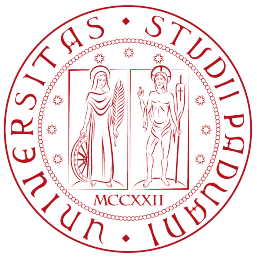


UNIVERSITY OF PADUA

Department of Mathematics "Tullio Levi-Civita"

Master's Degree in Data Science

Cognitive Behavioral and Social Data Project

Predicting political orientation from Twitter contents

Students
Eugenia Anello, Bianca Andreea Ciuche, Luca Dal Zotto, Bleona Mukaj

January 20, 2020 - Academic Year 2019/2020

Contents

1 Introduction **1**
1.1 Presentation of the problem . . . 1
1.2 Previous Literature . . . 2

2 Preparation of the Dataset **4**
2.1 Data Collection . . . 4
2.2 Data Cleaning . . . 6
2.3 Class Balancing . . . 7

3 Machine Learning Tools **10**
3.1 Text Analysis . . . 10
3.1.1 Bag Of Words . . . 10
3.1.2 Term Frequency–Inverse Document Frequency . . . 11
3.1.3 FastText . . . 12
3.2 Classification Algorithms . . . 12
3.2.1 Support Vector Machine . . . 12
3.2.2 Decision Tree and Random Forest . . . 13
3.2.3 Logistic Regression . . . 13
3.2.4 Stochastic Gradient Descent Classifier . . . 14
3.2.5 Multinomial Naive Bayes . . . 15
3.2.6 XGBoost Classifier . . . 15

4 Implementation and Results **16**
4.1 FastText . . . 16
4.1.1 Classifing Movimento 5 Stelle . . . 17
4.2 Analysis with some ML models . . . 18
4.2.1 Classifing Movimento 5 Stelle . . . 21
4.3 Conclusion . . . 23

Bibliography **24**

Chapter 1

Introduction

1.1 Presentation of the problem

Nowdays we have seen that political campaigns are placing more emphasis on social media tools as a low-cost platform for connecting with voters and promoting engagement among users in their political base. Social media is used by people to share their opinions and views. The use of the **Twitter** micro-blogging platform as a tool to predict the outcomes of social phenomena is a recurrent task in the recent social network analysis literature. An important part of the population shares opinions and news related to politics or causes they support, thus offering strong cues about their political preferences and ideologies. This trend is also fueled in part by the fact that voters are increasingly engaging with the political process online.

Twitter is an outlet for up-to-the-minute status updates, allowing campaigns, candidates and citizens to respond in real-time to news and political events. Virtually all candidates and elected officials have a presence on Twitter and many users rely on Twitter to stay informed about political events. The content and structure of the political discussion that takes place on this platform, easily accessible through their API, represents a unique opportunity for researchers interested in the study of elections and public opinion.

So, the aim of our study is to **predict the political orientation based on Twitter users' posts**. Our analysis yields this way:

First, for the 5 major italian parties we collected the data by using party's official Twitter profiles and by going through profile on Twitter randomly. We downloaded the data and then we created two dataframes, analyzing them together: one with the 100% of labeling agreement and the other with the 75% of agreement. And then afterwards, we pre-processed these datasets by cleaning and balancing by undersampling it.

Secondly, for the prediction of the political orientation "right vs left", we have considered 'Lega', 'Fratelli d'Italia' and 'Forza Italia' as **"right"** and 'Partito Democratico' as **"left"**.

Therefore, our task come under the category of binary classification since it involves only 2 classes. Then we trained and tested different classification models, dividing the whole dataset into two parts as follows: 80% of participants in training set and 20% in test set, assigning them randomly.

For a first attempt, we created a model using **FastText**, an advanced Natural Language Processing (NLP) tool which can be used both for text representation and for text classification.

Then we used some simpler word vectorizer tools: **Bag-of-Words (BoW)** and **Term Frequency–Inverse Document Frequency (TF-IDF)**. Their results have been used as an input for the following Machine Learning classifiers:

- Linear SVC (Support Vector Classifier)
- Logistic Regression
- Multinomial NB (Naive Bayes)
- Random Forest Classifier
- SGD Classifier (Stochastic Gradient Descent)
- XGBoost Classifier

Finally, we also investigated how the participants who vote for 'Movimento 5 Stelle' are classified, in base of our classifiers and their features.

1.2 Previous Literature

Predicting Political Orientation based on Twitter's Content: A Literature Review

Elections play crucial role in all democracies and social media is an important aspect in this process. Presently, political parties increasingly rely on social media platforms, mostly Twitter for political communication.The use of social media in political marketing campaigns has grown dramatically over the past few years. But how can we use Twitter on making prediction based on their contents?

There are several streams of research investigating the role of Twitter on making political prediction.

To begin with, the study (Praet et al.2018) [1] was the first to predict political leaning and party preference in Flanders, which has a multi-party system, using Facebook likes. They showed that even when excluding political likes, prediction can be done quite. From the predictive performances they conclude that left and right voters show more consistent behavior on Facebook compared to center voters. Secondly, they developed new methods to gain insights into voter profiles. Voter profiles are described by the most related individual Facebook pages to a certain political leaning and party.

Secondly, a study based on German federal election, 2009, (Tumasjan et al.2010) [2], has analysed 100.000 users' messages, mentioning parties or politicians. Overall, they found that Twitter is indeed used as a platform for political deliberation. The mere number of tweets reflects voter preferences and comes close to traditional election polls, while the sentiment of Twitter messages closely corresponds to political programs, candidate profiles, and evidence from the media coverage of the campaign trail.

Another study was the Political Ideology Prediction of Twitter Users (Preot,iuc-Pietro et al.2017) [3] of the University of Pennsylvania. This study analyzed user-level political ideology through Twitter posts. They made use of a novel dataset where fine-grained user political ideology labels are obtained through surveys as opposed to binary self-reports.

So according to these studies, and many other ones, we can see that social media have the potential to increase political participation, and make it easier for researcher to make predictions based on their contents.

Chapter 2

Preparation of the Dataset

2.1 Data Collection

The fundamental phase of every supervised machine learning algorithm is the training, by which the actual model is created. To be more specific, the purpose of this phase is to model the phenomenon at hand trying to approximate the (supposed existing) rule that maps the tweets of a person to his political orientation. To do this, it is crucial to have a set of twitter users whose real political orientations are known. This collection of data is referred to as **Ground Truth**.

Three different methods have been proposed to gather this information:

1. The compilation of an online survey;
2. The direct analysis of the tweets of the followers of the five major political parties;
3. The direct analysis of the tweets of random Twitter users.

Our group chose the second task, so let's see it in detail. Each group that chooses the second task is assigned one of the following Italian political parties: Lega, Partito Democratico, Fratelli d'Italia, Movimento 5 Stelle, Forza Italia. Then, the members of these groups have to go to the official page of the specific party and access to the list of followers of that page. For each follower these actions were performed:

A) Check if the account is not a bot, but a real user;

B) Keep the profile only if it is an Italian Twitter user ID;

C) Label that user according to his tweets text.

Since the involved subjects are not directly asked to express their political preferences, it is more likely that the ground truth resulting from this method should be clean from the effect of deception or from the poor compliance. This is the principal advantage offered by the so-called **indirect measures**. On the other hand, a limitation can been argued: by

discarding the non-active users, we are creating a group bias. Since it seems reasonable to assume that being active users does not characterize any particular political orientation, it is not necessary to worry about this bias.

Now it is shown how this phase has been performed: as a first step, we downloaded a list of several thousands of followers of the official Twitter page of Fratelli d'Italia. The scraping tool we used is **Twint**, a Python library that allows scraping Tweets from Twitter profiles without using Twitter's API. [4]

After the installation, the simple command twint -u @FratellidItalia –followers –user-full on the Anaconda Prompt, gave us some information of these followers, to be more specific we got: Twitter ID, User Name, whether that profile is Private, whether that profile is verified, Bio, Location, URL, the creation date of that profile and the number of following, followers likes, media and Tweets. Using this information, we managed to discard a good percentage of bots and non-active users.

As a final step, we filled an excel sheet (Figure 2.1) created and shared by the teachers with these data: Twitter ID, User Name, Sex, political preference, the relative frequency of each class and other eventual information. Each member of the group had to insert a number indicating his own idea about the political orientation of every user. It can be noticed that some profiles have not been classified in the same way by each member of the group. This is due to the fact that either that user has not a precise political orientation or the information given by his tweets is not enough to clearly understand his political preferences. That's where the two levels of agreement (75% and 100%) come from. Moreover, several users have been kept even though they have not a political orientation (at least not explicit). This is not a problem; indeed, they are useful because they will allow the classifier model to understand when a user should not be labeled in any way.

	Student ID	Student Name
	SID 1	Luca
	SID 2	Bleona
	SID 3	Bianca
	SID 4	Eugenia

Party Name	Political labeling
Lega-Salvini Premier - @LegaSalvini	1
Partito Democratico - @pdnetwork	2
Fratelli d'Italia - @FratellidItalia	3
Movimento 5 Stelle - @Mov5Stelle	4
Forza Italia - @forza_italia	5

Twitter ID Information			Political Party Labelling				Other Info (Notes)	Class Frequency					
Twitter ID	User Name	Sex (M/F)	SID 1	SID 2	SID 3	SID 4		0	1	2	3	4	5
[redacted]	[redacted]	M	3	3	1	1		0	0,5	0	0,5	0	0
[redacted]	[redacted]	M	3	3	3	3		0	0	0	1	0	0
[redacted]	[redacted]	M	2	4	2	4		0	0	0,5	0	0,5	0
[redacted]	[redacted]	F	1	1	3	3		0	0,5	0	0,5	0	0
[redacted]	[redacted]	M	1	1	1	1		0	1	0	0	0	0
[redacted]	[redacted]	F	1	3	1	3		0	0,5	0	0,5	0	0
[redacted]	[redacted]	M	0	0	0	0	No political tweets	1	0	0	0	0	0
[redacted]	[redacted]	M	3	3	3	3		0	0	0	1	0	0
[redacted]	[redacted]	M	1	3	1	3		0	0,5	0	0,5	0	0
[redacted]	[redacted]	M	5	5	5	5		0	0	0	0	0	1
[redacted]	[redacted]	M	0	0	0	0	No political tweets	1	0	0	0	0	0
[redacted]	[redacted]	F	1	1	1	1		0	1	0	0	0	0
[redacted]	[redacted]	M	1	1	1	1		0	1	0	0	0	0
[redacted]	[redacted]	M	3	3	3	3		0	0	0	1	0	0
[redacted]	[redacted]	M	3	1	1	3		0	0,5	0	0,5	0	0
[redacted]	[redacted]	F	3	3	1	1		0	0,5	0	0,5	0	0

Figure 2.1: First rows of the users classification file

In this way each group classified approximately 1000 users. In order to establish the ground truth, it is necessary to download the text of each user tweets. The most important aspect of the code we implemented is that it uses the Python library **Tweepy**, which requires the Twitter developer credentials.

As an output, a csv file per user was obtained. In each of these files, there is a list of the tweets of that user, including the ID of the tweet, the time it was created, the number of likes obtained, the number of times it has been retweeted, the source (smartphone, desktop, etc.) and the full text. (Figure 2.2)

	A	B	C	D	E	F
1	id	created_at	favorites	retweets	source	full_text
2						
3	1196485373817500000	18/11/2019 17:48	0	0	Twitter for Android	https://t.co/AI7WlWWGSZ
4						
5	1196413939363580000	18/11/2019 13:04	0	28	Twitter for Android	RT @[redacted]: Il non detto: non alzeremo i salari anche alle donne, ma li abbasseremo anche agli uomini. E giustizia sarÃ fatta. https:/â€¦
6						
7	1194725768733030000	13/11/2019 21:16	0	0	Twitter for Android	ðŸ¤"emendamento per contrastare i cambiamenti climatici ....ok .... https://t.co/PrTWQHpyII
8						
9	1191043534029310000	03/11/2019 17:24	0	0	Twitter for Android	@[redacted] @ItaliaViva Troveremo qualcuno che lotti PER il bene di italiani e non?
10						
11	1190205048208400000	01/11/2019 09:52	0	19	Twitter for Android	RT @[redacted]: Il centrodestra ha sbagliato ad astenersi sulla mozione Segre. Avrebbe dovuto votare contro, motivando con serietÃ la pâ€¦
12						
13	1189993755555820000	31/10/2019 19:53	1	0	Twitter for Android	@[redacted] Poi mettere insieme antisemitismo e islamofobia fa sorridere amaro
14						
15	1189992492395040000	31/10/2019 19:48	0	0	Twitter for Android	@[redacted] @[redacted] Prima ti dividono in tutti i modi vecchi/giovani populisti/dem statali/p.iva odiatori/umani ecc. buttano continuamente benzina
16						
17	1189912925621960000	31/10/2019 14:32	0	21	Twitter for Android	RT @[redacted] â‹† #CURIOSITÃ€: Affermare che un bambino ha bisogno di una #mamma (femmina) e un #papÃ (maschio) Ã© promuovere uno #stereotipo?
18						
19	1189912583010220000	31/10/2019 14:30	0	128	Twitter for Android	RT @[redacted]: Â«Ci sono quelli che odiano il Cristianesimo e chiamano il loro odio "amore universale per tutte le religioni."Â» https://â€¦
20						
21	1189851048434900000	31/10/2019 10:26	0	54	Twitter for Android	RT @[redacted]: I fascisti sono tornati. Hint: non sono fra gli astenutiâ€¦ https://t.co/V6QgllEWjR
22						
23	1189846142059710000	31/10/2019 10:06	0	0	Twitter for Android	Da piccoli "che c'Ã¨ di male" poi c'Ã¨ questo (link) poi occultismo #nofakenews https://t.co/kVCQ09CyRO
24						
25	1189604748661570000	30/10/2019 18:07	0	0	Twitter for Android	@[redacted] Serva per ricucire il paese

Figure 2.2: Example of a Tweets file

These csv files represent the starting point of the project. In the next section, it is shown how the data has been cleaned and organized in order to make it possible to use them with the Machine Learning tools.

2.2 Data Cleaning

The text obtained from tweets is not ready to be used for model training, so it needs to be pre-processed first. Clearly, we may not be able to make it completely clean but should try our best to pre-process as much as possible.

Data Cleaning is the process of identifying the incorrect, incomplete, inaccurate, irrelevant or missing part of the data and then modifying, replacing or deleting them according to the necessity. This is a crucial phase because if the data contains noise this will create distortion in the models.

The cleaning of the dataframes was done following this steps :

- **Remove urls:** all the urls have to be removed since they don't have any predicting significance
- **Remove punctuation:** characters such as "?", "!", ";" have to be removed.

- **Upcase /downcase:** all the words, for example Lega and lega must to have the same predicting power, so for this reason we have to downcase every word.
- **Remove emoji:** all the emoji have to be removed because it is not possible to analyze them.
- **Remove of Italian and English stop words:** Stop words are the words in that are used just for the sake of correct sentence formations. All Italian words like "anche" or "ancora" and all English words such as "what" or "the", won't have any predicting power since they are commonly used. For this reason, they may represent noise that have to be removed.

2.3 Class Balancing

In our case study we present two datasets. The first dataset with the political labeling agreement larger than 75% consists of 3,617 twitter accounts, while the second dataset with 100 % of agreement contains 3,174 users. We have focused our selection on Twitter users that have at least one tweet between June 1^{st} and December 1^{st} 2019. The users in these datasets have been annotated with different political labels: the value 0 corresponding to left, the value 1 corresponding to right. So in these datasets there is a row for each Twitter user with all the cleaned tweets in the column called Tweets and the political orientation in the column Party.

One of our main concerns when developing a text classification model is whether the different classes are **balanced**. This means that the dataset contains an approximately equal portion of each class. [5]

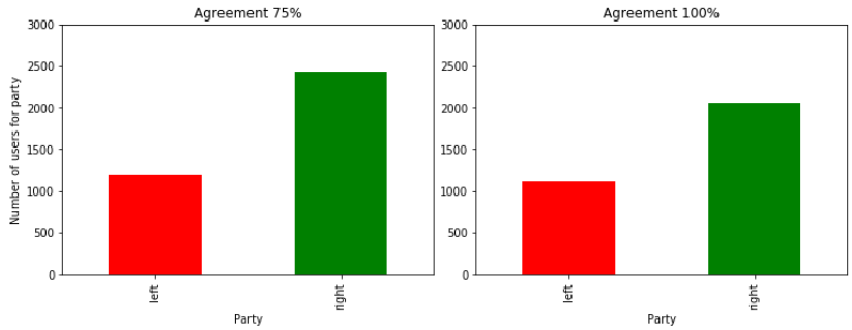


Figure 2.3: The unbalanced datasets after the cleaning

From Figure 2.3 we can see that the classes aren't balanced. In the dataframe with 75% of agreement, the number of users identified as left party are 1185, while the the number of users identified as right party are 2432. In the dataframe with 100% of agreement, the number of users identified as left party are 1064 while the the number of users identified

as right party are 2059. Therefore, the majority class is formed by right party's users and the minority class consists of left party's users.

A widely adopted technique for dealing with highly unbalanced datasets is called **undersampling**, which consists of removing samples from the majority class. We kept all the users labeled as left (the minority class) and we selected just a subset of the right users, so that the two classes would have the same cardinality. To give some numbers, when considering the dataframe with the labeling agreement larger than 75%, we found two classes of cardinality 1185, which means that the whole dataset has 2370 rows. On the other hand, the dataset with 100% agreement, gave two classes of cardinality 1064, for a total of 2128 users. In Figure 2.4 and Figure 2.5 it is possible to check this information.

```
# Now check the cardinality of the two classes after balancing


df_left = newdf75_all[newdf75_all['label']=='__label__0']
df_right = newdf75_all[newdf75_all['label']=='__label__1']

print('after undersampling the right class has cardinality: ' + str(len(df_right)))
print('after undersampling the left class has cardinality: ' + str(len(df_left)))
```

```
after undersampling the right class has cardinality: 1185
after undersampling the left class has cardinality: 1185
```

```
# barplot for a graphical verification

ax=newdf75_all.groupby('label').tweets.count().plot.bar(ylim=(0, 2000),color=['red','green'])
ax.set_xticklabels(['left','right'])
plt.title('Agreement 75%')
plt.xlabel('Political Orientation')
plt.ylabel('Number of users')
plt.show()
```

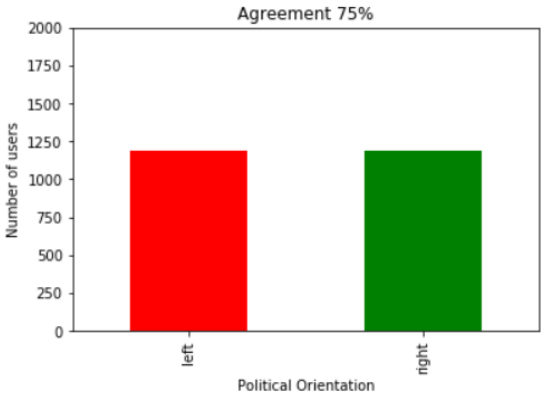


Figure 2.4: The balanced dataset with 75% agreement

```
# Now check the cardinality of the two classes after balancing


df_left = newdf1_all[newdf1_all['label']=='__label__0']
df_right = newdf1_all[newdf1_all['label']=='__label__1']

print('after undersampling the right class has cardinality: ' + str(len(df_right)))
print('after undersampling the left class has cardinality: ' + str(len(df_left)))
```

```
after undersampling the right class has cardinality: 1064
after undersampling the left class has cardinality: 1064
```

```
# barplot for a graphical verification

ax=newdf1_all.groupby('label').tweets.count().plot.bar(ylim=(0, 2000),color=['red','green'])
ax.set_xticklabels(['left','right'])
plt.title('Agreement 100%')
plt.xlabel('Political Orientation')
plt.ylabel('Number of users')
plt.show()
```

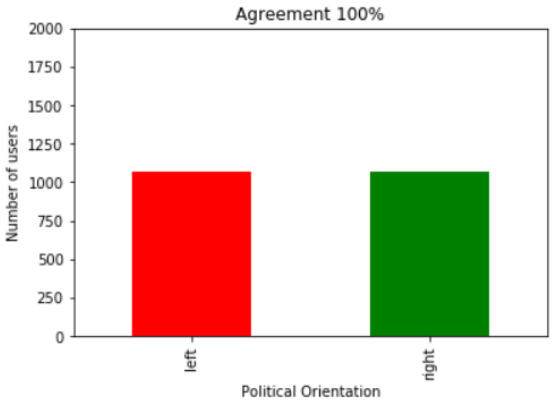


Figure 2.5: The balanced datasets with 100% agreement

Chapter 3

Machine Learning Tools

3.1 Text Analysis

BoW and **TF-IDF** are the first step in the text analysis pipeline. These two tools are mainly used to convert raw text to numerical features which are then fed to Natural Language Processing (NLP) or Machine Learning (ML) algorithms. In our specific scenario, textual data cannot be directly used as input for machine learning classifiers. Therefore, the text first has to be vectorized. [6]

3.1.1 Bag Of Words

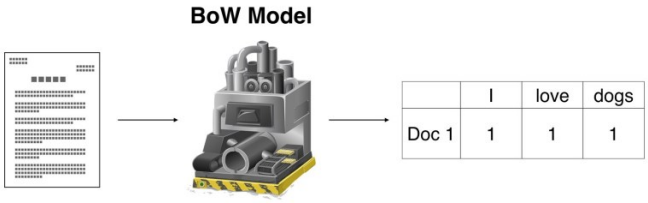
The **Bag-of-Words (BoW)** approach is perhaps the easiest method for text classification. It is an algorithm that counts how many times a word appears in a document. Those word counts allow us to compare documents and gauge their similarities for applications like search, document classification and topic modeling.

It is called a "bag" of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, while their position is not relevant. This technique involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

The BoW outputs a table where each row corresponds to a document and each column represents a unique word. The entries are the counts of each word in each document.

As a simple exaxmple, let's suppose that our set of documents consists of only one document – a sentence: "I love dogs". Applying the BoW model we get:



The bag-of-words can be as simple or complex as you like. The complexity comes both in deciding how to design the vocabulary of known words (or tokens) and how to score the presence of known words. For example, if we consider single words as tokens, then we are handling the so called **unigrams**, while if we consider n-words at a time, we are using **n-grams**.

3.1.2 Term Frequency–Inverse Document Frequency

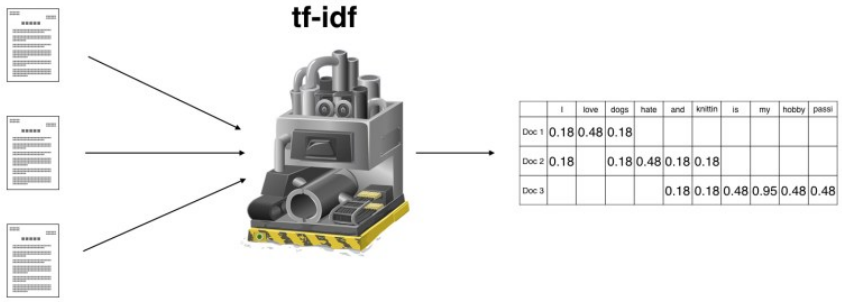
TF-IDF is shorthand for **Term Frequency – Inverse Document Frequency** [7]. So, two things: term frequency and inverse document frequency. Term frequency (TF) is basically the output of the BoW model. For a specific document, it determines how important a word is by looking at how frequently it appears in the document. Term frequency measures the local importance of the word: if a word appears a lot of times, then the word must be important.

The second component of TF-IDF is inverse document frequency (IDF). For a word to be considered a signature word of a certain document, it shouldn't appear that often in the other documents. Thus, a signature word's document frequency must be low, meaning its inverse document frequency must be high. The IDF is usually calculated as:

$$\text{idf(W)} = \log \frac{\#(\text{ documents })}{\#(\text{ documents containing word W})}.$$

The TF-IDF is the product of these two frequencies. For a word to have high TF-IDF in a document, it must appear a lot of times in said document and must be absent in the other documents. It must be a signature word of the document.

Let us show an example of how is TF-IDF applied:



3.1.3 FastText

FastText is an open-source, free, lightweight library for efficient learning of word representations and sentence classification, created by Facebook's AI Research (FAIR) lab. It is written in C++ and supports multiprocessing during training. FastText allows to train supervised and unsupervised representations of words and sentences.

FastText is capable of training with millions of example text data in hardly ten minutes over a multi-core CPU and perform prediction on raw unseen text among more than 300.000 categories in less than five minutes using the trained model.

In this specific case, we want to build a classifier that automatically recognize the party of a user only on the base of the tweets. To obtain a classifier we need labeled data, which consists of tweets and they corresponding parties (or labels). With the labeled data we can train our supervised classifier. All the labels start by the _label_, prefix, with is how fastText recognize what is a label or what is a word. [8]

3.2 Classification Algorithms

3.2.1 Support Vector Machine

The most elementary supervised learning model is the **artificial neuron**. In its most significant formulation, the **Perceptron** proposed by Rosenblatt, the input vector x is multiplied by the weight vector w and the result of this operation is compared with a certain threshold θ. In this way the output of the neuron is determined.

From a geometrical point of view, the neuron is looking for a **hyperplane** that separates the point representing the "right users" from those representing the "left users". If these two sets of point are linearly separable, an infinite number of separating hyperplanes exists. Since the final aim of the training is to classify data that do not belong to the training set, it can be proved that the hyperplane with the largest margin (i.e. the distance from the closest points, called support vectors) is the optimal one. This is the idea behind one of the most used classification algorithms: **Support Vector Machine** (Figure 3.1).

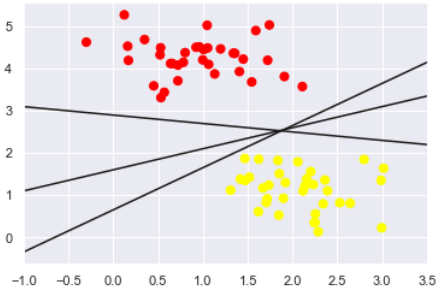


Figure 3.1: Examples of separating hyperplanes.

Sometimes the classes in analysis overlap, so the dataset does not allow any linear separation. To overcome this problem, the so-called **kernel methods** are used: they consist on projecting the data into a higher dimensional space, where a linear separation can be found. If it is not yet enough, a more general version of SVM, which tolerates violations of the separating hyperplane, can be adopted: **soft-margin SVM**. [9]

3.2.2 Decision Tree and Random Forest

A **decision tree** is an algorithm that classifies the samples by a recursive division of the dataset. This model can be represented graphically with a set of nodes, where the splits occur depending on the activating function of that node, obtaining in this way a tree-shaped graph which gives the name to this class of algorithm (Figure 3.2).

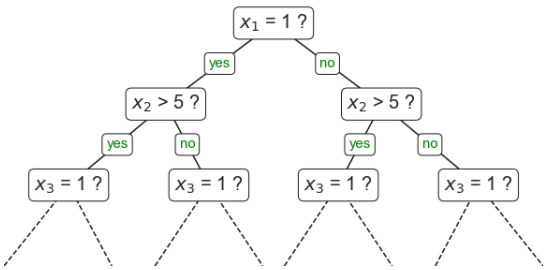


Figure 3.2: Example of a Random Tree.

The decision trees have a limitation concerning the depth of the tree (the number of divisions): if the dataset is divided just a few times, the classes obtained could still have elements with different labels. On the other hand, if we increase the complexity of the model, we could deal with problems of **overfitting**, i.e. the possible loss of the capacity of generalization, that is the application of the model on the test set. The solution consists on combining hundreds of decision trees, obtaining in this way a **Random Forest**.

3.2.3 Logistic Regression

Logistic Regression [10] is a 'Statistical Learning' technique categorized in Supervised Machine Learning methods dedicated to Classification tasks. The goal of binary Logistic Regression is to train a classifier that can make a binary decision about the class of a new input observation.

Basically, Logistic Regression measures the relationship between the dependent variable (our label, what we want to predict) and the one or more independent variables (our features), by estimating probabilities through it's underlying logistic function. Logistic Regression models the data using the **Sigmoid (logistic) function**:

$$g(x) = \frac{1}{1 + e^{-x}}$$

These probabilities must then be transformed into binary values, in order to actually make a prediction with discrete binary outcome between 0 and 1. Let's give a general usage schema of Logistic Regression, where it illustrates the steps that Logistic Regression goes through to give us our desired output (Figure 3.3).

Figure 3.3: Example of a Logistic Regression

We want to maximize the likelihood that a random data point gets classified correctly, which is called Maximum Likelihood Estimation: a general approach to estimating parameters in statistical models. We can maximize the likelihood using different optimization algorithms such as Newton's Method or Gradient Descent method, which can be used to find maximum (or minimum) of many different functions.

3.2.4 Stochastic Gradient Descent Classifier

Stochastic Gradient Descent (SGD) [11] is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. It has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Since it processes data in a sparse format, this classifier easily scale to problems with more than 10^5 training examples and more than 10^4 features.

The advantages of Stochastic Gradient Descent are:

- Efficiency
- Ease of implementation (lots of opportunities for code tuning)

The disadvantages of Stochastic Gradient Descent include:

- SGD requires a number of hyperparameters including the regularization parameter and the number of iterations
- SGD is sensitive to feature scaling

Stochastic gradient descent considers only 1 random point while changing weights unlike gradient descent which considers the whole training data. As such stochastic gradient descent is much faster than gradient descent when dealing with large data sets.

3.2.5 Multinomial Naive Bayes

The **Multinomial Naive Bayes** classifier is suitable for classification with discrete features (e.g., word counts for text classification). This model assumes the features come from a **multinomial** distribution: since we are dealing with just two possible classes, clearly, we are going to use a specific case of the multinomial distribution, which is the binomial distribution. The name of this classifier also suggests the idea behind its algorithm: in order to compute the probability of a certain label given some features, it uses the **Bayes** formula:

$$\frac{P\left(L_1|\text{ features }\right)}{P\left(L_2|\text{ features }\right)} = \frac{P\left(\text{ features }|L_1\right)}{P\left(\text{ features }|L_2\right)} \frac{P\left(L_1\right)}{P\left(L_2\right)}$$

Since we are making the (**Naive**) assumption of knowing the distribution of the features given a label, it is now straightforward to compute the desired likelihood.

Naive Bayes classifiers are often good baseline models that you can build off of to explore more sophisticated models. They are extremely fast and easily interpretable with the benefit of performing especially well on datasets with high-dimensionality.

3.2.6 XGBoost Classifier

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way.

The implementation of the algorithm was engineered for an efficient use of computing time and memory resources. A design goal was to make the best use of available resources to train the model. Some key implementation features include:

- **Sparse Aware** implementation with automatic handling of missing data values.
- **Block Structure** to support the parallelization of tree construction.
- **Continued Training** so that you can further boost an already fitted model on new data.

Boosting is an ensemble technique where new models are added to correct the mistakes made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the AdaBoost algorithm that weights data points that are hard to predict.

Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it actually uses a gradient descent algorithm to minimize the loss when adding new models. [12]

Chapter 4

Implementation and Results

4.1 FastText

FastText [13] needs labeled data to train the supervised classifier. **Labels** must start by the prefix _label_, which is how it recognizes what a label or what a word is. Below is an example of the required format for tweets with the label representing the political orientation: the value 0 corresponds to left, the value 1 corresponds to right.

_label_1 b taci zingaretti italiani dobbiamo combattere...
_label_0 rt maxmirandola sardine verona nonostante scel...

We will use only the dataframe with a labeling agreement larger than 75%, since it provides similar performances compared to the dataframe 100% of agreement. The dataframe is composed by 2370 rows: we'll take 80% of them as the training dataset and the remaining 20% of rows as the test data. As we already pre-processed our data, we can train a model via fastText with the following function:

```
import fasttext

model75 = fasttext.train_supervised(input="train75.txt",epoch=100)
model75.save_model("model_twitter75.bin")
```

Figure 4.1: The code used to train the model via FastText

Then we trained the model on training data, we fixed the parameter epoch=100 to go through the entire dataset 100 times and we use the unigram information. After we use the predict command to tell our model that we're going to do some predictions now. The binary file of the model, is what we'll use for the prediction.

```
model75.predict(test_df75.iat[1,1])
```

Figure 4.2: The code used to predict the political label given a user's text of tweets via FastText

Once trained, we need to measure how good our model is at predicting political orientation. For this, we can use the measure Precision which is the output of fastText function model.test.

```
model75.test("test75.txt")
```

Figure 4.3: The code used to obtain the accuracy of the model using FastText

Overall the model gives an accuracy of 67.5% on the test data. We can try to improve the performance of a model by using word bigrams, instead of just unigrams.

```
import fasttext

model75 = fasttext.train_supervised(input="train75.txt",wordNgrams=2,epoch=100)
model75.save_model("model_twitter75.bin")

model75.test("test75.txt")
```

Figure 4.4: The code used to train the model with additional bigram information via FastText

Now the accuracy of the model based on bigrams is 69.4 % on the test data. So, adding bigram information, the performance improves by 2%. We report the results of the 2 approaches used:

Model	Test accuracy
fastText	67.5%
fastText, bigram	69.4%

Table 4.1: Accuracy on the dataset with 75% of agreement using fastText

4.1.1 Classifing Movimento 5 Stelle

Now we want to investigate how the partecipants who vote for Movimento 5 Stelle are classified by the model. Using the trained fastText model, we are able to make predictions on labels that have not been seen during training.

```
dfmov75=pd.read_csv('dfmov75cl.csv').iloc[:,1:]
dfmov75.head(10)

M5_y_predict75=[]
for i in range(dfmov75.shape[0]):
 data= str(dfmov75.iloc[i,1]).replace('\n','')
 M5_y_predict75.append(model75.predict(data)[0][0])

M5_left75=M5_y_predict75.count('__label__0')/len(M5_y_predict75)
M5_right75=M5_y_predict75.count('__label__1')/len(M5_y_predict75)
print('M5S left: {}, M5S right: {}'.format(M5_left75,M5_right75))
```

Figure 4.5: The code used to classify Movimento 5 Stelle's users via FastText

In the simplier model with FastText the number of Movimento 5 Stelle's users classified as left are 67% and the number of Movimento 5 Stelle's users classified as right are 33%. If we add the bigram information, these percentages change in 72% and 28%.

4.2 Analysis with some ML models

Before starting the construction of the models, we have to split the dataset into two set: the first one, the **training/validation set**, will be used actually to train the models, the second one, the **test set**, will be used after the training phase to evaluate the accuracy of our model. Since the dataset is quite large, we opt to consider the 80% of the samples for the training set, and the remaining 20% for the test set. That large proportion dedicated to the training phase will ensure good modeling capacity, while the almost 500 samples of the test set should be enough to verify the quality of the model over users that will be different from each other.

In order to create these two sets, the simplest approach is to consider the first 80% of the rows as the training set, and the last 20% ones as the test set. This method has a big limitation: this split can be lucky or unlucky. The extreme consequence of this simplistic splitting is to collect only (for example) left users in the test set. This can lead to terrible results, since our model will be trained much better on right users but it will be tested only on left ones!

To solve this problem, we can use a cleverer tool provided by the library Scikit-Learn: train_test_split. Specifying some argument of this command we will have a random shuffling of the samples before the splitting. Moreover, we can use the stratify option which will keep the same proportion of right and left users in both the training and testing set (Figure 4.6).

```
train_df75, test_df75 = train_test_split(newdf75_all, test_size=0.2, random_state=0, stratify = newdf75_all['label'])

In the training set there are 1896 users: 984 right and 948 left
In the test set there are 474 users: 237 right and 237 left
```

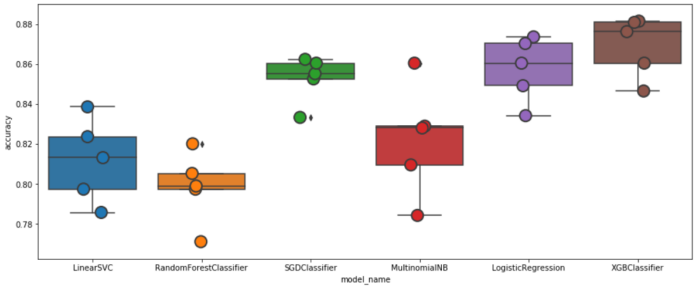
Figure 4.6: train_test_split implementation for the dataset with 75% of agreement

Now we can proceed with the text representation for the tweets column using the **bag of words** approach. The two tools considered are Count Vectoriser and TD-IDF Vectorizer which have already be presented in the section 3.1. As a result of this procedure, the tweets columns will be replaced with thousands of numerical features: the preferred encoding of the machine learning classifiers that will be use.

The models considered are Linear SVC, Random Forest, SGD Classifier, Multinomial NB, Logistic Regression, all available in the Scikit-Learn package. In addition, XGB Classifier has been taken into account.

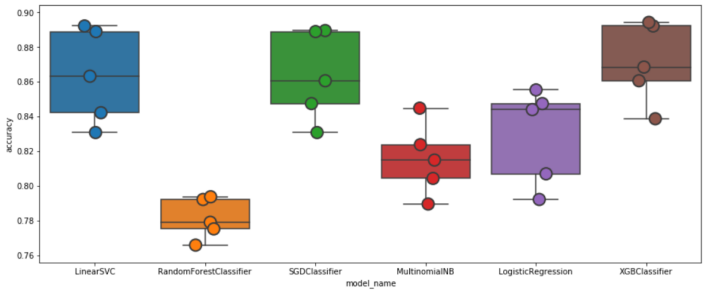
In order to have a first idea of the performance we can reach, we trained all these models both with the Count Vectorizer and TF-IDF outputs. The scoring index we considered in this very first part is the **5-folds cross validation accuracy**: this procedure consists on splitting the dataset into 5 parts. Each model will be fitted in 4 of these parts and evaluate in the remaining one, computing the accuracy level. This fit-evaluating process will be repeated for each possible choice of the evaluating set. Then by computing the mean of these five values we will have a score for the model.

The results are presented in Figure 4.7. Pay attention with one aspect: these scores are not an index of the general performance of the models! They should be used just to compare the different classifiers. After the model selection and the tuning phase, we will provide the accuracy results of the models in the test set, which has not been used up to now.



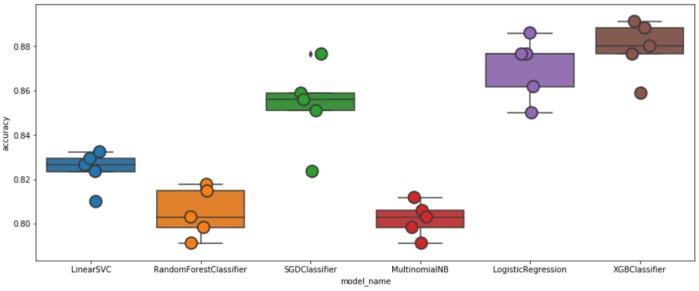

```
model_name
LinearSVC                0.811710
LogisticRegression       0.857600
MultinomialNB            0.822250
RandomForestClassifier   0.798546
SGDClassifier            0.852838
XGBClassifier            0.869187
Name: accuracy, dtype: float64
```

(a) *Word Vectorizer + agreement* $\geq$ *75%*



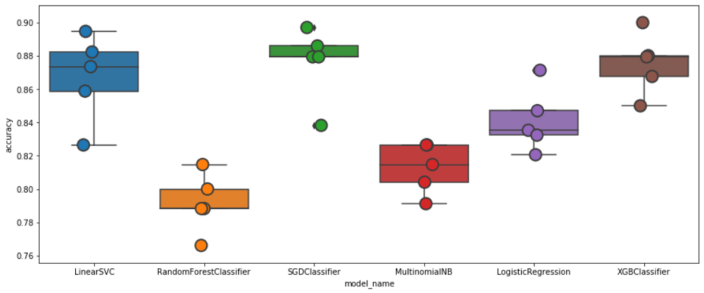

```
model_name
LinearSVC                0.863389
LogisticRegression       0.829106
MultinomialNB            0.815388
RandomForestClassifier   0.781125
SGDClassifier            0.863389
XGBClassifier            0.870771
Name: accuracy, dtype: float64
```

(b) *TF-IDF Vectorizer + agreement* $\geq$ *75%*




```
model_name
LinearSVC                0.824341
LogisticRegression       0.870134
MultinomialNB            0.802002
RandomForestClassifier   0.804943
SGDClassifier            0.853117
XGBClassifier            0.878965
Name: accuracy, dtype: float64
```

(c) *Word Vectorizer + agreement = 100%*




```
model_name
LinearSVC                0.867183
LogisticRegression       0.841328
MultinomialNB            0.812583
RandomForestClassifier   0.791452
SGDClassifier            0.876017
XGBClassifier            0.875435
Name: accuracy, dtype: float64
```

(d) *TF-IDF Vectorizer + agreement = 100%*

Figure 4.7: Comparing the performance of the models in the evaluating set

Analyzing these numbers, we can make two important considerations:

1. the models perform better if they are constructed over the features created by TF-IDF. This is something we could have expected: while world vectorizer simply counts the frequency of a specific word in a certain text, TF-IDF takes into account also the frequency of that word in the other texts. Since we are looking for the elements that discriminate one class from another, it is clear that it is more useful to understand which are the words that are used a lot by a class, and not by the other one.

2. The models constructed on the dataset with an agreement percentage of 100% perform better that the ones built on the 75% agreement dataset. Probably this is due to the fact that, using the 75% agreement dataset, even though we are considering a larger dataset, we are introducing some samples with a less clear political orientation. For an algorithm this constitutes noise.

For these reasons, starting from now we will consider only TF-IDF for the text analysis and the dataset with the agreement level of 100%.

As a final step of the training-evaluating phase, we will **optimize the hyperparameters** of the classifiers. To this end, one of the most traditional procedure is the **Grid Search** [14]. Roughly, it consists on considering different values for each hyperparameter and train the model with all their possible combinations. [15]

To give some practical in-sights, we validated the grid search with a 5-folds cross validation in the training validation set and we set the argument $\mathsf{n\_jobs} = -1$ in order to exploit all the cores of our machines. Then, by using the best_params_ method of the grid search object, we obtained the combination of hyperparameters which provide the higher precision. Finally, we used the predict method on the test set (neglected so far) in order to have the accuracy index of each model.

In Figure 4.8 it is shown the code used to optimize the hyperparameters of SVC.

```
# --- SVC hyperperameters optimization ---
# create the grid with an exponential sequence of values for each hyperparameter
param_grid = {'C': [2, 4, 8, 16, 32, 64, 128],'gamma': [2**-5, 2**-4, 2**-3, 2**-2, 2**-1]}

grid_search = GridSearchCV(estimator = SVC(), param_grid = param_grid, cv = 5, n_jobs = -1, verbose = 2) # build the grid

grid_search.fit(X_train_t, y_train) # fit

best_grid = grid_search.best_estimator_ # Use the optimal hyperparameters

predictions = best_grid.predict(X_test_t), accuracy_score(y_test, predictions) # compute the accuracy using the test set
```

Figure 4.8: The code used to optimize the hyperparameters of the models

This procedure has been carried out considering the most relevant hyperparameters of each model.

In Table 4.2 the accuracy result of each model on the test set is shown.

Classifier	Hyperparameters	Accuracy
Support Vector Classifier	C = 1 gamma = 0.25	86.2 %
Logistic Regression	C = 128 penalty = l2	86.9 %
Stochastic Gradient Descent	alpha = 1e-06 penalty = l2	86.2 %
Random Forest	max_depth = 50 n_estimators = 1000	85.7 %
Naive Bayes	alpha = 1e-05	84.7 %
XGBoost Classifier	auto	88.1 %

Table 4.2: Accuracy level of the classifiers after the optimization.

4.2.1 Classifing Movimento 5 Stelle

As a further step, we can try to test our models on the samples labeled as Movimento 5 Stelle. Anyway, this is not straightforward as it may seem. Indeed, several adjustments are required. First of all, we created a dataset with the users grouped into 3 classes: right, left and M5S. The original class distribution is shown in Figure 4.9: [1]

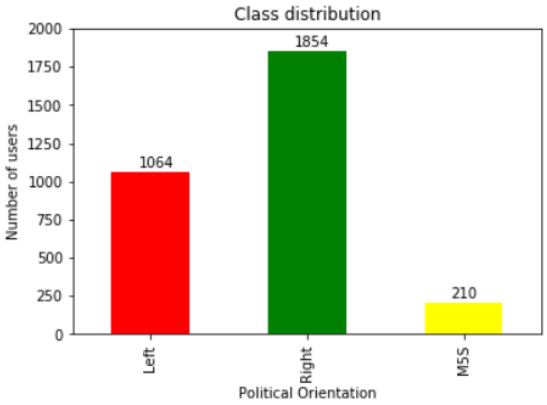


Figure 4.9: Class distribution of the new dataset

After having cleaned the data, in the same way as before, we balanced the right and left class. This is a crucial step, since only the users of these two classes will be involved in the training phase, while the users labeled m5s will be considered only in the test phase.

Now we have to represent in a numerical way the tweets. We will use only TF-IDF since it provides better performances compared to Count Vectorizer. Moreover, it is important to train the word vectorizer tool using the complete dataset: if we had fit it only with the tweets of right and left users, we would have found problems when it comes to transform

[1] for this part of the analysis we used only the users labeled with an agreement of 100%.

the tweets of M5S users, since the model may encounter new words, without knowing how to convert them. In this way, all the users will have the same features even though they use different words in their tweets. This is absolutely vital: only in this way we will be able to apply the model trained on right and left users over M5S users.

For the sake of completeness, in Figure 4.10 we report the code used for this purpose. It is possible to see that the transformed datasets (right-left users and M5S users) have the same number of features: 479 028.

```
from sklearn.feature_extraction.text import TfidfVectorizer # import TF-IDF that will be used

tf = TfidfVectorizer() # build the model

all_users_t = tf.fit_transform(all_users) # fit the model in the whole dataset
```

```
right_left_t = tf.transform(right_left) # transform the right-left dataset
```

```
<2128x479028 sparse matrix of type '<class 'numpy.float64'>'
        with 6276267 stored elements in Compressed Sparse Row format>
```

```
m5s_t = tf.transform(m5s) # transform the m5s dataset
```

```
<210x479028 sparse matrix of type '<class 'numpy.float64'>'
        with 836762 stored elements in Compressed Sparse Row format>
```

Figure 4.10: The code uses for the text rapresentation.

Finally, we can consider the classifiers which gave better result in the previous part of the analysis, train them on the transformed right-left users dataset, and then see how they label the M5S users. As we could have expected, they are almost equally distributed over the two classes (Figure 4.11).

	model_name	users predicted as right	users predicted as left
0	LinearSVC	0.528571	0.471429
1	SGDClassifier	0.533333	0.466667
2	LogisticRegression	0.504762	0.495238
3	XGBClassifier	0.542857	0.457143

Figure 4.11: Results of the classification of M5S users.

4.3 Conclusion

To conclude, the results of this project can be summed up in the following points:

- using FastText with the epoch parameter set equal to 100, we created a model with an accuracy level of 67.5%. By adding the bigrams information, we managed to increase the precision up to 69.4%;
- using the dataset with 100% of agreement and TF-IDF as text rapresentation tool, we built some classifiers with a quite good accuracy: after optimizing the hyperparameters of the models through a Grid Search, the best performances were reached by XGBoost Classifier (88.1% of accuracy), Logistic Regression (86.9% of accuracy), Support Vector Classifier and Stochastic Gradient Descent Classifier (both 86.2% of accuracy);
- as far as it concerns Movimento 5 Stelle's users classification, FastText labeled around 70% of them as left users, while the other models distributed them almost equally in the two political orientations.

These results have already been commented individually in the previous sections. Now let's just make some further observations and comparisons: first of all, it is not so surprising that FastText reaches a lower accuracy level compared to the other Machine Learning models. Indeed, there are lots of examples where simple models overcome the performances of more complex ones, depending on the context. In this specific case, the dataframe used with the classifiers was characterized by an huge number of features (between 400000 and 500000) and as a consequence, the models had already an high complexity from the beginning. Therefore, if we build an elaborate model over such a dataset, the complexity reaches even higher levels, which is not always an effective way to approximate a certain phenomenon.

Lastly, it is reasonable to obtain an almost equal distribution of Movimento 5 Stelle users in the two classes, since basically they propose themselves as an alternative to the ordinary political parties. However, it is interesting to see that FastText is a little skewed towards the left users: actually, according to some political commentators, the Movimento's ideology is closer to the left wing.

Bibliography

[1] Praet et al., *I like, therefore I am. Predictive modeling to gain insights in political preference in a multi-party system*, 2018.

[2] Tumasjan et al., *Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment*, 2010.

[3] Preot,iuc-et al., *Beyond Binary Labels: Political Ideology Prediction of Twitter Users*, 2010.

[4] https://github.com/twintproject/twint

[5] https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

[6] http://datameetsmedia.com/bag-of-words-tf-idf-explained/

[7] https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3

[8] https://fasttext.cc/blog/2016/08/18/blog-post.html

[9] J. VanderPlasp, *The Python Data Science Handbook*, O'Reilly, 2016.

[10] https://towardsdatascience.com/logistic-regression-classifier-8583e0c3cf9

[11] https://scikit-learn.org/stable/modules/sgd.html

[12] https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

[13] https://towardsdatascience.com/fasttext-sentiment-analysis-for-tweets-a-straightforward-guide-9a8c070449a2

[14] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, *A Practical guide to Support Vector Classication*, Department of Computer Science National Taiwan University, Taiwan, 2016.

[15] T. Boyle, *Hyperparameter Tuning*, in towardsdatascience.com, 16 Febbraio 2019.