results by dealing with answers to indirect questions. For example, in the third page of the survey (the page containing declarations from politicians and social accounts), there isn't any direct reference to a certain politician or a certain political party: this is a good solution for the bias problem.
Another important limitation is strictly connected with Twitter, and with the fact that Twitter is not a very popular social network in Italy. This issue

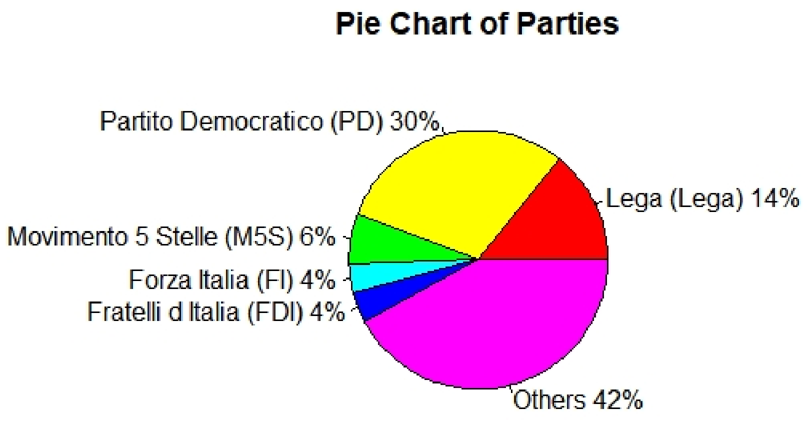
makes worse the capability to get a huge number of participants, which is already a typical problem in each questionnaire.

2.3 Results

Every time a participant finishes filling out the survey, the results are automatically saved in an online Excel document.
As said above, the problem of having a limited range of participants has been very strong; in fact, in about one and a half months (from the 17th October to 27th November) we had 430 participants. The fact of sharing this survey mostly between the people close to us (friends, relatives, Facebook groups of university students) influenced the average age of the participants (that is 26.22326). In fact, in the following histograms it is evident that the biggest concentration of participant is around 20 and 30 years old:



Another problem with this kind of survey is the fear of participants to precisely declare their closest party. In fact, there's a huge percentage (about 42%) of people that do not specify their favorite party, as can be seen in the following pie chart:



Anyway, maybe the biggest problem with those data is the very low percentage of people who sent their Twitter username. This is evident seeing that 316 participants (more than 73% of the whole population!) answered "None" when they were asked about their Twitter username. This was easily predictable by the fact that Twitter is not a very popular social network in Italy.

Chapter 3

Phase 2: Data Analysis

Once the first phase is ended, we have carried on with data analysis. In particular, four different approaches have been proposed to us, and each group had to choose which one of those to implement. The four approaches (tasks) are:

1. Task 1 includes only analysis based on data collected by the first Task of the previous phase. This task has to investigate the correlation between personality and political orientation and the correlation between direct and indirect questions;

2. Task 2 analysis is based on data collected by the second and third tasks of the previous phase. After having split data between two sets (training set and test set), this task has to build a machine learning model with the capability to predict which party of those five a new user is close to;

3. Task 3 also refers to data collected by task 2 and 3 of the previous phase. In this task, the purpose is to train a machine learning model who should be able to predict the political orientation (right or left) of a user on the base of the contents of his tweets;

4. Task 4 is based on data collected by the second and third Tasks of the previous phase too. This task has the aim to perform a topic extraction from tweets and study the temporal trend of those topics related to each party.

Our group chose to join Task 3 in this phase, namely working with data collected by other groups in the first phase.

3.1 Cleaning Tweets

When you have to deal with data, the first step is always to check how clean they are, and eventually, work to delete from those data all the features that are obstacles for the research. In this case, our data were full of noise, and for this cause, we had to clean them.
First of all, it is good to clarify that our data are 7503 Excel tabs, each tab regarding a user and containing all the written contents (tweets and retweets) of his related Twitter account. We have most immediately seen that many of those 7503 files were copies of other files in the same folder: for some users, there were even three identical files (that is referring to the same person) but with distinct names. So, the first step was to delete those excess copies.
After that, we proceeded the cleaning phase working on tweets contents. We first wrote a Python script with the capability to delete from each tweet all the problematic features (in particular emoticons, links, and special characters). Then, we chose a time window to work with: we intend to deal with 2019 contents, so we made sure that the script considered only contents starting from 01/01/2019. Our final aim was to predict political orientation, so we had to find out a way to distinguish between political and not political contents. Our idea was to write a list of keywords linked to political topics and discarding all the contents not containing at least one of these words, we would have just a list of political tweets for each user. Here the above-mentioned list:

```
keys=[ "5s", "berlusconi", "bibbiano", "boldrini", "capitan",
    "cavaliere", "cittadin", "conte", "crisi", "destra", "
   elezion", "euro", "famigli", "figli", "forzaitalia", "
   fratelliditalia",  "gentiloni", "govern", "grillo", "ilva"
   , "italian", "italiaviva", "lavor", "lega", "maio", "
   mattarella", "meloni", "migra",  "ministr", "movimento", "
   paes", "parlament", "partit", "pd", "politic", "president"
   , "raggi", "renzi", "repubblic", "salvini", "sardin", "
   segre", "sinistr", "sold", "vot" ]
```

As we can see, many of these words are just semantic roots. For example, with the word `"parlament"` we include many words belonging to the same semantic family, such as the adjective `"parlamentare"`, both the singular (`"parlamento"`) and the plural (`"parlament"`) noun and so on. At this point, to finish the cleaning phase, we only kept users who have more than 50 tweets according to those parameters.
Before going on with the last step of the cleaning phase, it is good to explain how the groups belonging to task 2 and task 3 collected data during the first phase. Precisely, once a group had a Twitter profile, each component of that group had to classify "by hand" which party that profile was close to,

obtaining in this way a table like this:

Twitter ID	**User Name**	**Sex**	**SID 1**	**SID 2**	**SID 3**	**SID 4**
@marco_ssl	Marco_SSL	M	3	3	1	1
@Mutaz53214	Mutaz	M	3	3	3	3
@predieridanilo	predieri danilo	M	2	4	2	4
@Loredan83532601	Loredana	F	1	1	3	3
@vikingur8	viking	M	3	3	3	3
...	...	...	...	...	...	...

where the columns Twitter ID and User Name identify a user and the columns SID 1, SID 2, SID 3 and SID 4 tell the classification thought by each component of a group (there could eventually be an additional value SID 5 for a group composed by 5 people). In particular, the classification is made by using a number from 0 to 5:

Number	**Party**
0	Others
1	Lega-Salvini Premier
2	Partito Democratico
3	Fratelli d'Italia
4	Movimento 5 Stelle
5	Forza Italia

We chose to deal only with users which classification was quite clear; we only kept users for which classification more than 75% of the group agreed.

3.2 Splitting Set

As in each machine learning problem, once we own cleaned data set next step is to split that set between the training set and test set. In particular:

- training set will be the data portion that we'll use to train the machine learning algorithm;

- test set will be the data portion that we'll use to check the machine learning algorithm; that is that we'll compute the algorithm trained on those data and we'll compare the results obtained with data in our possession, studying in this way the accuracy of that algorithm.

The usual approach is to split the whole set allocating the 80% of the set to the training set, and the remaining 20% to the test set.
But it's not enough; in fact for this job isn't enough to make a random division, because it is important to balance between left and right parties labeled users.
For this purpose, we chose to build four different couples of the training set and test set:

- The first set (unbalanced 75%) is composed of users for which classification more than 75% of the group agreed, and both training and test set are split in 80% of right labeled users and 20% of left labeled users. The size is of 2066 users for the training set and 516 users for the test set;

- The second set (unbalanced 100%) is composed of users for which classification 100% of the group agreed, and both training and test set are split in 80% of right labeled users and 20% of left labeled users. The size is of 2006 users for the training set and 501 users for the test set;

- The third set (balanced 75%) is composed of users for which classification more than 75% of the group agreed, and both training and test sets are split in 50% of right labeled users and 50% of left labeled users. The size is of 808 users for the training set and 202 users for the test set;

- The fourth set (balanced 100%) is composed of users for which classification 100% of the group agreed, and both training and test sets are split in 50% of right labeled users and 50% of left labeled users. The size is of 744 users for the training set and 186 users for the test set;

Precisely, we considered a "right labeled user" each user belonging to Lega-Salvini Premier, Fratelli d'Italia and Forza Italia, while for us a "left labeled user" is a user close to Partito Democratico. For the moment we left aside the Movimento 5 Stelle profiles, because of that party nature. Movimento 5 Stelle was born to gather people "with no left or right ideologies, but ideas". [2]

3.3 Model Definition and Training

Once that we have cleaned the data, we started to look for a model that can be trained and then used to predict the political orientation of a person.
The first thing was using a sentiment analysis tool for the Italian language.

We planned to analyze every single tweet of a user with this tool and, in this way, giving them a polarity score between 0 and 1 both for positivity and negativity of a sentence, where 0 for positivity means that the tweet is a negative sentence and 1 that's a positive sentence(and the opposite for negativity value).
For this purpose, we utilized SentITA, a tool developed by a Ph.D. student of Data Science and publicly available on GitHub [3]. Running this tool on every tweet of every twitter user, we were able to obtain numerical scores and use these scores to train a machine learning model.
During preliminary experiments, Convolutional Neural Networks and LSTM have been used as feature extraction methods. Moreover, along with neural networks preliminary experiments, other tests have been carried out by exploiting approaches as Word2Vec Embedding and TFD-IDF. All the mentioned methods have been discarded in favor of Sentita.

The next step, once we have reduced our problem from a language problem to a numerical problem, was using the random forest technique to train a model for each keyword. More specifically, for every keyword (the same used when cleaning the data), we have selected all the tweets containing the analyzed keyword and passed the polarity score of the single tweet to a random forest. Along with the polarity score of the tweet, we passed to the random forest the party that represents the user that wrote the tweet.
For example, for the keyword "Salvini", we took all of the tweets containing that word. For every tweet we had a polarity score, indicating if the tweet was talking with positive or negative sentences. So a tweet that talked positively about "Salvini" would have a positivity polarity score near +1. Along with that score, we knew what party would vote the Twitter user that wrote the examined tweet.
In this way, for every tweet, we had three information: the positivity and negativity polarity score of the sentences and the party that the user would eventually vote. We trained our random forest with these three parameters for every keyword.
Once this phase was completed, we obtained a trained random forest for each of the keywords.

The next step was to analyze some new users and predict their party.
For this purpose, we trained a random forest. To train this new random forest, each user is represented this way: we took their tweets, for each tweet we looked for keywords (as before) in that tweet and for each keyword found we used that keyword's random forests trained before to predict if that tweet is a left or right one. After this, we have many values for each user,

about which we calculated the mean and used this last value to train the last random forest which can predict if a user is a left or right one.

3.4 Other Experiments

3.4.1 Machine Learning Algorithms

The aforementioned experiments have also run even by using other algorithms. In light of this, XGBoost, SVM and Multiple Kernel Learning have been used. For SVM and Multiple Kernel Learning have been used with different kernels: Linear, Polynomial, Cauchy, and RBF(Radial Basis Function).
We discarded these other approaches because the random forests perform well for our task.

3.4.2 FastText

The first one was related to the setup we used. In particular, it aimed to pass other values to the random forests to increase user representation features. To reach this, other than extract polarity for each tweet, we also user FastText [4] with pre-trained models for Italian language, In particular, for each tweet, we represented it, other than only use positivity and negativity score, both with an average of its words vector embeddings (300 dimension per vector) got with FastText, which represents word as vector of determined size, which are built by capturing word mean related to other words it appears near. Unfortunately, this experiment didn't give nice results so we decided to discard it, also because it was really heavy to manage, due to word embeddings weight.

3.4.3 Language Model

Another experiment wasn't related to positivity and negativity score given by Sentita, but was entirely based on Language Models [5].
Language Models aims to predict the probability of a given sentence to be part of a particular class (in our case left or right), by calculating the probability of the words that sentence is made with N-Grams as explained in [5]. In our case, we created the Bi-Grams probability distributions, based on the train and test sets created before.

To predict a probability of a sentence we then used this formula:

$$p(c|x1x2) = \prod \left(\lambda(x1) * \frac{count(x1x2)}{count(x2)} + (1 - \lambda(x1)) * p(x2) \right)$$

where:

- $count(x1x2)$ represents the number of times x1 and x2 co-occurs in train test;
- $count(x)$ represents the number of times x appears in train test;
- $\lambda(x) = 1 - \frac{u(x)}{u(x)+count(x)}$;
- $u(x)$ is the number of unique words that appears after word x in train set as in [6].

This method achieved an accuracy of 80%, so worse than the configuration we used in the end. This is due probably to the lack of data since it's a statistical model, which normally need tons of data (much more than machine learning algorithms) to perform well.

Chapter 4

Results

4.1 Measures Used

We repeated the same experiment for each of the four couples of sets (unbalanced 75%, unbalanced 100%, balanced 75% and balanced 100%), and to analyze data we used some statistics. But before starting to treat this topic, it is good to introduce this notation according to [7]:

		TRUE CONDITION	
	TOTAL POPULATION	condition positive	condition negative
PREDICTED CONDITION	predicted condition positive	TP	FP
	predicted condition negative	FN	TN

With this notation, we can define the *accuracy* as [7]:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Another statistics we can define is the confusion matrix C, a 2×2 matrix defined as [8]:

$$C = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

Other important definitions of statistics are the definitions of *precision* and *recall* [7]:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

These last two definitions are necessary to compute the F1 score [9], a very useful test for unbalanced data. This test is so significant because it calculates the harmonic mean instead of normal mean(so it takes care about how many data of each class you have). The F1 score is so defined:

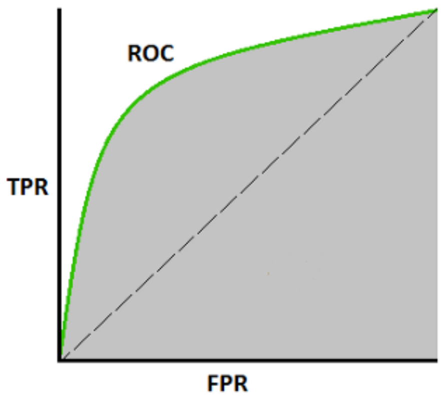
$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

Another important test is the ROC-curve test (ROC is an acronym for *Receiver Operating Characteristic*), but before defining what is a ROC curve is necessary to define what is the true positive rate TPR and what is the false positive rate FPR [10]:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Building a ROC curve means to build the graph of the true positive rate in function of the false positive rate. For the definitions of TPR and FPR we know for sure that this graph passes for the points $(0, 0)$ and $(1, 1)$. Here an example of a ROC curve:



The grey area under the curve is what is called the AUC (area under the curve), and the most AUC is close to 1, the most the model is precise. This

is a good statistics because it doesn't care about the data nature (that is, if the sets are balanced or not).
We decided to compute the ROC test (that is computing the *AUC* score), the accuracy test and the confusion matrix for all the four sets, and the F1 score just for the unbalanced sets.

4.2 Results on Normal Sets

```
#TESTING 75x100_unbalanced
Training set: 2066
Test set: 516
Accuracy : 0.8352713178294574
Confusion matrix:
[[ 24  77]
 [  8 407]]
Roc auc : 0.6091733269712513
F1 score : 0.9054505005561735

#TESTING 100x100_unbalanced
Training set: 2006
Test set: 501
Accuracy : 0.8423153692614771
Confusion matrix:
[[ 19  74]
 [  5 403]]
Roc auc : 0.5960230866540165
F1 score : 0.9107344632768362

#TESTING 75x100_balanced
Training set: 808
Test set: 202
Accuracy : 0.7029702970297029
Confusion matrix:
[[97  4]
 [56 45]]
Roc auc : 0.7029702970297029

#TESTING 100x100_balanced
Training set: 744
Test set: 186
Accuracy : 0.7204301075268817
Confusion matrix:
[[89  4]
 [48 45]]
Roc auc : 0.7204301075268817
```

It is immediately evident that 100% sets give more balanced results, both for accuracy, ROC-test and F1 score points of view than 75% sets, maybe because the training was made over sure profiles.
Another very important conclusion is that looking at their accuracy, there is a better behavior of unbalanced sets compared to balanced sets; this happens because unbalanced sets have more data to train models than balanced sets. We chose to deal with three right parties and just one left party, and unbalanced sets can take this thing into account.
The confusion matrix is also a test who says a lot. Let's analyze first the confusion matrices for unbalanced sets. We can see that there were good predictions for right parties (for example, for unbalanced 100% set there are 403 true negative and 5 false negatives); for left users instead, there are very bad predictions (still for unbalanced 100% set we have only 19 true positives and 74 false positives). We think that this is mainly due to three big reasons:

- first, the fact that we own a very big percentage of the right-oriented user: this thing hampers the algorithm to get well trained over left-oriented users;
- second, the irony: the tool we used to compute the polarity of a tweet is not able to distinguish between ironic and not ironic tweets, causing in this way noise in data;
- third, keywords: the keywords list we wrote unfortunately used to contain more words referring to the right orientation than words referring to left orientation.

Studying confusion matrices, instead, we can see good behavior with left-oriented users (because in both training and test set there is a total balance of left and right oriented users, that bypasses the first reason explained above). Anyway, there's still a bad behavior, this time-shifted over right-oriented users (because there are still the problems two and three treated above).
Overall getting about 84,2% of accuracy with high ROC-test score (59,6%) and F1 score (91,1%) is not a very bad result, and we can conclude that finding a way to definitively bypass the three reasons presented above could even bring to better results.

4.3 Results on Movimento 5 Stelle

After all, we used model trained on unbalanced 75% data, to try to predict users classified as *Movimento 5 Stelle* from groups during the previous task. We had a test set of 7 users of that political party, because our filtering considerably reduced that number.
Final results are below:

```
--- Testing 5s ---

Training set: 2066
Test set: 7
Users
    twitter_id     , Prediction
0  IncazzatoCronic, dx
1     MeliSandro72, dx
2       agoacciaio, dx
3        OrruPiras, dx
4     AndFranchini, dx
5  LiberoP68333298, dx
6         franca_fm, dx
```

As we can see, our model predicted all M5S users as right users. This could be due to the lack of left data to train the model.

Bibliography

[1] Online questionnaire, English version:
http://forms.gle/hhcyTxEwJWRN16E37.

[2] Movimento 5 Stelle site, Italian version:
https://www.movimento5stelle.it.

[3] Sentita, a sentiment analysis tool for Italian:
https://nicgian.github.io/Sentita/.

[4] Library for efficient text classification and representation learning:
https://fasttext.cc/.

[5] Language Models:
https://en.wikipedia.org/wiki/Language_model.

[6] Willem Bell Smoothings:
http://www.phontron.com/slides/nlp-programming-en-02-bigramlm.pdf.

[7] Precision and Recall:
https://en.wikipedia.org/wiki/Precision_and_recall.

[8] Confusion Matrix:
https://en.wikipedia.org/wiki/Confusion_matrix.

[9] F1 score:
https://en.wikipedia.org/wiki/F1_score.

[10] TPR, FPR and ROC curve:
https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Chapter 5

Appendix

5.1 Phase 1 Codes

Here is the code implemented in R to produce the histogram of age and the pie chart of the parties; both graphs are presented on page 4 of this work.

```
#HISTOGRAM OF AGES
library(readxl)
TASK1 <- read_excel(".../data/TASK1.xlsx")
attach(TASK1)
hist(Eta, col='green', xlab = 'Age', main = 'Histogram of Age
    ')
```

```
#PIE CHART OF PARTIES
library(readxl)
TASK1 <- read_excel(".../data/TASK1.xlsx")
attach(TASK1)
parties<-`Indica il partito che ad oggi voteresti se ci
    fossero le elezioni politiche`
p.pd<-length(parties[parties=='Partito Democratico (PD)'])
p.lega<-length(parties[parties=='Lega (Lega)'])
p.fi<-length(parties[parties=='Forza Italia (FI)'])
p.5s<-length(parties[parties=='Movimento 5 Stelle (M5S)'])
p.ot<-length(parties[parties=='Altri'])
p.fdi<-430-(p.pd+p.lega+p.fi+p.5s+p.ot)
slices <- c(p.lega, p.pd, p.5s, p.fi, p.fdi, p.ot)
lbls <- c("Lega (Lega)", "Partito Democratico (PD)", "
    Movimento 5 Stelle (M5S)","Forza Italia (FI)","Fratelli d
    Italia (FDI)", "Others")
pie(slices, labels = lbls, main="Pie Chart of Countries")
pct <- round(slices/sum(slices)*100)
lbls <- paste(lbls, pct)
lbls <- paste(lbls,"%",sep="")
```

```
pie(slices,labels = lbls, col=rainbow(length(lbls)), main="
    Pie Chart of Parties")
```

5.2 Phase 2 Codes

Here is the code implemented in Python to produce the results discussed above.
Next scripts are written in sequence, so to use them, just copy and paste as they are wrote in sequence.

clean_data.py

Needs directory `path + /Tweets` with extracted tweets and `path + /json/keys.json` with keywords.
This script cleans every user's tweet, by removing: special characters, url and newlines. Then keep a tweet only if it's after `starting_date` and has at least 1 keyword.

```
import os
import numpy as np
import pandas as pd
import datetime
import re
import shutil
import json
from joblib import Parallel, delayed
from sentita import calculate_polarity
import csv
from statistics import mean
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
    f1_score, roc_auc_score, classification_report
from sklearn.model_selection import GridSearchCV

path = "../data"
# removes directory name if already present in path and then
    create in again
def create_dir(name):
    try:
        if not os.path.exists(path + name):
            os.makedirs(path + name)
        else:
            shutil.rmtree(path + name)
            os.makedirs(path + name)
    except:
        create_dir(name)
```

```python
keys = []
with open(path + "/json/keys.json", 'r') as f:
    keys = list(json.load(f)["keys"])
# starting_date from which keep tweets
starting_date =  datetime.datetime(2018,12,31)
# minumum numer of tweets a user has to have
minimum_tweets = 50
# removes newlines form tweet
def remove_newlines(string):
    return re.sub(u'[\r\f\n]', ' ', string)
# removes url from tweet
def remove_url(string):
    return " ".join([ el for el in remove_newlines(string).split(" ") if not (el.lower() == ("rt") or el.lower().startswith("http"))])
#removes special characters from tweet
def remove_special(string):
    return re.sub(u\'[^A-Za-z0-9\s.,?!\"$\%&(){}[\]=:;-@*\+#\']', '',remove_url(string))

# creates directory cleaned_tweets
create_dir(name="/cleaned_tweets")

# keeps only tweets with keywords, after date starting_date and cleans them with remove_special, then if remaining at least minimum_tweets tweets writes a new csv file in path + /cleaned_tweets with user tweets cleaned

def clean(account_name, df):
    if len(df) > minimum_tweets:
        res = []
        for tweet in range(len(df)):
            cleaned_tweet = remove_special(df.iloc[tweet]["full_text"])
            if datetime.datetime.strptime(df.iloc[tweet]["created_at"], '%Y-%m-%d %H:%M:%S') > starting_date and len(cleaned_tweet) > 20:
                for keyword in keys:
                    if keyword in cleaned_tweet:
                        res.append(cleaned_tweet)
                        break
        if len(res) > minimum_tweets and account_name.endswith("_tweets.csv"):
            done = os.listdir(path + "/cleaned_tweets")
            name = account_name.split("_tweets")[0][1:] if account_name.startswith("@") else account_name.split("_tweets")[0]
            if not name in done and not name.startswith("Copy of"):
                pd.DataFrame(res).to_csv(path + "/
    cleaned_tweets/" + name + ".csv", sep="," ,encoding='utf
    -8-sig', header=["text"])

# reads user's csv file and cleans it
def clean_tweets(args):
    account_name = args
    try:
        df = pd.read_csv(path + "/Tweets/" + account_name ,
    encoding='utf-8-sig')
        clean(account_name, df)
    except:
        print("err")
# calls clean_tweets for each user in path + /Tweets
Parallel(n_jobs=20)(delayed(clean_tweets)((el)) for el in os.
    listdir(path + "/Tweets"))
```

sentita_extraction.py

It needs directory `path + /cleaned_tweets` with cleaned data and the sentITA library (https://github.com/NicGian/SentITA).
This script creates directory `path + /polarity_classified` and put in it a csv file for each user with polarities of each tweet of that user.

```

#creates directory path + /cleaned\_tweets
create_dir("/polarity_classified")
files = os.listdir(path + '/cleaned_tweets')
columns = ['text','positive','negative']
# removes emoji from tweet
def deEmojify(inputString):
    return inputString.encode('ascii', 'ignore').decode('
    ascii')

# for each user creates a csv file in path + /
    polarity_classified with user's name and positivity and
    negativity for each tweet
for file in files:
    data = pd.read_csv(path + "/cleaned_tweets/"+file, sep=",
    ", encoding="utf8")
    data['text'] = data['text'].apply(deEmojify)
    sentences = data['text'].values
    results, polarities = calculate_polarity(sentences)
    raw_data = np.concatenate([sentences.reshape((len(
    sentences),1)), polarities], axis=1)
    new_data = pd.DataFrame(data=raw_data, columns=columns)
    new_data.to_csv(path + '/polarity_classified/'+file,
    quoting=csv.QUOTE_NONNUMERIC)
```

remove_duplicates.py

This script removes duplicates directly from the original file containing all the users with their parties assigned.

```python
import pandas as pd
import numpy as np
import os
import string


def remove_duplicates(data_path, source_file,
    labelled_file_name):
    print("# Reading data from "+source_file)
    data = pd.read_csv(source_file, sep=";", encoding="utf8")
    diff = len(data)
    print("# Dropping duplicates")
    data.drop_duplicates(subset="Twitter ID",
                         keep="first", inplace=True)
    diff = diff - len(data)
    print("# Dropped " + str(diff) + " duplicates")
    print("# Writing distincts to "+labelled_file_name)
    data.to_csv(data_path+labelled_file_name,
                sep=",", encoding='utf-8-sig', header=["
    twitter_id", "SID 1", "SID 2", "SID 3", "SID 4", "SID 5"],
     index=False)
    print("# Wrote "+labelled_file_name+" to "+data_path)


data_path = "data/"
source_file_name = "original.csv"
source_file = data_path+source_file_name
labelled_file_name = "labelled_accounts.csv"
labelled_file = data_path+labelled_file_name

remove_duplicates(data_path, source_file, labelled_file_name)
```

assign_label.py

This script assigns the political parties to every user based on the previous votes given by groups on *task 1*. Two different thresholds were used to select users:

- 75%: only users with at least 75% agreement on political party were selected;
- 100%: only users with total agreement on political party were selected.

```python
import pandas as pd
import numpy as np
import math
import os
import shutil
from progress.bar import ChargingBar

def label(data_path, labelled_file):
    print("LABELING")
    print("Reading data...")
    data = pd.read_csv(labelled_file,
                       sep=",", encoding="utf8")
    df = pd.DataFrame(data)

    lista = ["SID 1", "SID 2", "SID 3", "SID 4", "SID 5"]

    # -------------- thresholds --------------
    soglie = [0.75, 1.0]
    for soglia_accettazione in soglie:
        users = []
        bar = ChargingBar("Labeling", max=len(df))
        for i in range(len(df)):
            sx, dx, num_voti = (0, 0, 0)
            for sid in lista:
                try:
                    voto = int(df.iloc[i][sid])
                    pass
                except:
                    voto = 0
                    pass
                if not math.isnan(voto):
                    num_voti += 1
                    if voto == 2:  # pd = sinistra
                        sx += 1
                    elif voto != 4:  # no m5s
                        dx += 1
            partito = 0
            if dx != 0 or sx != 0:
                if dx/num_voti >= soglia_accettazione:
                    partito = 1
                elif sx/num_voti >= soglia_accettazione:
                    partito = -1

            twitter_id = df.iloc[i]["twitter_id"]
            if partito != 0 and twitter_id != "":
                users.append([twitter_id, partito])
            bar.next()

        bar.finish()

        path_to_save = data_path+"labelled/"
        folders = os.listdir(data_path)
        if "labelled" not in folders:
            os.makedirs(path_to_save)
        pd_user = pd.DataFrame(users)
        pd_user.to_csv(path_to_save+str(round(soglia_accettazione*100))+"x100.csv", sep=",",
                       encoding='utf-8-sig', header=["twitter_id", "partito"], index=False)
        print("Data saved")

data_path = "data/"
labelled_file_name = "labelled_accounts.csv"
labelled_file = data_path+labelled_file_name
label(data_path, labelled_file)
```

create_sets.py

This script creates different datasets from the labeled data. It generates four different datasets based on two variables:

- percentage of agreement on political parties during task 1 (reliability);
- structure of the dataset (balanced or not).

The different datasets can be represented with the table below:

Dataset	**Reliability**	**Balanced**
Dataset 1	75%	Yes
Dataset 2	100%	Yes
Dataset 3	75%	No
Dataset 4	100%	No

It's important to notice that while creating the datasets, the script was also filtering users that were not present in the list generated by *clean_data.py*.

```python
import pandas as pd
import numpy as np
from progress.bar import ChargingBar
import os
from os import listdir
from os.path import isfile, join
import collections


def createDirs(path_to_save):
    try:
        os.makedirs(path_to_save)
        print(path_to_save+" created")
    except:
        print(path_to_save+" already exists")


def loadUsernames(path):
    files = [f for f in listdir(path) if isfile(join(path, f)
    )]
    bar = ChargingBar("Reading usernames", max=len(files))
    filtered_usernames = []
    for i in range(len(files)):  # per ogni file
        file_name = str(files[i]).replace(".csv", "").replace
    ("@", "")
        filtered_usernames.append(file_name)
        bar.next()
    bar.finish()
    print("Usernames loaded")
    return filtered_usernames


def selectSets(path_labelled, path_usernames, path_to_save,
    balanced):

    if balanced:
        path_to_save += "balanced/"
        print("Balanced")
    else:
        path_to_save += "unbalanced/"
        print("Unbalanced")

    createDirs(path_to_save)

    filtered_usernames = loadUsernames(path_usernames)

    soglie = [75, 100]

    for soglia in soglie:
        file_name = str(soglia)+"x100"
        print("--- "+file_name+" ---")
        data = pd.read_csv(path_labelled+file_name+".csv",
                           sep=",", encoding="utf8")
        df = pd.DataFrame(data)
        tot = len(df)
        print("Users in "+file_name+" = "+str(tot))
        lefties, righties = ([], [])

        bar = ChargingBar("Filtering", max=tot)
    for i in range(tot):
        user = [df.iloc[i]["twitter_id"], df.iloc[i]["partito"]]
        if user[0] in filtered_usernames:
            if user[1] == -1:
                lefties.append(user)
            else:
                righties.append(user)
        bar.next()
    bar.finish()

    tot = len(lefties)+len(righties)
    print("Filtered :" + str(tot))
    print("Left users: " + str(len(lefties)))
    print("Right users: " + str(len(righties)))

    training_percentage, test_percentage = (0.8, 0.2)

    trainingL = round(len(lefties)*training_percentage)
    testL = round(len(lefties)*test_percentage)
    trainingR = round(
        len(righties)*training_percentage) if not balanced else trainingL
    testR = testL

    training_set = lefties[:trainingL] + righties[:trainingR]
    print("training set:\n\tL " +
          str(len(lefties[:trainingL]))+"\n\tR "+str(len(righties[:trainingR])))
    if balanced:
        test_set = lefties[trainingL:] + \
            righties[trainingR:trainingR+testR]
        tot = len(test_set)
        print("balanced test set:\n\tL " +
              str(len(lefties[trainingL:]))+"\n\tR "+str(len(righties[trainingR:trainingR+testR])))
        print("\tL " +
              str(round(len(lefties[trainingL:])/tot*100))+"%\n\tR "+str(round(len(righties[trainingR:trainingR+testR])/tot*100))+"%")
    else:
        test_set = lefties[trainingL:] + righties[trainingR:]
        tot = len(test_set)
        print("unbalanced test set:\n\tL " +
              str(len(lefties[trainingL:]))+"\n\tR "+str(len(righties[trainingR:])))
        print("\tL " +
                  str(round(len(lefties[trainingL:])/tot*100))+"%\n\tR "+str(round(len(righties[trainingR:])/tot*100))+"%")

        pd.DataFrame(training_set).to_csv(path_to_save+str(soglia)+"x100_training.csv",
                                          sep=",", encoding='utf-8-sig', header=["twitter_id", "partito"], index=False)
        print(str(soglia)+"x100_training.csv saved")
        pd.DataFrame(test_set).to_csv(path_to_save+str(soglia)+"x100_test.csv",
                                      sep=",", encoding='utf-8-sig', header=["twitter_id", "partito"], index=False)
        print(str(soglia)+"x100_test.csv saved")


def selectSets5s(path_labelled, path_usernames, path_to_save):

    filtered_usernames = loadUsernames(path_usernames)

    soglie = [75, 100]

    for soglia in soglie:
        file_name = str(soglia)+"x100_5s"
        print("--- "+file_name+" ---")
        data = pd.read_csv(path_labelled+file_name+".csv",
                           sep=",", encoding="utf8")
        df = pd.DataFrame(data)
        tot = len(df)
        print("Users in "+file_name+" = "+str(tot))
        lefties, righties = ([], [])

        bar = ChargingBar("Filtering", max=tot)
        test_set = []
        for i in range(tot):
            user = [df.iloc[i]["twitter_id"], df.iloc[i]["partito"]]
            if user[0] in filtered_usernames:
                test_set.append(user)
            bar.next()
        bar.finish()

        pd.DataFrame(test_set).to_csv(path_to_save+file_name+"_test.csv",
                                      sep=",", encoding='utf-8-sig', header=["twitter_id", "partito"], index=False)
        print(file_name+"_test.csv saved")

```

```

# --- Paths
# labelled data
path_labelled = "../data/labelled/"
# filtered usernames
path_usernames = "../data/cleaned_tweets/"
# where to save the sets
path_to_save = "../data/set/"
# creates the folders necessary to run the script
createDirs(path_to_save)
# balanced datasets
selectSets(path_labelled, path_usernames, path_to_save,
    balanced=True)
# # unbalanced datasets
selectSets(path_labelled, path_usernames, path_to_save,
    balanced=False)
# test set per i 5 stelle
selectSets5s(path_labelled, path_usernames, path_to_save)
```

create_keys_data.py

Needs `path + /polarity_classified` and `path + /set` directories with data. For each set, this script creates a `path + /keys_data_ + set_name` directory. In each of these directories, there is a directory for each keyword. In each keyword's directory, creates `x_train, x_test, y_train, y_test` csv files with data relative to that keyword's tweets polarities.

```
values= ["75x100_unbalanced", "100x100_unbalanced", "75
    x100_balanced", "100x100_balanced"]
# return an array with all user's tweet with keyword in it
def extract_for_user(args):
    user, user_data, key = args
    sub_res = []
    try:
        df = pd.read_csv(path + "/polarity_classified/" +
    user + ".csv", encoding='utf-8-sig', engine='python')
        for i in range(len(df)):
            if key in df.iloc[i]["text"]:
                sub_res.append([df.iloc[i]["positive"], df.
    iloc[i]["negative"], user_data["partito"].values[0]])
    except:
        print(user)
    return sub_res

# parallelizes extract_for_user for each user given a key
def get_values_forest(key, data):
    res = []
    partial_res = Parallel(n_jobs=20)(delayed(extract_for_user)((user, data[data["twitter_id"] == user], key)) for user in data["twitter_id"])
    try:
        res = np.concatenate(tuple([el for el in partial_res if len(el) > 0]), axis=0)
    except ValueError:
        print("Error with key: "+key)
        pass
    finally:
        return res
# creates all keys_data for each set created with selector.py
for s in  values:
    if s == "75x100_unbalanced":
        p1 = "unbalanced/75x100_training.csv"
        p2 = "unbalanced/75x100_test.csv"
    elif s == "100x100_unbalanced":
        p1 = "unbalanced/100x100_training.csv"
        p2 = "unbalanced/100x100_test.csv"
    elif s == "75x100_balanced":
        p1 = "balanced/75x100_training.csv"
        p2 = "balanced/75x100_test.csv"
    else:
        p1 = "balanced/100x100_training.csv"
        p2 = "balanced/100x100_test.csv"

    labelled = pd.read_csv(path + "/set/" + p1, encoding='utf-8-sig')
    labelled_test = pd.read_csv(path + "/set/" + p2, encoding='utf-8-sig')
    # csv columns names
    cols = ['P', 'N', 'SX/DX']
    drop_cols = ['SX/DX']
    # creates directory where to store data
    create_dir("/keys_data_" + s)

    # calls get_values_forest for each keyword and creates train and test files, storing them in keys_data + s
    for key in keys:
        create_dir("/keys_data_" + s + "/" + key)
        vals_train = get_values_forest(key, labelled)
        vals_test = get_values_forest(key, labelled_test)
        data_train, data_test = pd.DataFrame(vals_train, columns=cols), pd.DataFrame(vals_test, columns=cols)
        a, b, c, d = data_train.drop(drop_cols, axis=1), data_train["SX/DX"], data_test.drop(drop_cols, axis=1), data_test["SX/DX"]
        # store in keys_data_ + s : x_train, y_train, x_test, y_test
        a.to_csv(path + "/keys_data_" + s + "/" + key + "/x_train.csv", sep=",", encoding='utf-8-sig', index=False)
        b.to_csv(path + "/keys_data_" + s + "/" + key + "/y_train.csv", sep=",", encoding='utf-8-sig', header=False, index=False)
        c.to_csv(path + "/keys_data_" + s + "/" + key + "/x_test.csv", sep=",", encoding='utf-8-sig', index=False)
        d.to_csv(path + "/keys_data_" + s + "/" + key + "/y_test.csv", sep=",", encoding='utf-8-sig', header=False, index=False)
```

create_averages.py

It needs `path + /keys_data` and `path + /set` directories.
For each keyword, this script trains a random forest that which, given polarities of a tweet, predicts if it's a right or left tweet. After these, the script uses those models this way: for each user, for each tweet in test, looks for each keyword if is in it, if it does, predict if it is a right or left one, and then returns the mean of these values for each user (in the end, a user is represented by the mean of that user's tweets predictions) and store this data in `path + /averages_ + set` csv files.

```

# trains and tests a keyword's model on data
def test_model(x_train, y_train, x_test, y_test, word):
    clf = RandomForestClassifier(max_depth=20, random_state=0)
    clf.fit(x_train, y_train)
    pred = clf.predict(x_test)
    print("\n Accuracy " + word + " : " + str(accuracy_score(y_test, pred)))
    confusion_matrix(y_test, pred)
    return clf, accuracy_score(y_test, pred)

# loads trains and tests for a keyword
def get_data(word, s):
    a = pd.read_csv(path + "/keys_data_" + s + "/" + word + "/x_train.csv", encoding='utf-8-sig')
    b = pd.read_csv(path + "/keys_data_" + s + "/" + word + "/x_test.csv", encoding='utf-8-sig')
    c = pd.read_csv(path + "/keys_data_" + s + "/" + word + "/y_train.csv", encoding='utf-8-sig', header=None)
    d = pd.read_csv(path + "/keys_data_" + s + "/" + word + "/y_test.csv", encoding='utf-8-sig', header=None)
    return ( a, b, c, d )

```

```python
# for each tweet of user, looks for each keyword if it's present and in case, predicts with relative model if user is right or left and returns mean of all values.
def evaluate_user(args):
    u, labelled = args
    actual_user = labelled.iloc[u]
    df = pd.read_csv(path + "/polarity_classified/" + actual_user["twitter_id"] + ".csv", encoding='utf-8-sig')
    res1, res2 = ([], [])
    for tweet in range(len(df)):
        for key in keys:
            if key in df.iloc[tweet]["text"]:
                try:
                    r = df.loc[tweet, ["positive", "negative"]]
                    pred = models[key].predict_proba([r])[0]
                    res1.append(pred[0])
                    res2.append(pred[1])
                except:
                    pass
    return [actual_user["twitter_id"], mean(res1), mean(res2), actual_user["partito"]]
for s in  values:
    if s == "75x100_unbalanced":
        p1 = "unbalanced/75x100_training.csv"
        p2 = "unbalanced/75x100_test.csv"
    elif s == "100x100_unbalanced":
        p1 = "unbalanced/100x100_training.csv"
        p2 = "unbalanced/100x100_test.csv"
    elif s == "75x100_balanced":
        p1 = "balanced/75x100_training.csv"
        p2 = "balanced/75x100_test.csv"
    else:
        p1 = "balanced/100x100_training.csv"
        p2 = "balanced/100x100_test.csv"
    # reads data
    labelled = pd.read_csv(path + "/set/" + p1, encoding='utf-8-sig')
    labelled_test = pd.read_csv(path + "/set/" + p2, encoding='utf-8-sig')
    models = {}
    acc = 0
    cols = ['P', 'N']
    # trains each keyword's model to predict if a tweet is from a right or left user
    for word in keys:
        try:
            x_train , x_test, y_train, y_test = get_data(word, s)
            models[word], a = test_model(x_train.values,
y_train.values,x_test.values, y_test.values, word)
            acc += a
        except:
            pass

    # calculates mean of prediction for train users and store it
    partial_res_train = Parallel(n_jobs=16)(delayed(evaluate_user)((user, labelled)) for user in range(len(labelled)))
    pd.DataFrame(partial_res_train, columns=["id", "dx", "sx", "partito"]).to_csv(path + "/averages_train_" + s + ".csv", sep=",",encoding='utf-8-sig', header=["id", "dx", "sx", "partito"], index=False)
    # calculates mean of prediction for test users and store it
    partial_res_test = Parallel(n_jobs=16)(delayed(evaluate_user)((user, labelled_test)) for user in range(len(labelled_test)))
    pd.DataFrame(partial_res_test, columns=["id", "dx", "sx", "partito"]).to_csv(path + "/averages_test_" + s + ".csv", sep=",",encoding='utf-8-sig', header=["id", "dx", "sx", "partito"], index=False)
```

make_predictions.py

Needs `path + /set` directory and `path + / average_ + set` csv for each set.
For each set, this script makes prediction of test set and print results in a file names `path + /res.txt`

```

vals = ["75x100", "100x100"]
sets = ["unbalanced", "balanced"]

# test a model and returns results on test set
def test_model(x_train, y_train, x_test, y_test):
    clf = GridSearchCV(RandomForestClassifier(), param_grid={"n_estimators": [i for i in range(5, 30)], "criterion": ("gini", "entropy")})
    clf.fit(x_train, y_train)
    pred = clf.predict(x_test)
    return "\nTraining set: "+str(len(x_train)) + "\nTest set: "+str(len(x_test))+"\nAccuracy : " + str(accuracy_score(y_test, pred))+"\nConfusion matrix:\n"+str(confusion_matrix(y_test, pred))+"\nR^2 : "+str(r2_score(
    y_test, pred))+"\nRoc auc : "+str(roc_auc_score(y_test,
    pred))+"\nF1 score : "+str(f1_score(y_test, pred))

# reads train data
def get_data_train(labelled, averages):
    return pd.DataFrame([list(averages.loc[averages["id"] ==
    labelled.loc[user, "twitter_id"]][["dx", "sx"]].values[0])
     for user in range(len(labelled))]).values

# reads test data
def get_data_test(labelled, averages):
    return pd.DataFrame([averages.loc[averages["id"] ==
    labelled.loc[user, "twitter_id"]][["partito"]].values[0]
    for user in range(len(labelled))]).values

r = ""
# for each set makes predictions and store data
for s in sets:
    for v in vals:
        labelled_train = pd.read_csv(
            path + "/set/" + s + "/" + v + "_training.csv",
    encoding='utf-8-sig')
        labelled_test = pd.read_csv(
            path + "/set/" + s + "/" + v + "_test.csv",
    encoding='utf-8-sig')
        averages_train = pd.read_csv(
            path + "/averages_train_" + v + "_" + s + ".csv",
     encoding='utf-8-sig')
        averages_test = pd.read_csv(
            path + "/averages_test_" + v + "_" + s + ".csv",
    encoding='utf-8-sig')
        r += "\n\n--- Testing " + v + "_" + s + " ---\n"
        x_train, y_train = get_data_train(
            labelled_train, averages_train), get_data_test(
    labelled_train, averages_train)
        x_test, y_test = get_data_train(labelled_test,
    averages_test), get_data_test(
            labelled_test, averages_test)
        r += test_model(x_train, y_train.ravel(), x_test,
    y_test.ravel())

with open(path + '/res.txt', 'w') as f:
    f.write(r)
```