POLITECNICO DI TORINO
Mathematical Engineering
Master's Degree

Master's Thesis

Sentiment analysis using ensemble methods: an application to Twitter

Supervisor:
Prof. Roberto Fontana

Author:
Gianluca Fino

April 2019

To my family

Sentiment analysis using ensemble methods: an application to Twitter

Abstract

This work mainly concerns the sentiment analysis (an application of natural language processing) and the ensemble methods. The first part of the thesis shows how data are collected from Twitter (with a procedure called scraping) and the way in which a data set is built with the frequency of the words most used by nine Italian politicians. After that, using SAS software, a descriptive analysis and a study of language similarity among some of the Italian politicians are proposed. This part, developed during the internship period, inspired the sentiment analysis and the rest of the study, concerning text mining and classification. Some of the most famous algorithms used in these fields and the evaluation of their performance are reported. After that, the focus is on ensemble methods, which combine the previous algorithms in order to achieve better results. Sentiment analysis is explained in detail, with particular attention to the branch of human emotions. An application is proposed: some tweets, collected from a famous Twitter profile, are labelled with an emotion using the ensemble methods. The results obtained show how these methods are able to achieve good performance in this context.

Keywords: sentiment analysis, ensemble methods, classification, machine learning, Python, Twitter

Acknowledgements

Thanks to my family and my girlfriend, for always been close to me in difficult times.

My friends for making me live these years with lightheartedness.

My classmates for sharing this experience in the best possible way.

My colleagues for introducing me to the world of work.

No pain no gain.

Contents

List of Figures

List of Tables

Introduction

Internet is a huge source of information and web scraping is the easiest way to get data from it. Nowadays, social networks play a fundamental role in the dissemination of information. For example, in the 2016 American election Twitter was the largest source of breaking news. In this context, text mining has become a topic of particular interest. It is the process of deriving high-quality information from text and it is strictly correlated to sentiment analysis.

My idea was initially to develop a project in Python to become familiar with scraping and subsequently to analyze the obtained data with SAS [1].

Scraping can be done through the use of application programming interface (API). In computer science, API indicates every set of procedures available to the programmer, collected together to provide a series of specific tools for the realization of a certain task. The first part of the study concerns the analysis of the words of the most recent tweets written by nine politicians representing the main political forces of Italy.

Tweets are usually the expression of a subjective context rather than an objective one. This is the starting point of the study and the reason why I decided to perform an emotion analysis. It is a type of sentiment analysis that refers to the detection of human feelings. With the purpose of doing it as precisely as possible, I used what are commonly called *ensemble methods*. They are a powerful machine learning technique, developed in the last years and widely used in classification problems.

My work is composed by five chapters which naturally constitute three parts. The first one concerns what I introduced before about scraping and his application on Italian politicians. The second one reviews some popular classification models and it is very useful to understand the State-Of-The-Art of machine learning algorithms. After that it focuses on ensemble methods that, in contrast to ordinary learning approaches, try to construct a set of learners and combine them. The last part is an application of emotion analysis and it uses the methods above on real data to underline the power of this ensemble techniques.

Chapter 1 describes the main steps in which data are collected and how they are used to create a data set with the frequency of words used by each politician. After that, using the data set previously created, an analysis is carried out in SAS. It shows a visualization of the results in the form of graphs and tables and a study of language similarity among the Italian politicians.

In Chapter 2 some of the most important methods of classification in machine learning and the ways to evaluate their performance are shown.

[1] SAS is a statistical software developed in C by SAS Institute.

In Chapter 3 these methods are combined to increase their accuracy. Some different ways of doing it are shown and they all are a particular type of ensemble methods. To train my models I used a data set available online, with 7666 tweets and 7 different emotions.

In Chapter 4 sentiment analysis is proposed. It refers to the use of natural language processing, textual analysis and computational linguistics to extract information and associate it with a label. In my case, tweets are classified with an emotion like joy, sadness, etc...

Finally, Chapter 5 contains the conclusions.

Chapter 1

Tweets of Italian politicians

The first chapter shows the way in which information are retrieved through Twitter [1], the baselines of the script made in Python language and a descriptive analysis using SAS.

The politicians involved in the study are Berlusconi, Bonino, Di Maio, Grasso, Grillo, Maroni, Meloni, Renzi and Salvini. To get their tweets, scraping (a procedure for extracting data from websites) is used.

In section 1.1 the structure of the program is shown. In section 1.2 the connection to Twitter application is explained. Section 1.3 introduces some notions of text mining used in Italian languages. In the next section the output of the main functions developed and the creation of data frame that will be imported in SAS are shown. Then a visual description of the most used words is given, through a word cloud representation (section 1.4.3). Finally, a descriptive analysis and a correlation study between politicians are proposed.

1.1 Program overview

In figure 1.1 the conceptual scheme of the program is shown. First of all, through Twitter API the connection to the famous social network is established. After that for each politician tweets are downloaded, cleaned according to the rules of natural language, split into words, counted their frequency and put them in a vector. A data set with these vectors is created and written on a text file. This process is repeated for several days until it is decided to stop the scheduling. Every day, at the same time, the script runs, downloads the new tweets and adds them to the previous. Scheduling is used to improve the number of tweets available (Twitter imposes a limit to the scraping, allowing the download of the two hundred most recent for profile).

Subsequently the updated data set is loaded in SAS. In the following sections every step of this process is explained in detail. The Python code can be divided into the following sections:

[1] Twitter is a social network, created on 2006 by Obvious Corporation of San Francisco, on which users post and interact with messages known as "tweets" with a maximum length of two hundred and eighty characters. The name "Twitter" comes from the English verb to tweet which means "chirp." Tweet is also the technical term of the service updates and it can be done through the site itself, via SMS, with instant messaging programs, e-mail, or through various applications based on Twitter's application programming interface (API).

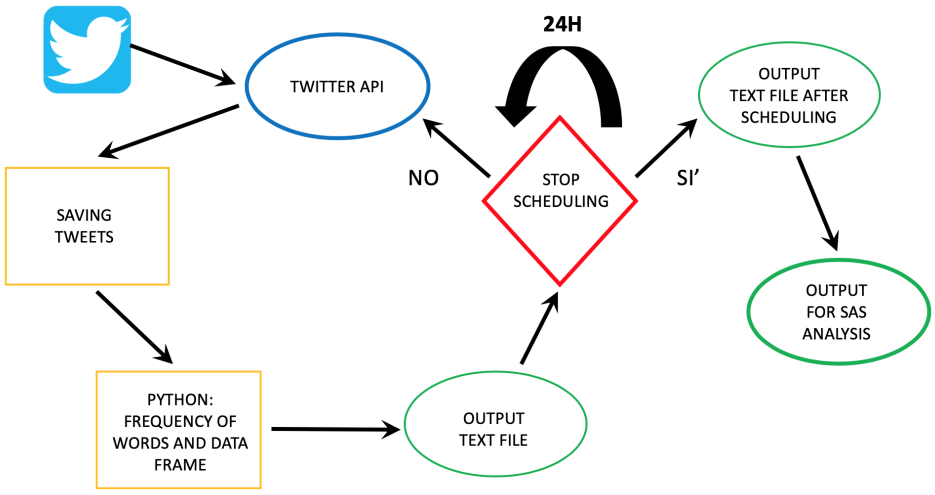


Figure 1.1: *Conceptual scheme*

- Import of libraries needed and definition of some variables.
- Download of tweets through Tweepy library and its API service.
- Definition of functions used.
- Text mining on tweets.
- Creation of a data frame, thanks to Pandas library, with n-rows and nine columns, having for each politician the frequency of all words used.
- Creation of text files.
- Use of the Wordcloud library to create an image containing the most used words.

1.2 Twitter API

The first step to connect on Twitter application and to use the API service is to get four secret keys (figure 1.2). Subscription is required and it is available on platform, at link https://apps.twitter.com/. Entering personal data as explained in the website tutorial makes free connection possible.

Once the access credentials have been obtained, the program can be implemented in python. First of all, download of Tweepy library is a possible choice to easily use the keys. After that the modules "OAuthHandler" and "API" are used, in order to connect to the site and create an object called "api" (figure 1.3). This object has some functions like *user_timeline* or *search*, which make it easy get tweets from a personal page (first function), or about a topic (the second one).

A tweet is a message with a maximum length of two hundred and eighty characters. After the download a json form is obtained and it has to be analyzed

to get the necessary information, such as the full text of the tweets and if is a retweet or not.

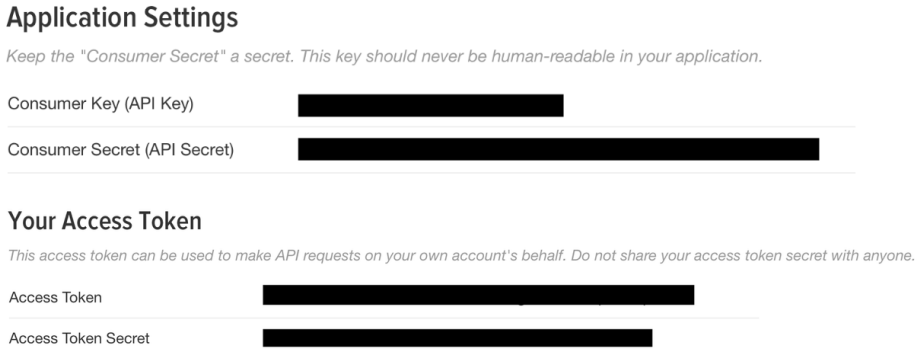


Figure 1.2: *Twitter credentials*

```
#Credenziali dell'applicazione Twitter
consumer_key='[redacted]'
consumer_secret='[redacted]'
access_token='[redacted]'
access_secret='[redacted]'

#Api di twitter
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
api = tweepy.API(auth)
```

Figure 1.3: *Python code for Twitter API*

1.3 Text analysis on tweets

Once the tweets have been downloaded, some data cleaning procedures are used. In this case tweets are sentences in Italian language and some procedures of natural language processing available in English (they will be explained in detail in 4.1), like "stemming", are not commonly used in Italian. It reduces every name or verb to its word stem (for example "playing", "plays" and "played" are considered as the same word "play").

Italian grammar is more complicated and cannot be used easily in this context. One option for doing this is to manually create a dictionary in which every of the most common words refers to its root, such as its infinite form. The project does not get a lot of advantages from the previous technique and so "stemming" is not used. A procedure that can be used in every language is the elimination of stop words [1]. They are some commonly used words (such as "the") that the program ignores, because they are meaningless and are not useful for analysis. They are all included in a list of words, called stop list, which is not universal but depends on the natural language processing tools used. A common choice in Python script is NLTK library.

```
#Nomi dei politici censiti
dictPolitici = {
    'B' : '@berlusconi',
    'BO' : '@emmabonino',
    'DM' : '@luigidimaio',
    'GR' : '@PietroGrasso',
    'G' : '@beppe_grillo' ,
    'MA' : '@RobertoMaroni_',
    'M' : '@GiorgiaMeloni',
    'R' : '@matteorenzi',
    'S' : '@matteosalvinimi'
}
```

Figure 1.4: *Dictionary of censored politicians*

```
37 print(funcStatusHash(dictPolitici['B']))
```

```
['7f47607ace64820c0e096ffdca77a630', '148bcfed40468e596a5d17fe5ed40120', '8b47beb9
6cacc6f05ad40a5f68df36b4d', 'efe5ab48bd5cb4763bc6ea3fcc644c7d', '47c409d3eb25775c3
5cb6254eb87b335c', '6a5cd2c1887aa4098062ac5e3ff80732', '62b8035b71439a5811be901c9f
49dcf42', 'cb1cdc8e21864d234b545ee952938915', '4a5d20187580d76f49ba4da1add57dcf',
'ed82afa15bfe8106f47048b0804ea65e', 'b0f8516c5464b9e137dbb7fe52a051ac', 'f44e1bd61
7245844f63fad61130690f51', 'fa5f643038a768c3181d257fab67bfd1', '9a55d55ad6a7835553
```

Figure 1.5: *Output of function "funcStatusHash"*

Another procedure, that is very simple to implement, exploits re library in order to use regular expression to remove punctuations, emoji and other special character (more details in 4.1).

1.4 Main functions and output

1.4.1 Main functions

In this section the most important variables and functions are shown. First of all, a dictionary called "DictPolitici" is defined, which makes more practical the connection to politician's profile (figure 1.4). After that and the definition of other variables, some functions are developed.

One of the most interesting is "funcStatusHash", that is used to obtain an identifier for each tweet. This very big number (more than 30 digits), called hash, is a unique code of fixed length in hexadecimal format that is randomly reproduced. It is an alternative to the tweet ID and it can be very useful in case in which it is not easy to obtain an identifier. A portion of result is shown in figure 1.5. When a comparison between old and new tweets is done, the hash is used to choose which ones can be add to the list. This happens because, as it is mentioned before, the program is able to obtain the last 200 tweets, regardless of the number of the most recent ones.

After that, "funcStatusSingle" applied to Renzi's account is presented (figure 1.6). The sentences are split into words, with the hash number at first. There are many prepositions and punctuation marks, which will be deleted before the counting of frequency.

The so-called stop words of the Italian language are removed, in addition to

```
94 print(funcStatusSingle(dictPolitici['DM']))
```

```
['[', '(', '462288796349031255996693586483856349 71', ')', 'toti', 'vuol', 'far', 'ricc
di', 'ad', 'autostrade', '?', 'lo', 'dica', 'alle', 'famiglie', 'delle', 'vittime', '.
, 'dovrà', 'essere', 'un', "'", 'azienda', 'di', 'stato', '.', ']', '[', '(', '1334659
2', ')', 'sto', 'seguendo', 'con', 'la', 'massima', 'apprensione', 'ciò', 'che', 'è',
, 'che', 'si', 'profila', 'come', 'immane', 'tragedia', '.', 'siamo', 'in', 'stretto',
', 'e', 'stiamo', 'andando', 'sul', 'luogo', 'con', 'il', 'viceministro', 'rixi', '.',
nza', 'in', 'queste', 'ore', 'alla', 'città', ']', '[', '(', '248940886713081287381831
', 'dignità', ',', '@caritasroma', 'esprime', 'apprezzamento', 'per', 'le', 'misure',
```

Figure 1.6: *Output of function "funcStatusSingle"*

t

```
59 print (funcWords(dictPolitici['S']))
```

```
[('#salvini', 36), ('italia', 30), ('italiani', 19), ('amici', 17), ('#genova', 17), (
15), ('oggi', 15), ('grazie', 15), ('fatti', 14), ('genova', 14), ('anni', 13), ('dopo
', 13), ('diretta', 13), ('ora', 12), ('senza', 12), ('parole', 11), ('avanti', 11), (
'europa', 11), ('paese', 11), ('#inonda', 11), ('lavoro', 10), ('mila', 10), ('#nonsto
('prima', 9), ('altri', 9), ('ancora', 9), ('cittadini', 9), ('#rairadio1', 9), ('fuoc
8), ('può', 8), ('stato', 8), ('casa', 8), ('ieri', 8), ('bene', 8), ('#zapping', 8),
), ('vigili', 8), ('pagare', 8), ('anno', 7), ('#primagliitaliani', 7), ('fare', 7), (
i', 7), ('lavorando', 7), ('essere', 7), ('persone', 7), ('già', 7), ('autostrade', 7)
```

```
59 print (funcWordsApp(dictPolitici['S']))
```

```
[('#salvini', 36), ('italia', 30), ('italiani', 21), ('amici', 17), ('#genova', 17), ('s
), ('oggi', 15), ('immigrati', 15), ('live', 14), ('ora', 14), ('fatti', 14), ('genova',
', 13), ('anni', 13), ('diretta', 13), ('senza', 12), ('tutta', 11), ('voglio', 11), ('e
'parole', 11), ('avanti', 11), ('#inonda', 11), ('lavoro', 10), ('mila', 10), ('#nonstop
, ('prima', 9), ('altri', 9), ('ancora', 9), ('cittadini', 9), ('#rairadio1', 9), ('fuoc
8), ('stato', 8), ('casa', 8), ('ieri', 8), ('bene', 8), ('fare', 8), ('#zapping', 8), (
('qualcuno', 8), ('vigili', 8), ('pagare', 8), ('#primagliitaliani', 7), ('cosa', 7), ('
vorando', 7), ('essere', 7), ('anno', 7), ('persone', 7), ('già', 7), ('autostrade', 7),
```

Figure 1.7: *Output of function "funcWords"*

other words of little interest such as "rt", "via" and "http".

The function "funcWords", through collections library, counts the frequency of words. As can be seen in figure 1.7 words are shown with their frequency, in descending order (Salvini's official account is used as example). The function with suffix "App" shows the counting after scheduling. For instance the word "italiani", is quoted two more times after one day.

Each day the most recent tweets are added to the previous two hundred and, according to the same procedures, the frequency is updated. The words stored can increase a maximum of fifty per day, in order to allow the inclusion of new words and not lose the frequency of those already considered. To better understand the problem let us make a practical example, setting the number of words considered to one hundred and fifty. The word "world", on the second day, is one hundred and fiftieth place in the ranking of the most used with a frequency of three. The next day a new word is quoted four times; this exceeds "world" in the rankings, which is no longer considered causing a loss of information. Therefore, the increase in the number of words examined is necessary.

	@berlusconi	@emmabonino	@luigidimaio	\
#4marzo	NaN	NaN	5.0	
#80euro	NaN	NaN	NaN	
#agorarai	9.0	NaN	NaN	
#aosta	8.0	NaN	NaN	
#autonomia	NaN	NaN	NaN	
#barbari	NaN	NaN	NaN	
#berlusconi	20.0	NaN	NaN	
#bersagliomobile	11.0	NaN	NaN	
#byebyevitalizi	NaN	NaN	26.0	
#centrodestra	32.0	NaN	NaN	
#consultazioni	6.0	NaN	NaN	
#conte	NaN	NaN	NaN	
#convergisulvincolo	NaN	NaN	6.0	
#decretodignità	NaN	NaN	6.0	
#diciotti	NaN	NaN	NaN	
#doppiamorale	NaN	NaN	NaN	
#elezioni2018	NaN	NaN	NaN	
#fateliscendere	NaN	NaN	NaN	

Figure 1.8: *Data frame Pandas*

1.4.2 Data frame

A data frame can be considered as a matrix in $\mathbb{R}^{n,m}$ composed by n-rows (observations) and m-columns (variable). In this project the first column shows the words in exam while the following nine represent, for each politician, the frequency of these words. Few lines and four columns are shown in figure 1.8, in which "/" indicates that there are more columns in Pandas data frame which are not plotted, because of Jupyter Notebook visualization limit. Once the data frame has been created the phase of scraping and data manipulation is over. At this point it is possible to write this information on a text file, written with ";" after each word or frequency, in order to subsequently import a table in SAS. In figure 1.9 some lines of text file, opened in Microsoft Excel, are shown.

	A	B	C	D	E	F	G	H	I	J	K
1	Parole	Berlusconi	Bonino	Di Maio	Grasso	Grillo	Maroni	Meloni	Renzi	Salvini	Totale
2	#4marzo	0	0	5	0	0	0	0	0	0	5
3	#80euro	0	0	0	0	0	0	0	11	0	11
4	#agorarai	9	0	0	0	0	0	0	0	10	19
5	#aosta	8	0	0	0	0	0	0	0	0	8
6	#autonomia	0	0	0	0	0	5	0	0	0	5
7	#barbari	0	0	0	0	0	13	0	0	0	13
8	#berlusconi	20	0	0	0	0	0	0	0	0	20
9	#bersagliomobile	11	0	0	0	0	0	0	0	0	11
10	#byebyevitalizi	0	0	26	0	0	0	0	0	0	26
11	#centrodestra	32	0	0	0	0	0	0	0	0	32
12	#consultazioni	6	0	0	0	0	0	0	0	0	6
13	#conte	0	0	0	0	0	0	0	6	0	6
14	#convergisulvincolo	0	0	7	0	0	0	0	0	0	7
15	#decretodignità	0	0	6	0	0	0	0	0	0	6
16	#diciotti	0	0	0	0	0	0	0	14	6	20
17	#doppiamorale	0	0	0	0	0	0	0	7	0	7
18	#elezioni2018	0	0	0	11	0	0	0	0	0	11
19	#fateliscendere	0	0	0	0	0	0	0	11	0	11
20	#favoletta	0	0	0	0	0	0	0	7	0	7
21	#fdi	0	0	0	0	0	0	25	0	0	25
22	#fiduciaconte	0	0	0	5	0	0	0	0	0	5
23	#flattax	0	0	0	0	0	0	0	6	0	6
24	#forzaitalia	97	0	0	0	0	0	0	0	0	97
25	#genova	0	0	0	0	0	0	0	10	17	27

Figure 1.9: *First lines of the text file opened in Microsoft Excel*

1.4.3 Word cloud

Word cloud is a visual representation of text data, typically used to depict the most important keywords on websites. In this case it is used to represent the most frequency words, as shown in figure 1.10. A figure, as input of the word cloud, is needed. A usual choice can be a colored circle in a white background. The libraries used in Jupyter Notebook are Wordcloud and PIL. It should be noted that the python interpreter of computer may not be able to use them, giving an error, so this part of code must be commented during the scheduling.

1.5 Descriptive analysis in SAS

This section shows the results obtained with SAS. After a first step of data import [2], section 1.5.1 shows some results about the count of tweets and words. Then, a correlation matrix between the politicians in question is created and some figures are shown (section 1.5.2).

1.5.1 Frequency of words

In the tables 1.1 and 1.2 the most quoted words, on the first day and after a week of scheduling, are reported. Some words, like "#forzaitalia", "@emmabonino" and "@pietrograsso" are excluded, as personal references of a single politician. It can be noted how the word "thank", for example, was not present on the first day of scheduling because it did not reach sixty citations, while it is after seven days because it is mentioned seventy-five times in total.

Two bar charts are shown below, representing the total frequency of words in table 1.2 (figure 1.11) and the number of tweets posted by the politicians in the last seven days (1.12).

[2] This data refers to the period of time between 09/05/2018 and 03/06/2018.

Oss	Parole	Berlusconi	Bonino	Di_Maio	Grasso	Grillo	Maroni	Meloni	Renzi	Salvini	Totale
1	italia	41	20	17	19	13	7	55	16	27	215
2	governo	33	0	41	5	0	14	41	13	30	177
3	oggi	18	21	18	9	9	23	25	27	19	169
4	lavoro	10	14	9	35	0	6	20	22	7	123
5	paese	22	14	10	20	8	0	0	13	7	94
6	italiani	13	0	7	6	5	0	24	6	24	85
7	europa	20	35	0	0	0	0	10	7	10	82
8	anni	15	8	3	11	8	7	8	12	9	81
9	ogni	6	0	4	21	19	5	9	9	8	81
10	solo	11	24	5	8	13	4	0	8	7	80
11	voto	10	16	26	0	0	5	0	7	14	78
12	presidente	17	7	13	9	0	3	17	5	6	77
13	ora	8	11	0	0	8	8	9	11	21	76
14	sempre	6	11	4	13	17	7	11	7	0	76
15	essere	11	14	4	10	15	0	7	5	8	74
16	stato	21	12	6	6	11	0	7	8	0	71
17	fare	8	9	0	5	8	0	8	13	16	67
18	futuro	19	12	6	12	0	4	0	5	8	66
19	stelle	9	0	44	0	0	0	0	12	0	65
20	grande	18	7	0	0	13	15	6	5	0	64
21	fatto	11	13	6	0	10	4	0	19	0	63
22	politica	15	20	5	13	0	4	0	6	0	63
23	prima	7	9	7	7	5	0	11	8	9	63

Table 1.1: *Words quoted more than sixty times*

Oss	Parole	Berlusconi	Bonino	Di_Maio	Grasso	Grillo	Maroni	Meloni	Renzi	Salvini	Totale
1	italia	41	20	17	19	13	7	61	17	28	223
2	governo	33	4	41	5	0	17	44	13	31	188
3	oggi	18	21	18	10	10	24	26	27	20	174
4	lavoro	10	14	9	35	2	6	22	22	8	128
5	paese	22	14	10	20	10	2	0	13	8	99
6	anni	16	8	3	12	9	13	9	12	9	91
7	italiani	13	4	7	6	3	0	25	6	25	89
8	ogni	6	5	4	21	21	5	10	9	8	89
9	solo	11	24	5	8	16	4	3	8	9	88
10	voto	10	16	26	4	0	5	3	7	17	88
11	ora	8	11	2	2	9	8	9	11	26	86
12	europa	20	35	1	0	0	0	10	7	12	85
13	essere	11	14	4	10	21	0	7	5	9	81
14	sempre	6	11	4	13	17	7	11	7	5	81
15	grande	18	7	3	2	17	18	6	5	3	79
16	stato	21	12	6	6	13	0	7	8	5	78
17	presidente	17	7	13	9	0	3	17	5	6	77
18	ecco	2	2	10	4	38	8	4	3	5	76
19	futuro	19	12	6	12	5	4	4	5	9	76
20	grazie	15	8	9	4	5	3	2	9	20	75
21	fatto	11	13	6	0	14	4	4	19	2	73
22	politica	15	20	5	13	3	5	4	6	2	73
23	fare	8	9	3	5	8	0	8	13	16	70
24	stelle	9	0	44	0	0	0	2	12	2	69
25	prima	7	9	7	7	5	0	11	8	10	64

Table 1.2: *Words quoted more than seventy times after seven days of scheduling*

Figure 1.10: *Word cloud obtained with the homonymous library*

It can be noted that after seven days the amount of data has increased: number of tweets written by Berlusconi, Bonino, Di Maio, Grasso, Grillo, Maroni, Meloni, Renzi e Salvini are respectively at 202, 208, 202, 206, 217, 215, 216, 209 and 232 for a total of 1907 compared to the previous 1800, reporting an increase of about 6% in the number of tweets.

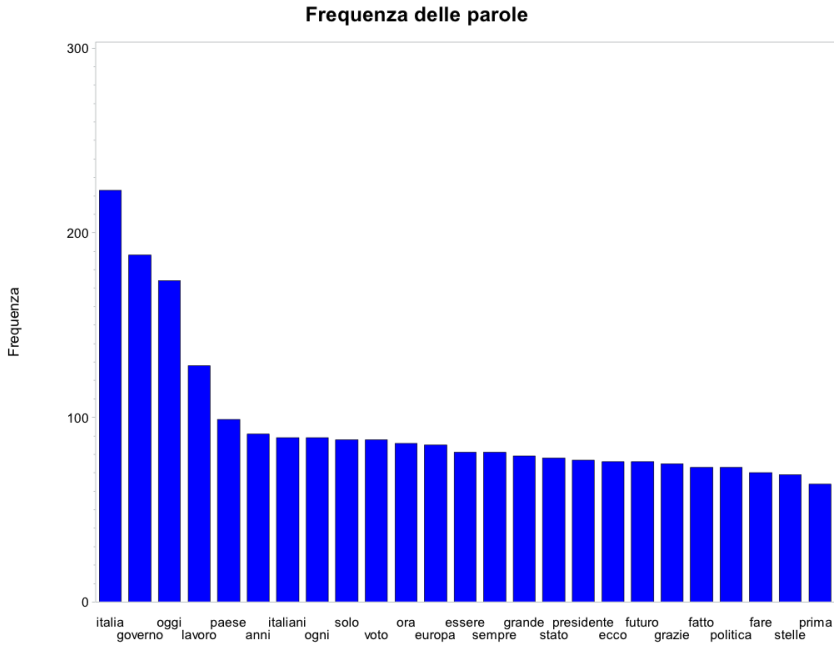


Figure 1.11: *Ordered bar chart: words quoted more than seventy times after seven days of scheduling.*

1.5.2 Correlation of the language of politicians

The goal of this section is to figure out which politicians are more similar in the way of speech. For this purpose, a correlation analysis, through "CORR" procedure, is used [2]. The input data frame contains all the words collected after two days of scheduling. The choice of forty-eight hours is made to guarantee more words (two hundred for each politician) and a more reliable analysis.

Subsequently a graph is drawn with the trend of the words of the table 1.3, with the purpose of graphically verifying the results obtained from the correlations. Berlusconi's line (blue) and Meloni's one (light blue) suggest a greater similarity of language between the two politicians (figure 1.13) and to underline this fact, the lines are reported thicker.

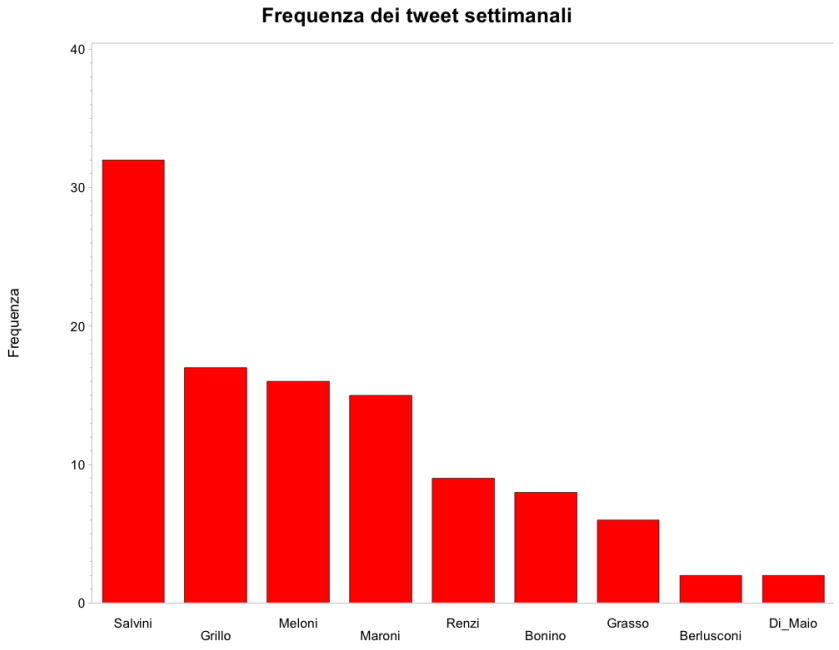


Figure 1.12: *Ordered bar chart: number of tweets published in the last week for each politician.*

Coefficienti di correlazione di Pearson, N = 993 Prob > |r| con H0: Rho=0									
	Berlusconi	**Bonino**	**Di_Maio**	**Grasso**	**Grillo**	**Maroni**	**Meloni**	**Renzi**	**Salvini**
Berlusconi	1.00000	0.21836 <.0001	0.20000 <.0001	0.17353 <.0001	0.16545 <.0001	0.17875 <.0001	0.33295 <.0001	0.27597 <.0001	0.23711 <.0001
Bonino	0.21836 <.0001	1.00000	0.11164 0.0004	0.17420 <.0001	0.14490 <.0001	0.08904 0.0050	0.18265 <.0001	0.22610 <.0001	0.13925 <.0001
Di_Maio	0.20000 <.0001	0.11164 0.0004	1.00000	0.13753 <.0001	0.09647 0.0023	0.15015 <.0001	0.27633 <.0001	0.23671 <.0001	0.17853 <.0001
Grasso	0.17353 <.0001	0.17420 <.0001	0.13753 <.0001	1.00000	0.18848 <.0001	0.09992 0.0016	0.22453 <.0001	0.20677 <.0001	0.13341 <.0001
Grillo	0.16545 <.0001	0.14490 <.0001	0.09647 0.0023	0.18848 <.0001	1.00000	0.13831 <.0001	0.16695 <.0001	0.19739 <.0001	0.15839 <.0001
Maroni	0.17875 <.0001	0.08904 0.0050	0.15015 <.0001	0.09992 0.0016	0.13831 <.0001	1.00000	0.23011 <.0001	0.20523 <.0001	0.16462 <.0001
Meloni	0.33295 <.0001	0.18265 <.0001	0.27633 <.0001	0.22453 <.0001	0.16695 <.0001	0.23011 <.0001	1.00000	0.28219 <.0001	0.33239 <.0001
Renzi	0.27597 <.0001	0.22610 <.0001	0.23671 <.0001	0.20677 <.0001	0.19739 <.0001	0.20523 <.0001	0.28219 <.0001	1.00000	0.26506 <.0001
Salvini	0.23711 <.0001	0.13925 <.0001	0.17853 <.0001	0.13341 <.0001	0.15839 <.0001	0.16462 <.0001	0.33239 <.0001	0.26506 <.0001	1.00000

Table 1.3: *Correlation of words for selected politicians after two days of scheduling.*

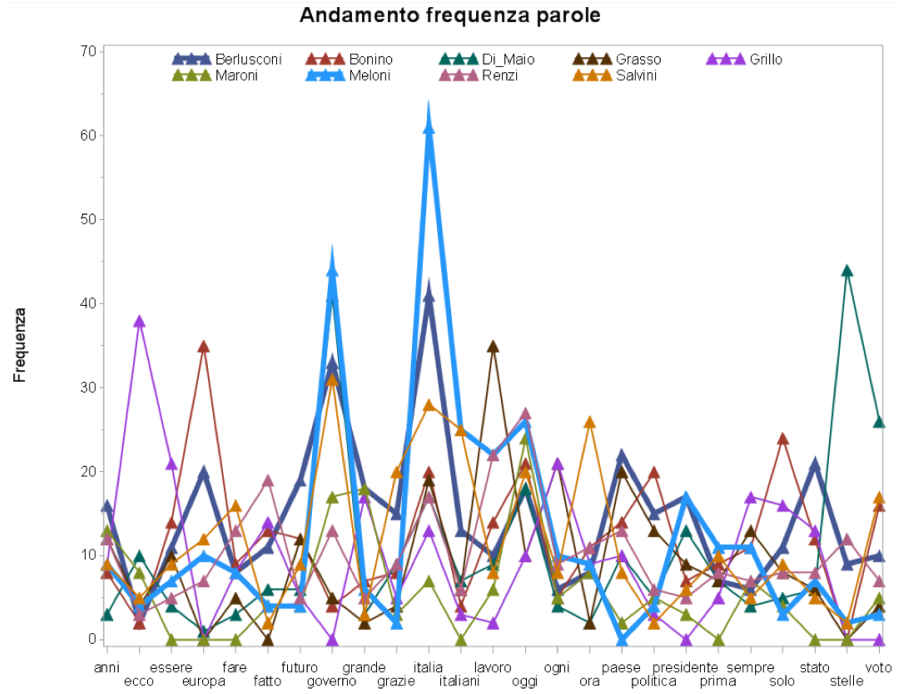


Figure 1.13: *Broken lines show the frequency of the most used words.*

Chapter 2

Classification problems

This chapter introduces some of the most famous algorithms used in machine learning (ML) for classification problems. This overview is very useful to understand the State-Of-The-Art machine learning algorithms. After that, evaluation metrics and k-fold cross-validation are proposed in order to choose the best model for the problem in exam. Finally, a method used to improve the performance of an algorithm, called stochastic gradient descent, is introduced.

2.1 Main concepts of machine learning

Machine learning predicts things based on patterns it has been trained with. In this context, classification is the process where computers group data together based on predetermined variables, also called features. When dealing with a classification problem, it is important to always keep in mind that 100% accuracy cannot be achieved. ML is used to obtain classifications quickly and automatically with as much precision as possible. There are some fundamental concepts, that must be exposed before analyzing the main algorithms [3]:

- Data preparation: the first step of a machine learning approach. It consists of preprocessing data before feature extraction and training.
- Feature extraction: the process used to extract features. They are individual measurable properties for each observation in a data set. In the simplest cases features are the columns of the input data set.
- Training set: a subset of the input one. It is used to train the model.
- Testing set: the remaining part of a data set, used to see how well an algorithm performs.
- Model: a mathematical representation of a real-world process, obtained by combining features and one algorithm of ML. It is the mathematical structure from the input x_i to the prediction y_i.
- Learning: the process of an algorithm learning from the training set in order to create the mapping function from the input to the output. It can be of two types: supervised when the output is known, unsupervised otherwise.

- Hyperparameter tuning: hyperparameters of the model to change to obtain better performance.
- Overfitting: it occurs when the model fits the training data too well and predicts very bad on new data.
- Evaluation of performance: use some metrics or methods to evaluate the models.

2.2 Decision trees

Decision trees are simple, useful for interpretation and, in case of classification problems, they are used to predict a qualitative response. This algorithm inherits the name from its structure. Each node that is not a leaf is associated with a feature test, also known as split. The answer can be positive or negative and, based on it, a splitting rule is applied. The feature space is divided into a number of simple regions $R_1, R_2, ..., R_m$ and the set of decisions are represented in a tree (figure 2.1) [5].

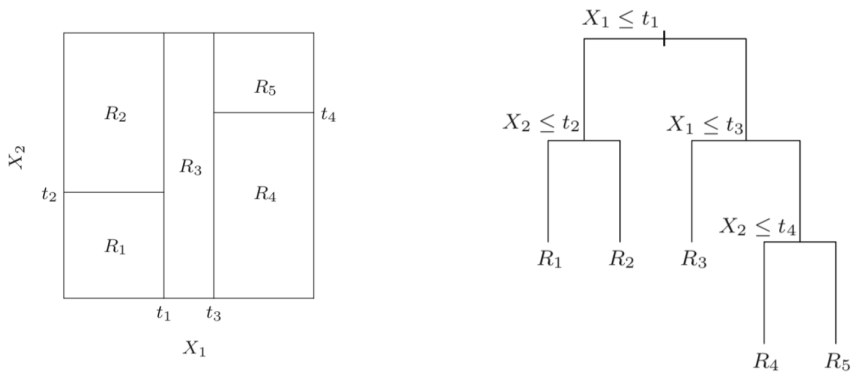


Figure 2.1: *Partition: toy example of a partition by binary splitting, as used in CART algorithm.*

Each subset (region) is considered as the given data set for the next step, generating a recursive process. In a node m, representing a region R_m with N_m observations, the proportion of class k observations is defined as:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k), \tag{2.1}$$

where I refers to the indicator function.

As a criterion for the splits one of these three measures is typically used. Let the misclassification error as

$$E = 1 - \hat{p}_{mk}, \tag{2.2}$$

the Gini-Index

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{2.3}$$

and the cross-entropy or deviance

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}) \tag{2.4}$$

Scikit-learn, the python library used to import the module DecisionTreeClassifiers, implements CART (classification and regression trees) algorithm. By default, it uses the Gini Index to evaluate splits in the data set.

To better understand this algorithm, an example with some numerical values can be useful (figure 2.2). Each internal node shows some information about its split, the Gini index, the distribution of sample and the most probably class. The minimum value of Gini index will always be 0 when all observations belong to one label; this occurs when $\hat{p}_{mk}$ is equal to one. In the orange rectangle of the second level, the class chosen for the leaf is joy. The maximum value of Gini index could be when all target values are equally distributed. In this case it is equal to $1 - \frac{1}{k}$, where k is the number of classes in the classification problem. This is clear in the bottom of the tree in which a sample of 2463 observations is equally divided into three classes.

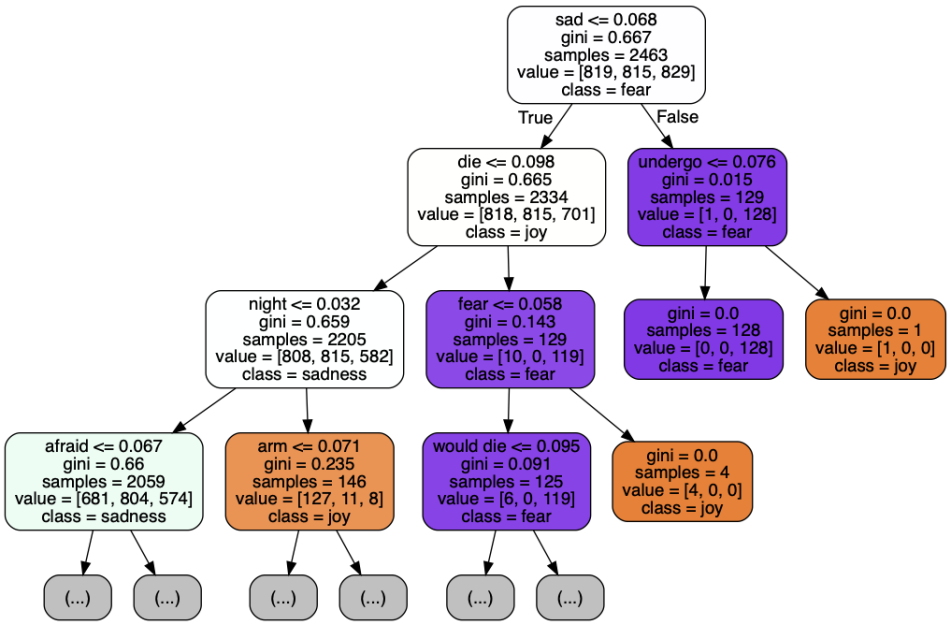


Figure 2.2: *A section of decision tree in sentiment analysis application with three classes. The leaves with orange rectangle correspond to joy class; those with purple color to the fear class; leaves for the sadness class are not present because the tree was cut on the third level.*

2.3 Logistic regression

2.3.1 Logistic function

An explanation of standard logistic function is the starting point to understand the logistic regression algorithm [6]. It is a sigmoid function, which takes any input $t \in \mathbb{R}$ and outputs a value in $[0, 1]$ (figure 2.3).

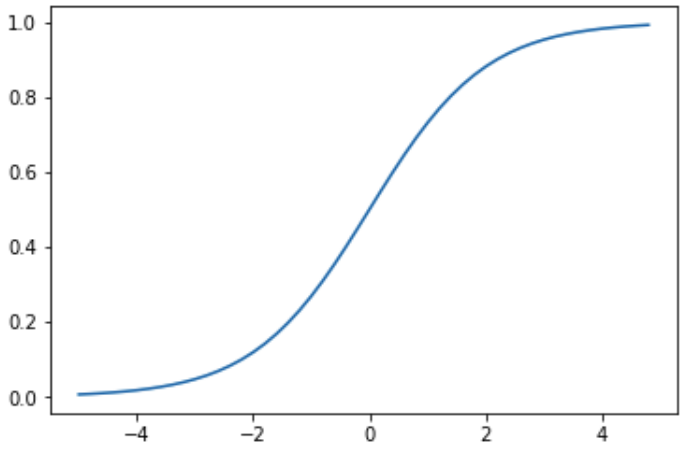


Figure 2.3: *Logistic function*

The logistic function $\sigma(t)$ is defined as follows:

$$\sigma(t) = \frac{\exp(t)}{1 + \exp(t)} = \frac{1}{1 + \exp(-t)} \tag{2.5}$$

Assuming t as linear function in two dimensions, $t = \beta_0 + \beta_1 x$, the logistic function can be rewritten as:

$$p(x) = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x))} \tag{2.6}$$

where $p(x)$ is a probability, and by definition ranges from 0 to 1. In a classification problem, with $y \in \{0, 1\}$, $p(x)$ can be seen as the probability that a new point belongs to class 1:

$$p(x) = P(G = 1 | X = x) \tag{2.7}$$

Therefore, predicting the class is equivalent to finding the value of β_0 and β_1.

The logit (natural logarithm of the odds) is introduced as:

$$g(p(x)) = logit(p(x)) = \ln\left(\frac{p(x)}{1 - p(x)}\right) = \beta_0 + \beta_1 x \tag{2.8}$$

So,

$$\frac{p(x)}{1 - p(x)} = \exp(\beta_0 + \beta_1 x) \tag{2.9}$$

2.3.2 Multinomial logistic regression

Multinomial logistic regression is a type of linear method for classification with more than two classes involved. It models the posterior probabilities of K classes via linear functions in x. The name derives from its underlying technique that is quite the same as the linear regression. At the same time, it ensures that the output value is between $[0, 1]$ and that they sum to one (due to the definition of probability). The model involves $K - 1$ logit transformations:

$$\begin{aligned}
\log \frac{P(G = 1|X = x)}{P(G = K|X = x)} &= \beta_{10} + \beta_1^T x \\
\log \frac{P(G = 2|X = x)}{P(G = K|X = x)} &= \beta_{20} + \beta_2^T x \\
&\vdots \\
\log \frac{P(G = K - 1|X = x)}{P(G = K|X = x)} &= \beta_{(K-1)0} + \beta_{K-1}^T x,
\end{aligned} \tag{2.10}$$

It is useful to apply the exponential function on both terms of the equations and express

$$\begin{aligned}
&P(G = K|X = x) = 1 - \sum_{l=1}^{K-1} P(G = l|X = x) \Longrightarrow \\
&P(G = K|X = x) = 1 - P(G = K|X = x) \left(\sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x) \right) \Longrightarrow \\
&P(G = K|X = x) = \frac{1}{1 + \sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x)}
\end{aligned}$$

The resulting model is

$$\begin{aligned}
P(G = 1|X = x) &= \frac{\exp(\beta_{10} + \beta_1^T x)}{1 + \sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x)} \\
P(G = 2|X = x) &= \frac{\exp(\beta_{20} + \beta_2^T x)}{1 + \sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x)} \\
&\vdots \\
P(G = K|X = x) &= \frac{1}{1 + \sum_{l=1}^{K-1} \exp(\beta_{l0} + \beta_l^T x)},
\end{aligned} \tag{2.11}$$

and of course all probabilities sum to one. As in the binary case, predicting the class is equivalent to finding the value of $\vec{\beta}$.

It should be noted that in natural language processing, logistic regression classifiers are commonly used as an alternative to naive Bayes classifiers (explained in the next section), but they may not be appropriate given a very large number of classes to learn.

2.4 Naive Bayes classifiers

Naive Bayes classifiers are a family of method based on Bayes' theorem. In the general form the Bayes' theorem is:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{2.12}$$

where A and B are events and P(B) is not equal to 0. $P(A|B)$ and $P(B|A)$ are conditional probabilities, while P(A) and P(B) are marginal probabilities.

Considering a partition $A_1, ..., A_n$ of Ω ($A_i \bigcap A_j \neq \emptyset \ \forall i \neq j$ and $\bigcup_{i=1}^{n} A_i = \Omega$) the previous equation can be rewritten:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)} = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{n} P(B|A_j)P(A_J)} \quad \forall i = 1, ..., n \tag{2.13}$$

One of the main algorithms among Naive Bayes classifiers is Multinomial Naive Bayes, that is one of the most popular in the field of text analysis. Comparing to the previous algorithm, learning in such a model is faster than for a logistic regression classifier. It introduces multinomial distribution to compute the probabilities in situations where there are more than two possible outcomes. Although it is a simple method, it is very competitive in data mining after an appropriate preprocessing. In this context the features are the frequency of words (as will be explained in 4.2). It assumes that the value of a particular feature is independent from the value of others. This implies that correlations between features are not considered. Hence, it is called naive.

Considering a set of predictors X as $(x_1, ..., x_d)$ and the class y, the goal is to find the probability of class y given the vector of features X [6]:

$$P(Y = y|X) = \frac{P(x_1|Y = y) \times P(x_2|Y = y) \times ... \times P(x_d|Y = y) \times P(Y = y)}{P(x_1, ..., x_d)} \tag{2.14}$$

Since $P(x_1, ..., x_d)$ is constant, the denominator can be removed, and a proportionality can be introduced.

$$P(Y = y|X) \propto P(x_1|Y = y) \times P(x_2|Y = y) \times ... \times P(x_d|Y = y) \times P(Y = y) \tag{2.15}$$

In a two classes problem, say 0 and 1 as labels, the algorithm predicts class one if $\mathrm{P(Y = 1|X)} > 0.5$, class zero otherwise. The idea of the algorithm is the same in multi classification problems, in which the class with the largest probability is chosen.

Hence,

$$y = \underset{y}{\operatorname{argmax}}\, P(Y = y) \prod_{i=1}^{d} P(x_i|y) \tag{2.16}$$

In Multinomial Naive Bayes, maximum-likelihood estimates are provided for each term in 2.16. The estimate for $P(Y = y)$ for $y \in \{1, \ldots, k\}$ takes the following form:

$$P(Y = y) = \frac{N_y}{n}, \tag{2.17}$$

where n is the number of observations and $N_y = \sum_{i=1}^{n} N_{yi}$ is simply the number of times that the label y is seen in the training set T. $P(x_i|y)$ can be expressed as:

$$P(x_i|y) = \frac{N_{yi}}{N_y}, \tag{2.18}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T. To avoid zero probabilities a smoothing version of it is also introduced:

$$P(x_i|y) = \frac{N_{yi} + \alpha}{N_y + \alpha n}. \tag{2.19}$$

Considering $\vec{\theta}$ as a parameter vector consisting of values for all probabilities in the model, the previous estimates can be derived by maximization of the log-likelihood function [7]. Remember that maximize a function or its logarithm is the same. $L(\vec{\theta})$ is defined as:

$$L(\vec{\theta}) = \sum_{i=1}^{n} \log P(Y = y^{(i)}) + \sum_{i=1}^{n} \sum_{j=1}^{d} \log P(x_j^{(i)}|y^{(i)}) \tag{2.20}$$

Maximum-likelihood estimation finds the parameter values that maximize $L(\theta)$.

2.5 K-nearest neighbors

Given a query point x_0, k-nearest neighbors (KNN) finds the k closest points in distance to x_0 and then classify it using majority vote among the k neighbors. To understand how this algorithm works a toy example with $k = 6$ is presented in figure 2.4. In order to give a label (red or blue) to a new observation, the yellow one, six neighbors are considered. It can be seen in the circle that four points are red, while two are blue. So, the new point is classified by majority vote as red.

To evaluate which ones are the neighbors, the concept of distance is introduced [8]. The Euclidean distance is one of the most used and it can be seen as a particular case of Minkowski distance. The last is defined as

$$d(\vec{x}, \vec{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}, \tag{2.21}$$

where p is a positive number. Manipulating the value of p is possible to obtain three different popular distances:

- $p = 1$: Manhattan distance
- $p = 2$: Euclidean distance
- $p = \infty$: Chebychev distance

Another famous measure is the Mahalanobis one. It evaluates the distance between a point P and a distribution D, measuring how many standard deviations are from P to the mean of D. So, the Mahalanobis distance

of an observation $\vec{x} = (x_1, x_2, \ldots, x_N)^T$ from a set of observations with mean $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_N)^T$ and covariance matrix S is defined in a vector notation as:

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})} \tag{2.22}$$

Another one is the cosine similarity, that is a measure of similarity between two non-zero vectors. It can be derived from the definition of dot product as follows:

$$\vec{x} \cdot \vec{y} = \|\vec{x}\| \|\vec{y}\| \cos(\theta) \Longrightarrow \cos(\theta) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|} \tag{2.23}$$

From the previous formula it is clear that the cosine similarity measures the angle between the two vectors.

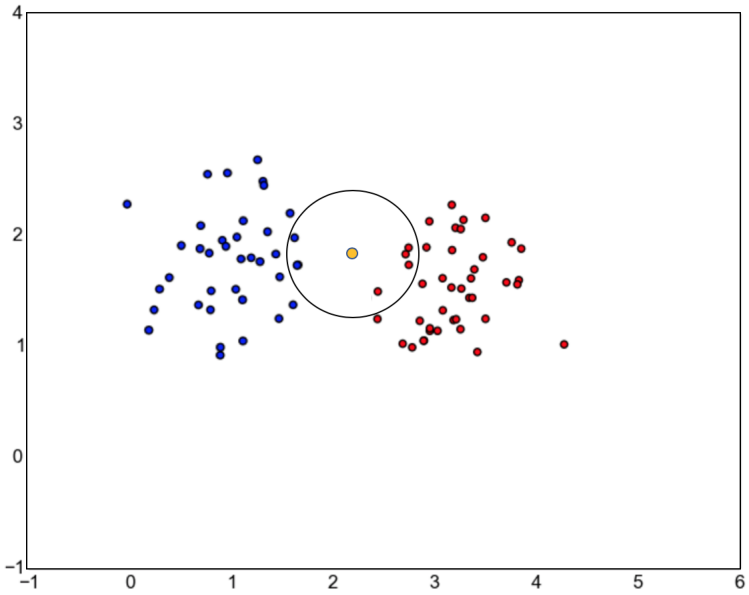


Figure 2.4: *Two-dimensional plot with k-nearest neighbors algorithm. The k parameter is equal to six and the circle shows the six nearest neighbors.*

2.6 Support vector machines

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used successfully in classification challenges. In the first part of the section a two classes classification problem is analyzed, while at the end a more than two classes approach is presented. The support vector machine is a generalization of a classifier called the maximal margin classifier. This simple version of SVM, that requires classes be separable by a linear boundary, helps to understand how this complex algorithm works.

2.6.1 Maximal margin classifier

The maximal margin classifier finds a maximum marginal hyperplane which helps in classifying new data points. An hyperplane is defined as:

$$\beta_0 + \beta_1 X_1 + ... + \beta_p X_p = 0, \tag{2.24}$$

where p is the dimension [6]. A value $X = (X_1, X_2, ..., X_p)^T$ can be seen as a point of the space that lies on the hyperplane if it satisfies the previous formula. If X does not lie on the hyperplane it means that it can be on one of the two sides:

$$\beta_0 + \beta_1 X_1 + ... + \beta_p X_p > 0,$$

or

$$\beta_0 + \beta_1 X_1 + ... + \beta_p X_p < 0.$$

It can be used as a very elementary classifier: a test observation is assigned to a class depending on which side of the hyperplane it is.

In the simplest case in figure 2.5 it should be noted that an infinite number of hyperplanes can be used, only by shifting a tiny bit up or down, or rotating a little, the existing one. In order to have a single choice, the maximal margin hyperplane (also known as the optimal separating hyperplane) is introduced. It has the farthest minimum distance to the training observations.

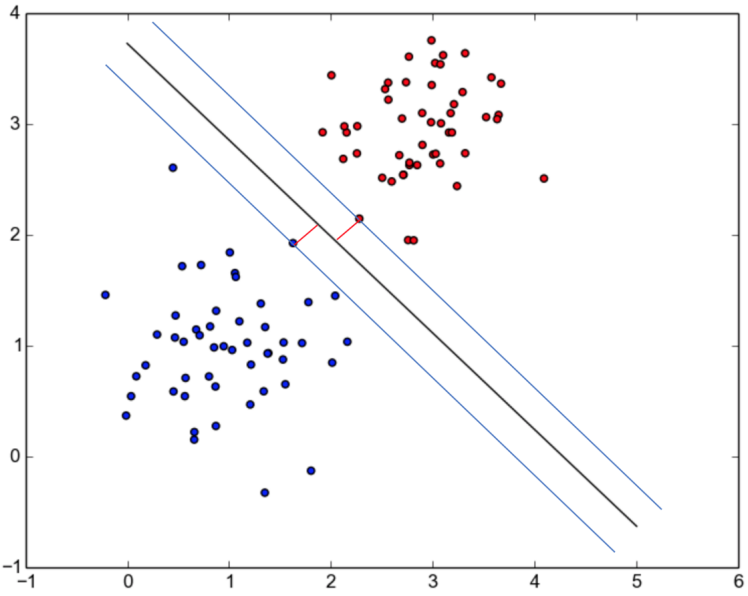


Figure 2.5: *Two classes of observations, shown in blue and in red. The separating hyperplane is displayed as a black line.*

In figure 2.5 two training observations, called support vectors, are equidistant from the maximal margin hyperplane and they lie along the blue lines indicating the width of the margin. The maximal margin hyperplane depends only on the support vectors, because a movement of one of these two points affects the optimal separating hyperplane, while a movement of others not.

Let a set of n training observations $x_1, ..., x_n \in \mathbb{R}^p$ and its labels $y_1, ..., y_n \in \{-1, 1\}$, the maximal margin hyperplane is defined as the solution to the following optimization problem:

$$\begin{aligned} &\max_{\beta_0,\beta_1,...,\beta_p} \quad M \\ &\text{s.t.} \sum_{j=1}^{p} \beta_j^2 = 1 \\ &\qquad y_i(\beta_0 + \beta_1 x_{i1} + \ldots \beta_{ip} x_{ip}) \geq M, \quad \forall i = 1, \ldots, n \end{aligned} \tag{2.25}$$

The optimization problem chooses $\beta_0, \ldots, \beta_p$ to maximize M (the margin of the hyperplane) under the constraints that each observation is at least a distance M from the hyperplane and that it is on the correct side.

2.6.2 Support vector classifier

A relaxed version of the problem 2.25, called support vector classifiers, admits some misclassified observations in order to achieve robustness and better classification:

$$\begin{aligned} &\max_{\beta_0,\beta_1,...,\beta_p} \quad M \\ &\text{s.t.} \sum_{j=1}^{p} \beta_j^2 = 1 \\ &\qquad y_i(\beta_0 + \beta_1 x_{i1} + ...\beta_{ip} x_{ip}) \geq M(1 - \epsilon_i), \quad \forall i = 1, ..., n \\ &\qquad \epsilon_i \geq 0; \quad \sum_{i=1}^{n} \epsilon_i = C, \end{aligned} \tag{2.26}$$

where C is a tuning parameter that sets the number of misclassified observations. ϵ_i is a parameter that can be equal to zero, giving the problem 2.25, if the observation is on the correct side of the hyperplane; greater than 1 if it is on the wrong side; $\in (0, 1)$ if it is within the margin. The support vector classifiers can be represented as

$$f(x) = \beta_0 + \sum_{j=1}^{p} \alpha_i (x_{ij} x_{i'j}), \tag{2.27}$$

with n parameters α_i.

2.6.3 SVMs

In most cases data are not linearly separable and a different approach is necessary. In order to handle this non-linearity some functions, such polynomials or splines, are applied. This involve a change in the number of features. The idea of SVMs is to find an automatic way to achieve efficient computations enlarging the feature space. This is done with kernels: functions used to quantify the similarity of two observations. In the simplest case it can be seen as an inner product between two observations:

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}, \tag{2.28}$$

where p is the dimension of the predictor space. To obtain a more flexible decision boundary a polynomial kernel of degree d is introduced:

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d. \tag{2.29}$$

There are different types of kernels, one of the main used is the radial one (figure 2.6). It is defined as:

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} x_{i'j})^2), \tag{2.30}$$

with γ as a positive constant that ranges from 0 to 1. $\gamma = 0.1$ is usually a good default value.

Another popular choice in literature is the neural network kernel, defined as:

$$K(x_i, x_{i'}) = \tanh(k_1 \sum_{j=1}^{p} x_{ij} x_{i'j} + k_2), \tag{2.31}$$

with k_1 and k_2 as constants. So, a support vector machine can be seen as a support vector classifier with a non-linear kernel. It can be defined as

$$f(x) = \beta_0 + \sum_{j=1}^{p} \alpha_i K(x_{ij} x_{i'j}), \tag{2.32}$$

with n parameters α_i, one for each observation.

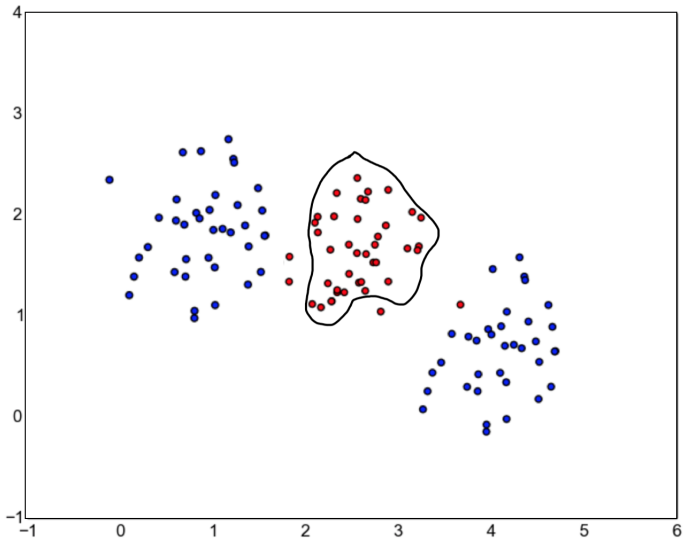


Figure 2.6: *Two classes of observations, shown in blue and in red. The radial kernel is used as decision boundary.*

SVMs can also be extended to multi classification problems [9]. The most commonly used methods are one-versus-one and one-versus-all. In the first

method $\frac{K(K-1)}{2}$ SVMs are constructed, each of which compares a pair of classes. An observation is classified using all the SVMs. The resulting class is obtained by majority votes of all classifiers.

The second approach constructs K separate binary classifiers for K-class classification, comparing one of all the K classes to $K-1$ remaining ones. A test observation is applied to the class that gives the maximum output value.

2.7 Performance evaluation

To evaluate which model is the best, applied to a particular problem, performance evaluation is needed [6]. First of all, some evaluation metrics and the receiver operating characteristic (ROC) curve are defined. After that, cross-validation is introduced in order to make the classification more robust, repeating the experiment multiple times, using all the different parts of the training set as validation sets.

2.7.1 Metrics

	PREDICTIONS	
ISTANCES	NEGATIVE	POSITIVE
NEGATIVE	TN	FP
POSITIVE	FN	TP

Figure 2.7: *Confusion matrix in binary case*

In order to evaluate the accuracy for each class, confusion matrix is introduced (figure 2.7 for binary case). The accuracy of a classifier is defined as follow:

$$Accuracy = \frac{TN + TP}{TN + FP + FN + TP}, \tag{2.33}$$

where the acronyms indicate true negative, true positive, false positive and false negative. In multiclassification problems true positive are on the diagonal (figure 2.8). The value of accuracy can be seen on the diagonal of normalized confusion matrix, as in figure 2.9 where the values for joy, sadness and fear are respectively 83.3%, 76.7% and 77.3%. These percentages are obtained by divided the value in each cell for the sum of the row (for instance 0.833 is equal to $\frac{225}{225+30+15}$). In some cases, this metric may not be accurate: for example if one set has a small percentage of "positive" (say 1%), a classifier that always says "negative" obtains an accuracy of 99%. Alternative measures are:

$$Precision = \frac{TP}{FP + TP}, \tag{2.34}$$

that, given all the predicted labels for a given class, indicates how many instances are correctly predicted;

$$Recall = \frac{TP}{FN + TP}, \tag{2.35}$$

that shows the percentage of positives classified as positives;

$$F_1\ score = \left(\frac{recall^{-1} + precision^{-1}}{2}\right)^{-1} = 2 \cdot \left(\frac{precision \cdot recall}{precision + recall}\right), \quad (2.36)$$

that is the harmonic average of the precision and recall and reaches its best value at 1, in which precision and recall are perfect;

$$Specificity = \frac{TN}{FP + TN}, \quad (2.37)$$

that is equal to 1 - FPR (false positive rate), which is the proportion of false positives on the total of observations.

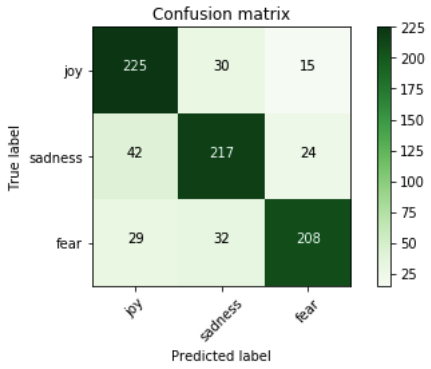


Figure 2.8: *Confusion matrix with number of instances*

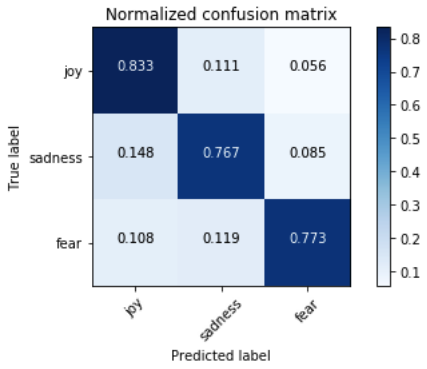


Figure 2.9: *Confusion matrix with accuracy on the diagonal*

The ROC curve is created by plotting the true positive rate (TPR) against the FPR [12]. The first is also called precision or sensitivity, while the second is equal to 1 - specificity. This is strongly related to the concept of area under the curve (AUC). The top left corner of the plot is the "ideal" point, where there is a true positive rate of one and a false positive rate of zero. It means that a larger area under the curve is usually better.

An extension in multi-class problems is proposed in figure 2.11, in which every line represent the ROC curve for each class [11]. The dimension of space needed to describe a problem with c classes is $c(c-1)$. It is clear that a binary classification ROC curve can be drawn in a bi-dimensional plot, while in three-classification problem the dimension requested is 6.

2.7.2 K-fold cross-validation

Cross-validation uses the initial training data to generate multiple train-test splits [6]. In case of k-fold cross-validation, data is partitioned into k subsets, called folds. Then, the algorithm is iteratively trained on $k-1$ folds using the remaining one (called held-out fold) as the test set. A function to evaluate test error is needed: it can be the mean squared error, MSE, in case of a regression problem or the number of misclassified observations in case of a classification one (figure 2.10).

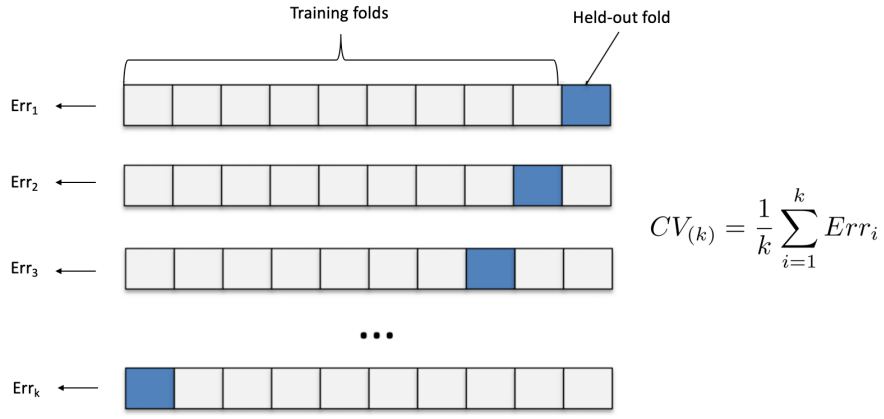


Figure 2.10: *k-fold cross-validation for classification*

The test error is then computed on the observations in the held-out fold and this procedure is repeated k times. Then, k-fold CV estimate is computed by averaging these values. For regression problems

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i, \tag{2.38}$$

while in classification ones

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} Err_i, \tag{2.39}$$

where Err_i indicates the number of misclassified observations in the i-th held-out.

Since the calculation is repeated k times, cross-validation can provide a more truthful result of the performance measure used for a given method. It is very useful in data science and it is a preventative measure against overfitting. This

phenomenon is "the production of an analysis that corresponds too closely or exactly to a particular set of data and may therefore fail to fit additional data or predict future observations reliably". An overfitted model follows the training data, it is too dependent from them and it is likely to have a bad performance on new unseen data. In contrast an underfitted model cannot capture the underlying trend of the data and it cannot predict on new data well enough.

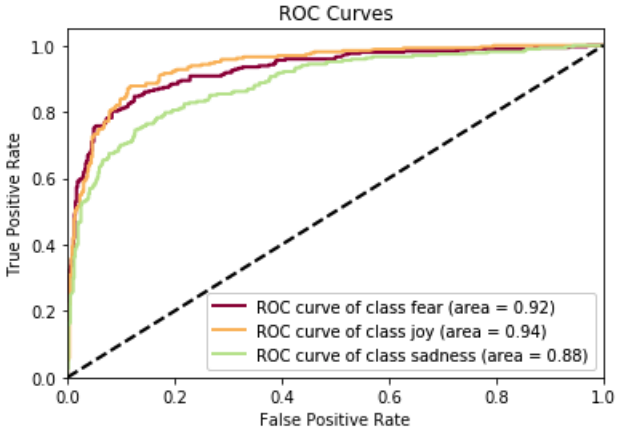


Figure 2.11: *ROC curves of three-classification problem*

2.8 Stochastic gradient descent

Stochastic gradient descent (SGD) is an iterative method for optimizing a differentiable objective function. It is a stochastic approximation of the gradient descent, where the term stochastic refers to the fact that samples are random selected. It is a popular algorithm for training models in machine learning, like support vector machines and logistic regression. The python module that implements this method is SGDClassifier. In order to use one of the algorithms above the loss parameter is set to "hinge" or "log".

SGD is one of three variants of gradient descent [10]. Batch gradient descent and mini-batch gradient descent are the others and they differ in how much data are used to compute the gradient of the objective function. Considering an objective function that has the form of a sum:

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} J_i(w), \tag{2.40}$$

with the parameter w to be estimated. Each J_i is associated to each observation in the training set. In order to minimize the above function, the batch gradient descent method performs the following iterations:

$$w := w - \eta \cdot \nabla_w J(w), \tag{2.41}$$

where η is the learning rate.

Stochastic gradient descent (SGD) in contrast performs a parameter iteration, also called update, for each training observation x_i. It performs one update at a time, and it is usually much faster than batch method. At each iteration

$$w := w - \eta \cdot \nabla_w J(w; x^{(i)}), \tag{2.42}$$

where $J(w; x^{(i)})$ indicates $J_i(w)$. This formula is applied until the algorithm converges. Briefly, it happens when η decrease with an appropriate rate [1].

Mini-batch gradient descent performs an update for every mini-batch of n training observations, instead of using one. This usually allows the algorithm to get better performance in less time. The equation becomes

$$w := w - \eta \cdot \nabla_w J(w; x^{(i:i+n)}), \tag{2.43}$$

where $x^{(i:i+n)}$ indicates that n observations are used.

[1] More details are not reported, because out of the scope of the thesis.

Chapter 3

Ensemble methods

Also known as learning multiple classifier systems, they are a powerful machine learning technique, developed in the last years and widely used in classification problems [13]. The idea of combining multiple classifiers starts from the principle that there is no perfect machine learning algorithm. Every model has its limitations and weaknesses, so the goal is to consider the best possible decision taken overall in order to overcome these boundaries. Combining multiple algorithm is at least more accurate than random guessing, because random errors cancel each other out and correct decision are reinforced. Some competitions, like Kaggle, have in multiple classifiers the most winning ones. This chapter provides an introduction to ensemble methods, with particular focus on some well-known algorithms in the context of classification problems: voting methods, bagging, boosting and stacking.

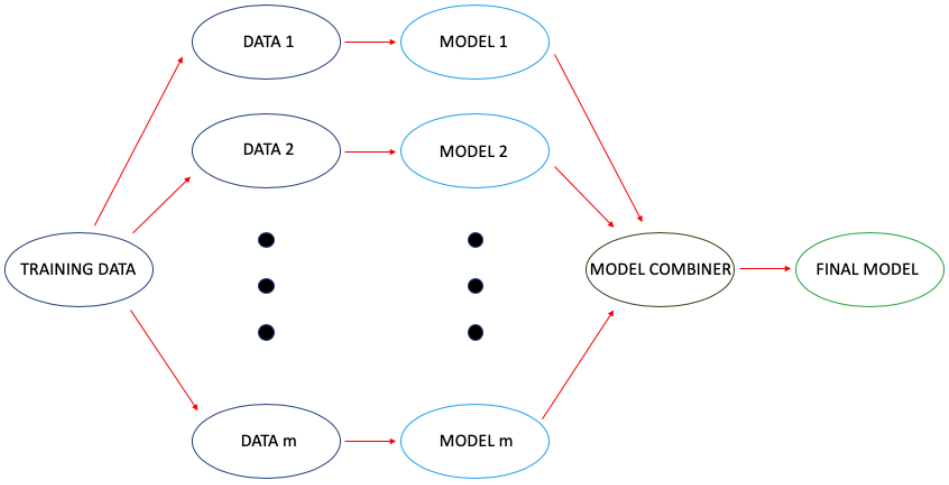


Figure 3.1: *Ensemble models architecture*

Ensemble methods can be divided into two types:

- Homogeneous ensembles: they are a combination of n base learners, also

called weak learners, generated by base learning algorithms of the same type, like decision tree, SVM etc... One of the most representative methods is Random Forest, that, as the name suggests, is composed by a huge number of decision trees.

- Heterogeneous ensembles: different learners are used, so the base learners are indicated as individual ones, in order to make a distinction from the previous method.

A common architecture of homogeneous ensembles is shown in figure 3.1, in which m models are trained on a portion of the training data and then combined together giving the final model. The model combiner can be obtained in different ways and it will be analyzed later.

3.1 Voting methods

Voting methods build several estimators and then average their predictions. The ensemble one has usually a best performance than single base estimators. Consider the following scenario with a set of T individual classifiers $h_1, \ldots, h_T$ and K classes, each one labelled as c_k with $k \in [1, \ldots, K]$ [4]. For a classifier h_t $t \in [1, \ldots, T]$ the outputs associated to an instance x are a K-dimensional vector $(h_t^1, \ldots, h_t^K)$. h_t^k has two formulations:

- Class label: $h_t^k \in \{0, 1\}$ that takes value 1 if the class is k and 0 otherwise.
- Class probability: $h_t^k \in [0, 1]$. It can be seen as an estimate of the posterior probability $P(c_k|x)$ for classifier h_t.

There are several ways to combine prediction. First of all, the mode can be used to obtain the value that occurs most often for each classification. This type is called plurality voting:

$$H(x) = c_{\underset{j}{\text{argmax}} \sum_{i=1}^{T} h_i^j(x)}, \tag{3.1}$$

where $H(x)$ is the output of the ensemble classifier.

Another one is majority voting, where at least 50% of classifiers choose the same class. In case of an observation does not reach this percentage, is cannot be classified. This is a reject option and is clear that in plurality voting it does not exist.

$$H(x) = \begin{cases} c_j & if \quad \sum_{i=1}^{T} h_i^j(x) > \frac{1}{2}\#T \\ rejection & otherwise \end{cases} \tag{3.2}$$

where $\#T$ indicates the number of models used.

For example, consider a three-classification problem, with class $y \in \{0, 1, 2\}$. Using four classifiers, the prediction classes for the first observation are 0, 2, 0 and 1. It is clear that plurality chooses class 0 as best prediction, while majority voting gives a reject option because class 0 reaches 50%, without exceeding it.

Another possibility is to simply use the mean of predictions. It can be done via class label or probability representation. As was said before, in the first case when a model classified an instance in a class j the resulting vector have

all entries equal to zero and one in position j. For example, it can be (0, 0, 1, 0) for third class in a four-classification problem. Suppose to have other two classifiers that give (0, 1, 0, 0) and (0, 0, 1, 0) as class label vectors. Evaluate the mean implies to sum the elements in the same position and divide by the number of classifiers, giving (0, 1/3, 2/3, 0). For each class it is defined as:

$$H^j(x) = \frac{1}{T}\sum_{i=1}^{T} h_i^j(x) \tag{3.3}$$

Now, choosing the higher value in the vector it is clear that the predicted class by the ensemble classifier is the third.

An example with the prediction probabilities applied to a three-classification problem is reported. For an observation the vectors of class probabilities are the following:

- first classifier: [0.37, 0.34, 0.29]
- second classifier: [0.4, 0, 0.6]
- third classifier: [0.57, 0.08, 0.35]

Applying formula 3.3, the classifier evaluates the mean in each class and the result is [0.4467 0.14 0.4133]. Now, the predicted class is chosen as the higher value in the vector. So, the observation is associated to the first class.

The last case of voting methods evaluates a weighted mean, giving more importance to the most accurate classifiers:

$$H^j(x) = \sum_{i=1}^{T} w_i h_i^j(x), \tag{3.4}$$

where the sum of weights is one.

In both cases of mean and weighted mean the output $H(x)$ of the ensemble classifier is obtained as the maximum value in the vector composed by $H^j(x)$, it is defined as:

$$H(x) = c_{\underset{j}{\operatorname{argmax}}\, H^j(x)} \tag{3.5}$$

With an accurate choice of weights, the last ensemble can be the best among voting methods.

3.1.1 Choice of weights

Let $\vec{l} = (l_1, \ldots, l_T)^T$ the vector composed by the outputs of the individual classifiers, where l_i is the class label predicted for the instance x by classifier h_i and p_i its accuracy [4].

Maximum-likelihood can be used to find the estimates for the weights of equation 3.4. Since every output of classifier h_t^k can be seen as an estimate of the posterior probability $P(c_k|x)$, it is possible to refer to the observations made for Multinomial Naive Bayes in section 2.4:

$$P(Y = c_k|l) = \log\left(P(Y = c_k)\prod_{i=1}^{T} P(l_i|c_k)\right) \tag{3.6}$$

This posterior probability can be rewritten in order to obtain an approximation for the weights:

$$\begin{aligned} P(Y = c_k|l) &= \log P(Y = c_k) + \sum_{i=1}^{T} \log P(l_i|c_k) \\ &= \log P(Y = c_k) + \log \left(\prod_{i=1, l_i = c_k}^{T} P(l_i|c_k) \prod_{i=1, l_i \neq c_k}^{T} P(l_i|c_k) \right) \\ &= \log P(Y = c_k) + \log \left(\prod_{i=1, l_i = c_k}^{T} p_i \prod_{i=1, l_i \neq c_k}^{T} (1 - p_i) \right) \\ &= \log P(Y = c_k) + \sum_{i=1, l_i = c_k}^{T} log \left(\frac{p_i}{1 - p_i} \right) + \sum_{i=1}^{T} log(1 - p_i) \\ &= \log P(Y = c_k) + \sum_{i=1}^{T} h_i^k(x) \log \left(\frac{p_i}{1 - p_i} \right) \end{aligned}$$

The last equation comes from the fact that $h_i^k \in \{0, 1\}$, takes value 1 if the class is k and 0 otherwise. It can be seen that the first term of the right-hand side does not depend on the classifiers, while the second according to 3.4 shows that

$$w_i \propto \log \left(\frac{p_i}{1 - p_i} \right) \tag{3.7}$$

It is a very interesting result because it suggests that the optimal weights should depend on the accuracy of the classifier.

3.1.2 Calibration of probabilities

For heterogeneous ensembles method calibration of probabilities is required [14]. It is a rescaling operation that is applied after a model made its predictions. Well-calibrated probabilities reflect the true likelihood of the class predicted. To understand this fundamental concept, think about a classification problem that have to say an e-mail is a spam or not.

A probability value of 50% would be well-calibrated if, for example, in one hundred past prediction fifty would be spam. The first step is divide data into groups based on their class probabilities. For example, it can be done using 10 sets, each one also called bin, with range of 10% ([0,10%], ..., (90. 100%]). Suppose that 30 samples are classified with probability less than 10% and there is a single event. The midpoint of the first set is 5% and the observed event rate would be 3.33%. The calibration plot displays the midpoint of the bin on the x-axis and the observed event rate on the y-axis. Ideally, the model produced well-calibrated probabilities if the points falls on the line $y = x$ (figure 3.2).

In literature there are two popular approaches: Platt Scaling and Isotonic Regression [15]. In particular, there are some algorithms that do not produce predictions of probabilities, so these must be approximated. Some examples are support vector machines and decision trees. The calibration can bring many advantages to these models. In the first method calibrated probabilities can be

obtained applying logistic regression to the output of the algorithm:

$$\hat{p} = \frac{1}{1 + \exp -(\beta_0 + \beta_1 f)}, \tag{3.8}$$

where f is the output, β_0 and β_1 are parameters fitted by maximum likelihood estimation on the training set. It is usually done with k-fold cross-validation.

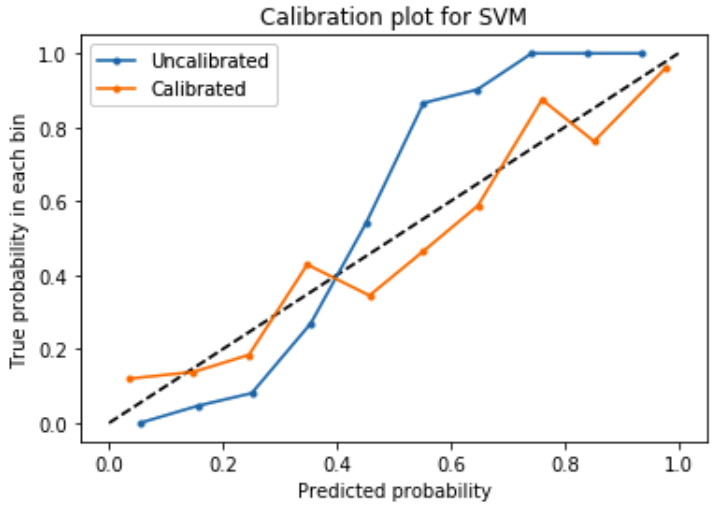
Isotonic regression is more general than Platt method with sigmoid function. The only restriction is that the mapping function has to be isotonic (monotonically increasing). Given the prediction f_i and the true value y_i the assumption is

$$y_i = m(f_i) + \epsilon_i \tag{3.9}$$

It solves the following problem:

$$\begin{aligned} &\min \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2 \\ &\text{s.t. } \hat{y}_{min} = \hat{y_1} \leq \hat{y_2} \leq \cdots \leq \hat{y_n} = \hat{y}_{max}, \end{aligned} \tag{3.10}$$

where each w_i is strictly positive and each y_i is a real number.




```
Uncalibrated model F1 score: 0.835

Calibrated model F1 score: 0.839
```

Figure 3.2: *Calibration plot for SVM model in two-classification problem*

3.2 Bagging

Bagging (stands for Bootstrap AGGregatING) is particularly useful and frequently used in the context of decision trees, because of their high variance [16].

Indeed, it is a procedure used in order to decrease the variance of a machine learning algorithm. For example, the predictions for an average of trees have lower variance than the variance of the individual ones.

The concept of bootstrap is introduced. It is a type of resampling with replacement, in which smaller random samples of the same size (bootstrap samples) are obtained by the original data set. In bagging the classifiers are built on bootstrap samples of the training set. Then, their outputs are combined by a plurality vote.

Its general idea is shown in figure 3.3. From a data set of N observations, M bootstrap samples are constructed, each one with N observations. Since, it happens with replacement each observation has a probability $p = \frac{1}{N}$ of being selected and $p^* = 1 - \frac{1}{N}$ of not being selected. So, the probability of not being selected n times is

$$\left(1 - \frac{1}{n}\right)^n \approx e^{-1} = 0.368 \tag{3.11}$$

This procedure is also called 0.632 bootstrap, because the training data contains approximately 63.2% of the instances. A toy example of this approach is shown in figure 3.4.

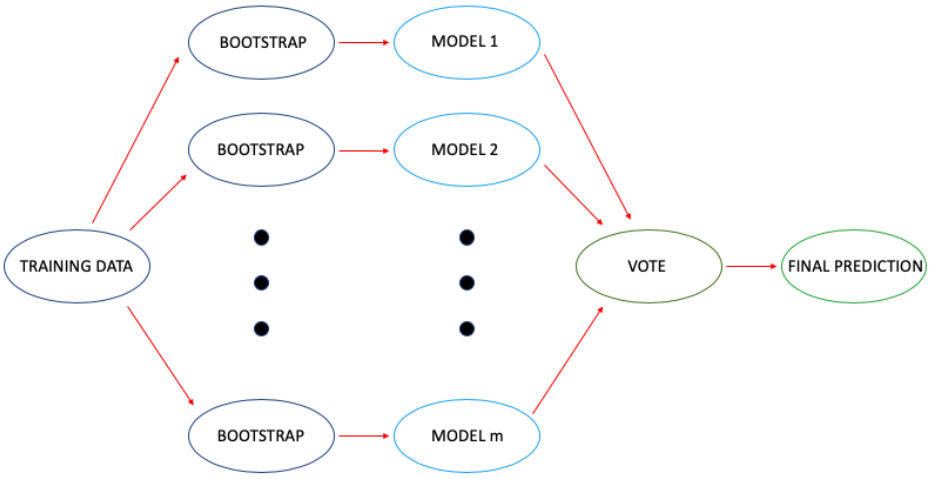


Figure 3.3: *Bagging procedure*

3.2.1 Random forest

Random forest is used in the context of decision trees and its name derives from the idea of using many of them to increase the quality of the predictions. It is very similar to bagging, but in training each model just a random sample of m predictors, among the full number of p, is chosen as split candidates [17]. This difference causes the trees in the forest to be de-correlated. For classification problems, a usual choice is to set $m = \sqrt{p}$.

```
Original dataset: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


1-Bootstrap dataset: [4, 2, 1, 2, 5, 3, 10, 7, 4, 7]
2-Bootstrap dataset: [10, 9, 5, 1, 2, 3, 4, 6, 8, 6]
3-Bootstrap dataset: [1, 2, 1, 2, 6, 10, 1, 1, 4, 10]
4-Bootstrap dataset: [8, 7, 6, 6, 4, 5, 7, 6, 4, 2]
```

Figure 3.4: *Bootstrap approach: toy example of four bootstrap samples obtained from a data set of ten observations.*

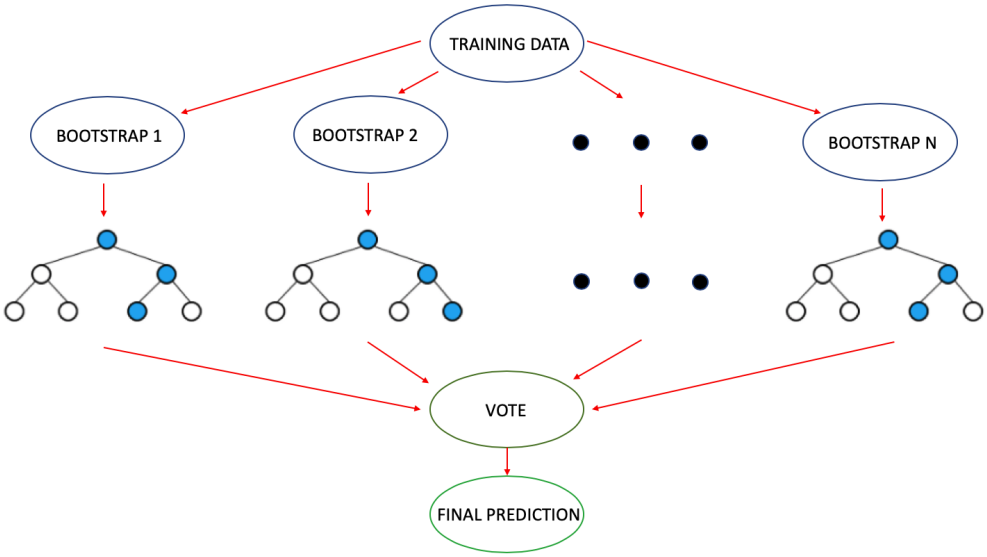


Figure 3.5: *Random forest procedure*

3.3 Boosting

This is a general approach, which refers to a family of algorithms able to convert weak learners to stronger ones. In this procedure the models are grown sequentially: each one is grown using information from the previous (figure 3.6). The most popular boosting algorithm due to Freund and Schapire is "AdaBoost.M1." In order to understand how this algorithm works an example is proposed. It is a two-class classification problem with N observations and the output class $Y \in \{-1, 1\}$. A weak classifier $M(X)$, one whose error rate is slightly better than random guessing, takes a vector of predictor variables X and produces a prediction in one of the two classes. Random guessing refers to a method in which the events involved have an equal chance of being chosen.

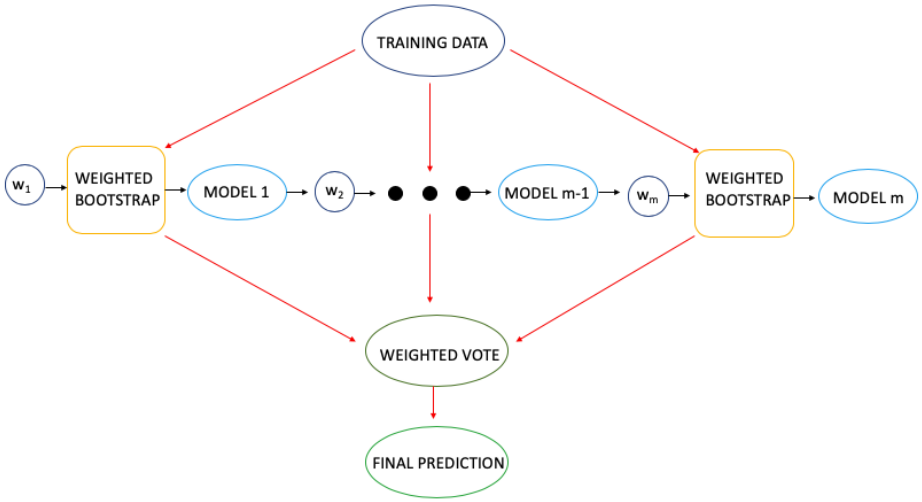


Figure 3.6: *Boosting procedure*

Given one weak classification algorithm, it is sequentially applied to repeatedly modified versions of the training data, producing a sequence of M weak classifiers. At the first step, classifier M_1 is trained in the usual way. Some weights are initialized giving the same importance to every observation ($w_i = \frac{1}{N}$). From the second one, until the end, the training data set is modified applying some weights $w_1 \dots w_n$ to each observation. Weak classifiers are trained to the data set and the one with the lowest classification error e_m is selected. Then, the weight α_m for the M_m is introduced as

$$\alpha_m = \frac{1}{2} \log \left(\frac{1 - e_m}{e_m} \right) \tag{3.12}$$

For a classifier with less than 50% accuracy, where e_m is greater than 0.5, the weight α_m is negative and vice versa. So, it is possible to combine the prediction flipping the sign. It should be noted that, sometimes in literature, the term $\frac{1}{2}$ is omitted in formula 3.12. However, it does not change the logic of

this term. Then, for each data point its weight is updated:

$$w_{m+1}(x_i, y_i) = \frac{w_m(x_i, y_i)\exp(-\alpha_m y_i M_m(x_i))}{Z_m}, \tag{3.13}$$

where Z_m is a normalization constant that guarantees the sum of all instance weights is equal to 1.

For example, imagine that a positive weighted classifier misclassifies the observation in exam, the "exp" term in the numerator would be always larger than 1, giving more weight to this misclassified observation in the next iteration. This happens because $y_i M_m(x_i)$ is -1, α_m is positive and so also $-\alpha_m y_i M_m(x_i)$ is positive.

Otherwise, weights associated to correct predictions are decreased due to the negative term $-\alpha_m y_i M_m(x_i)$ that, applying the exponential function, gives a number $\in (0, 1)$. Summing up, the relative influence of misclassified observations is increased, inducing the next classifier in the sequence to predict better.

Then the prediction of each algorithm is combined with the others through a weighted majority vote. The final prediction obtained is

$$F(X) = sign\left(\sum_{m=1}^{M} \alpha_m M_m(X)\right), \tag{3.14}$$

where α values give more weight to the more accurate classifiers.

3.3.1 Adaboost for multi-class classification

The two-class adaboost algorithm can also be extended in multi classification problem. The logic is the same of the binary case, but with one critical adjustment to the algorithm. According to [19] the proposed algorithm is equivalent to a forward stagewise additive modeling one, that minimizes an exponential loss for multi classification. The new algorithm, called SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function), is almost identical to AdaBoost but with a simple update to the α_m of formula 3.12. The term $\log(K-1)$, where K is the number of class, is introduced obtaining

$$\alpha_m = \log\left(\frac{1-e_m}{e_m}\right) + \log(K-1) \tag{3.15}$$

It is clear that for two-class problem this term is equal to 0 and does not affect α_m. As natural consequence, α_m in order to be positive only requires $1 - e_m > \frac{1}{K}$, in contrast to 0.5 in two-class adaboost.

One of the most famous algorithms for boosting in Python is called AdaBoostClassifier.

3.4 Stacking

Another type of ensemble method that can achieve excellent results in classification is stacking. It is a general procedure composed by different learners (algorithms) and levels [20]. A learner, called meta-learner, is trained to combine the individual ones of the previous level. This procedure can be applied over and over again. In figure 3.7 a general procedure of two levels is shown.

The name comes from the fact that the final model is said to be stacked on the top of the others. It should be noted that in any ensemble method there is not guarantee that the model created will have better performance than all the algorithms used.

Input:

Data set D = $\{(x_1, y_1), (x_2, y_2), \dots, (x_M, y_M)\}$

First-level learning algorithms: $A_1, \dots, A_T$

Second-level learning algorithm: A*

Process:

```
1. for t = 1 : T:
2.     h_t = A_t(D);                          # each algorithm trained on D
3. end;
4. D' = ∅;
5. for i =1 : M:
6.     for j = 1 : T:
7.         z_{i,t} = h_t(x_i)                 # output of each algorithm for i-th observation
8.     end;
9.     D' = D' ∪ { (z_{i,1}, … , z_{i,T} ), y_i };   # update of new data set
10. end;
11. h' = A*(D');                              # second level learner trained on new data set
```

Output:

$H(x) = h'(h_1(x), \dots, h_T(x))$

Figure 3.7: *Stacking procedure on two levels*

Chapter 4

Sentiment analysis

Emotion analysis is one of the main techniques in text mining. Also roughly knows as text analytics, it is the process of deriving high-quality information from text [21]. This involves using Natural language processing (NLP). It is a subfield of computer science that refers to the interactions between computers and natural languages, developed and evolved by human through communication and natural use. Machine translation is one of the most famous applications for NLP, with Google Translator as the main example.

The applications of sentiment analysis are many: for example, surveys on public opinion, customer satisfaction in relation to a specific product or service and research relating to political orientations before the elections.

Emotion analysis refers to the process of identifying human emotions from a sentence or a period. To make this analysis machine learning algorithms need some data as input. In this study the training set is composed by 7666 tweets associate to 7 different emotions. Evaluations of performance are done for only three emotions, because the small size of the data set affects a lot the program accuracy. A classification made for more than five emotions cannot reach the 70% of accuracy, while with tree emotions is almost 80%.

4.1 Data preparation

Data preparation is the act of preprocessing data. It refers to all the steps that should be followed to clean and standardize textual data into a form that could be used as input by natural language processing or applications. In order to use the machine learning algorithms for sentiment analysis some transformations are needed.

First of all, word tokenization is applied. It is the process of splitting or segmenting sentences into their words. After that, procedures of text normalization, like stemming, the elimination of stop words and cleaning text (removing punctuations, emoticons, html etc...) are used. Stemming, as introduced in 1.3, is a procedure in which one term is reduced to its stem (figure 4.1). Lemmatization is very similar to this technique, but every word is reduced to its root, also known as lemma, that is different from the stem. The last one may not be a lexicographically correct word, while the first one is always present in the dictionary.

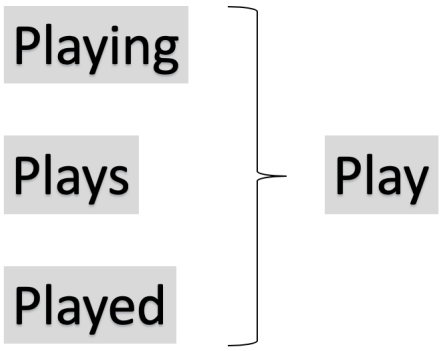


Figure 4.1: *Example of stemming procedure*

Contractions are another example of text normalization. They are shortened version of words or expressions. They exist in either written or spoken forms and they are used quite extensively in the informal language of social media. In case of English language, they are often created by removing one of the vowels from the word (for instance *isn't* sometimes replaces *is not*). A contraction map available at https://github.com/search?q=text+analytics is used (figure 4.2). This link refers to a search query on github.com, one of the main web platforms used by software developments.

To remove punctuations, emoji and other special character regular expressions can be very useful. "Regular expressions use the backslash character ('\') to indicate special forms or to allow special characters to be used without invoking their special meaning. A regular expression (or RE) specifies a set of strings that matches it. " [22]. It has a lot of functions like "sub" (substitute words or characters with others), or "findall" that find all the words of the desired type in a string. For example r' \# \w+', used as first argument of re.findall, allows to obtain all the words that begins with # (the second argument needed from the function is a string in which to search).

```
CONTRACTION_MAP = {
"ain't": "is not",
"aren't": "are not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
```

Figure 4.2: *Few lines of the dictionary CONTRACTION_MAP, defined to handle contractions of English words.*

4.2 Feature extraction

Before training a model a procedure of feature extraction is needed [21]. As was introduced in 2.1 features can be the attribute or a data set. In text mining something more complicated is required, in order to transform sentences in number vectors. So, feature vectors are a row representation of words into the corpus (a sentence or a period), expressed by numbers. In the program two different types are used:

- Bag of Words Model
- TF-IDF Model

The first one involves a frequency representation, in which each column represents a word in the document and its value is expressed by a number that is the frequency of times that the word occurs. For instance, in a toy example using two sentences the word "father" occurs two times (figure 4.3).

The second one is TF-IDF, that stands for term frequency inverse document-frequency. As the name suggests, it involves two terms multiplied by each other:

$$tf\text{-}idf(d,t) = tf(d,t) \times idf(t) \tag{4.1}$$

The first, that represents the bag of words model, is the number of the time that a word occurs in a corpus, while the second is expressed as

$$idf(t) = \log\left(\frac{1+n_d}{1+df(d,t)}\right) + 1, \tag{4.2}$$

where n_d is the total number of documents or sentences and $df(d,t)$ is the number of documents that contain term t. The second one is used to re-weight the frequency of the words that are very common in the documents, like (“the”, “is” in English). The resulting tf-idf vectors are then normalized by the Euclidean norm:

$$tf\text{-}idf(d,t) = \frac{tf\text{-}idf(d,t)}{\|tf\text{-}idf(d,t)\|} \tag{4.3}$$

In order to understand the inverse document-frequency a numerical example is proposed: there are 100 documents and 99 of them contain the word "hello", while 49 contain the word "man". In the first case:

$$idf(t) = \log\left(\frac{1+100}{1+99}\right) + 1 \simeq 0 + 1 = 1,$$

giving a weight of one to the word present in almost all documents. In the second case:

$$idf(t) = \log\left(\frac{1+100}{1+49}\right) + 1 \simeq 0.7 + 1 = 1.7,$$

giving more weight to this word.

An output of this technique is presented in figure 4.4, where the sentences used are the same of example in figure 4.3.

In order to have a more complete extraction, in addiction to words, another possibility is to consider also the n-grams. They are a combination of n contiguous words in the corpus (figure 4.5).

```
Feature vectors:

[[1 1 0 0 2 1 0 0 1 0 0 1 1 1 2]
 [0 0 1 1 0 0 1 1 1 1 1 0 0 0 0]]


Feature names:

['afraid',
 'angry',
 'area',
 'central',
 'father',
 'find',
 'forest',
 'frigthened',
 'get',
 'jogging',
 'park',
 'person',
 'present',
 'reliable',
 'would']
```

Figure 4.3: *Bag of words model: feature vectors and feature names.*

```
Feature vectors:

[[0.263 0.263 0.    0.    0.525 0.263 0.    0.    0.187 0.    0.    0.263
  0.263 0.263 0.525]
 [0.    0.    0.392 0.392 0.    0.    0.392 0.392 0.279 0.392 0.392 0.
  0.    0.    0.   ]]


Feature names:

['afraid',
 'angry',
 'area',
 'central',
 'father',
 'find',
 'forest',
 'frigthened',
 'get',
 'jogging',
 'park',
 'person',
 'present',
 'reliable',
 'would']
```

Figure 4.4: *TF-IDF model: feature vectors and feature names.*

```
Feature vectors:

[[0.202 0.202 0.202 0.    0.    0.    0.    0.404 0.202 0.202 0.202 0.202
  0.    0.    0.    0.144 0.    0.202 0.    0.    0.    0.    0.202 0.202
  0.202 0.202 0.202 0.202 0.404 0.202 0.202]
 [0.    0.    0.    0.283 0.283 0.283 0.283 0.    0.    0.    0.    0.
  0.283 0.283 0.283 0.201 0.283 0.    0.283 0.283 0.283 0.283 0.    0.
  0.    0.    0.    0.    0.    0.    0.   ]]


Feature names:

['afraid',
 'afraid father',
 'angry',
 'area',
 'area get',
 'central',
 'central park',
 'father',
 'father find',
 'father would',
 'find',
 'find reliable',
 'forest',
 'forest central',
 'frigthened',
 'get',
 'get frigthened',
 'get present',
 'jogging',
 'jogging forest',
 'park',
 'park area',
 'person',
 'person afraid',
 'present',
 'present would',
 'reliable',
 'reliable person',
 'would',
 'would angry',
 'would get']
```

Figure 4.5: *TF-IDF model for single words and 2-grams: feature vectors and feature names.*

4.2.1 Sparse matrix

In the corpus used as input to the algorithms there are thousands of tweets. Remembering what was said before in chapter 1: every tweet is a sentence or a period with a maximum length of two hundred eighty characters. It is clear that in a feature representation (figure 4.3, in case of two tweets) there are a lot of zeros, for those terms that have not been said. A matrix that contains mostly zero values is called sparse, in contrast to a matrix in which most of the values are non-zero, called dense. It is computationally expensive to represent and work with sparse matrices. In python language there are libraries developed to handle this type of matrices, like SciPy and its module csr_matrix. Figure 4.6 shows this type of representation: the row and column numbers of the non-zero elements are displayed in the parentheses and its value on the right.

4.3 Application

In this section a script developed in Python is proposed. To understand better the methods of chapter 3 algorithms of chapter 2 are used to construct some ensemble classifiers and numerical results are reported. Then, the best method according to its value of accuracy and F_1 score is chosen. The goal of the study is to use what was previously learned about scraping, ensemble methods and sentiment analysis to associate an emotion to some tweets.

4.3.1 Python script

As input, a data set available online with 7666 tweets and 7 different emotions is used. Several tests were done and at the end it was decided to perform a classification on three classes of the emotion data set: joy, sadness and fear. The script can be divided into the following sections:

- Import of libraries and definition of functions used for text normalization and the evaluation of performance.
- Definition of a Twitter class to connect to a specific query
- Text mining and feature extraction for tweets
- Training of some models, cross-validation and choice of the best one
- Voting methods
- Tree-based methods
- Stacking
- Download of tweets and emotion classification

```
Matrix as arrays:

[[1 0 0 1 0 0 0 1]
 [0 0 2 0 0 1 0 0]
 [0 0 0 2 0 1 0 0]]

CSR matrix representation:

  (0, 0)        1
  (0, 3)        1
  (0, 7)        1
  (1, 2)        2
  (1, 5)        1
  (2, 3)        2
  (2, 5)        1
```

Figure 4.6: *CSR representation for sparse matrix*

4.3.2 Numerical results

After the text mining procedure on the emotion data set, some models are trained. As was said before cross-validation can be used to avoid overfitting and have a more realistic value for the accuracy. An example of 5-fold cross-validation on decision trees is shown in figure 4.7. The first number is the mean of F_1 score for the five folds and the second one is its variance.

```
Decision tree model performance
Accuracy: 0.665
Precision: 0.67
Recall: 0.666
F1 Score: 0.665

F1 Score of 5-fold cross validation
[0.67437439 0.64889324 0.6841531  0.68656218 0.66478078]

F1 Score: 0.672 (+/- 0.014)
```

Figure 4.7: *5-fold cross-validation on decision tree*

Python allows to use some methods like GridSearchCV to make cross-validation over more parameters. An example is shown in figure 4.9, where random forest uses several numbers of trees, five or ten as minimum number of samples in a node and the Gini impurity or the Information gain as criterion of split. The best combination is chosen according to the higher value of accuracy. As in the previous example of cross-validation, the first value of the grid is a mean of all values obtained by a 5-fold cv and the second one is its variance. At the end the classification report, obtained with the homonym python method, is proposed.

In order to introduce the voting methods on Python a toy example on emotion data set is proposed. The algorithms used are k-nearest neighbors, decision trees and multinomial naive bayes. There are different possibilities to improve the accuracy of these models, simply modifying the hyperparameters like the number of neighbors, the minimum number of leaves etc...

The voting methods used are the plurality one with its mode, the mean and the weighted mean. It can be seen in figure 4.8 that the best ensemble method is obtained with the last one, that gives more importance to more accurate algorithms. In this case the first one has the highest value of accuracy, so during the combination it will have more influence. The vector of weights is chosen among one hundred different combinations. It should be noted that averaging the three previous algorithms results in an ensemble classifier with at least 0.69 of accuracy, that is higher than the worst in use (decision tree with 0.67).

The weights can be compared to the expected values of section 3.1. Remember that the proportion of weights should be dependent on the accuracy of individual learners [4]:

$$w_i \propto \log\left(\frac{p_i}{1-p_i}\right) \tag{4.4}$$

```
Evaluating the accuracy for each model:

K-nearest neighbors: 0.721
Decision Tree: 0.67
Multinomial Naive Bayes: 0.692

The ensemble model with MODE: 0.735
The ensemble model with MEAN: 0.738
The ensemble model with WEIGHTED MEAN: 0.742
The weights used are [[0.46 0.24 0.3]]
```

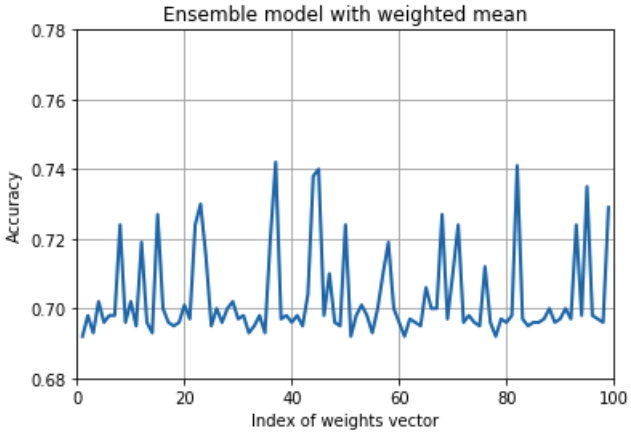


Figure 4.8: *Average methods developed in Python*

In figure 4.8 the weights used, randomly chosen in one hundred different combinations, are 0.46, 0.24 and 0.3. These values are not too far from those expected by formula 4.4.

A more complete example is shown in figure 4.10. There are sixteen algorithms in it:

- three logistic regression models
- three multinomial Naive Bayes
- two logistic regression models with stochastic gradient descent approach
- two support vector machines
- two decision trees
- one support vector machine with stochastic gradient descent approach
- one 50-nearest neighbors
- one 100-nearest neighbors
- one random forest

As has been introduced in chapter 2 naive Bayes and logistic regression classifiers are usually very used in sentiment analysis [21]. This is why they have been chosen three times. The other models are chosen to reach the best possible predictions.

In order to achieve better results five hundred of combinations of weights are generated. This method chooses the best and gives an accuracy of 0.797. The average prediction is always better than the worst algorithm used. In particular, the value of accuracy is always higher than 0.754 (figure 4.10).

```
Best parameters found:
{'criterion': 'gini', 'min_samples_leaf': 5, 'n_estimators': 400}

Grid scores:

0.72 (+/- 0.03) for {'criterion': 'entropy', 'min_samples_leaf': 5, 'n_estimators': 200}
0.73 (+/- 0.02) for {'criterion': 'entropy', 'min_samples_leaf': 5, 'n_estimators': 250}
0.73 (+/- 0.03) for {'criterion': 'entropy', 'min_samples_leaf': 5, 'n_estimators': 300}
0.73 (+/- 0.03) for {'criterion': 'entropy', 'min_samples_leaf': 5, 'n_estimators': 350}
0.74 (+/- 0.03) for {'criterion': 'entropy', 'min_samples_leaf': 5, 'n_estimators': 400}
0.73 (+/- 0.04) for {'criterion': 'entropy', 'min_samples_leaf': 5, 'n_estimators': 450}
0.74 (+/- 0.03) for {'criterion': 'entropy', 'min_samples_leaf': 5, 'n_estimators': 500}
0.68 (+/- 0.03) for {'criterion': 'entropy', 'min_samples_leaf': 10, 'n_estimators': 200}
0.67 (+/- 0.04) for {'criterion': 'entropy', 'min_samples_leaf': 10, 'n_estimators': 250}
0.69 (+/- 0.04) for {'criterion': 'entropy', 'min_samples_leaf': 10, 'n_estimators': 300}
0.68 (+/- 0.06) for {'criterion': 'entropy', 'min_samples_leaf': 10, 'n_estimators': 350}
0.69 (+/- 0.05) for {'criterion': 'entropy', 'min_samples_leaf': 10, 'n_estimators': 400}
0.69 (+/- 0.04) for {'criterion': 'entropy', 'min_samples_leaf': 10, 'n_estimators': 450}
0.69 (+/- 0.03) for {'criterion': 'entropy', 'min_samples_leaf': 10, 'n_estimators': 500}
0.73 (+/- 0.02) for {'criterion': 'gini', 'min_samples_leaf': 5, 'n_estimators': 200}
0.74 (+/- 0.03) for {'criterion': 'gini', 'min_samples_leaf': 5, 'n_estimators': 250}
0.74 (+/- 0.03) for {'criterion': 'gini', 'min_samples_leaf': 5, 'n_estimators': 300}
0.74 (+/- 0.03) for {'criterion': 'gini', 'min_samples_leaf': 5, 'n_estimators': 350}
0.74 (+/- 0.02) for {'criterion': 'gini', 'min_samples_leaf': 5, 'n_estimators': 400}
0.73 (+/- 0.03) for {'criterion': 'gini', 'min_samples_leaf': 5, 'n_estimators': 450}
0.74 (+/- 0.02) for {'criterion': 'gini', 'min_samples_leaf': 5, 'n_estimators': 500}
0.67 (+/- 0.03) for {'criterion': 'gini', 'min_samples_leaf': 10, 'n_estimators': 200}
0.67 (+/- 0.06) for {'criterion': 'gini', 'min_samples_leaf': 10, 'n_estimators': 250}
0.68 (+/- 0.03) for {'criterion': 'gini', 'min_samples_leaf': 10, 'n_estimators': 300}
0.68 (+/- 0.04) for {'criterion': 'gini', 'min_samples_leaf': 10, 'n_estimators': 350}
0.69 (+/- 0.05) for {'criterion': 'gini', 'min_samples_leaf': 10, 'n_estimators': 400}
0.69 (+/- 0.04) for {'criterion': 'gini', 'min_samples_leaf': 10, 'n_estimators': 450}
0.68 (+/- 0.06) for {'criterion': 'gini', 'min_samples_leaf': 10, 'n_estimators': 500}

Classification report:

              precision    recall  f1-score   support

        fear       0.85      0.64      0.73       302
         joy       0.61      0.83      0.70       261
     sadness       0.73      0.68      0.70       259
```

Figure 4.9: *Grid search with 5-fold cross-validation on random forest: accuracy values*

```
Evaluating the accuracy for models used:

Logistic regression: 0.77
Multinomial Naive Bayes: 0.708
Stochastic gradient descent, logistic: 0.777
Stochastic gradient descent, svm: 0.785
Support vector machine: 0.785
Decision tree: 0.7
50-nearest neighbors: 0.736
100-nearest neighbors: 0.725
Random forest: 0.706

The ensemble model with WEIGHTED MEAN: 0.797
```

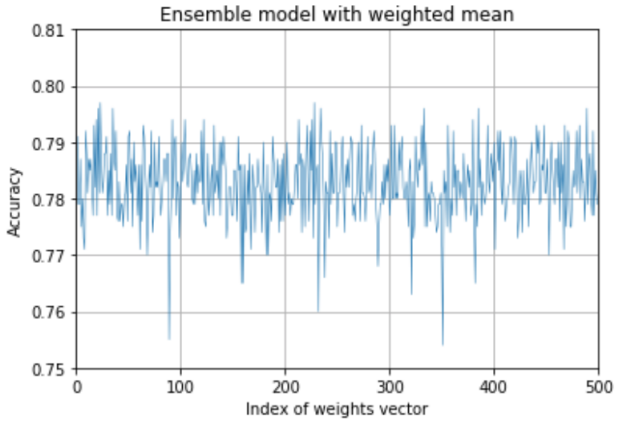


Figure 4.10: *Voting method: weighted mean of 16 algorithms.*

As discussed in chapter 3 ensemble methods are usually applied to decision tree-based classifiers. The well-known algorithms used in Python related to trees are:

- DecisionTreeClassifier: it implements CART algorithm for decision trees.
- BaggingClassifier: it implements bagging.
- GradientBoostingClassifier: it implements gradient boosting approach.
- RandomForestClassifier: it implements random forest algorithm.
- ExtraTreesClassifier: it implements extremely randomized trees.

BaggingClassifier and GradientBoostingClassifier allow to use several algorithms as base estimator. In this case DecisionTreeClassifier is chosen.

Extra randomized trees model is very similar to random forest except for two important things [23]. First, when it looks for the best split to separate the samples of a node into two groups, it tests random splits over fraction

of features and the best split is chosen among those. Instead, random forest tests all possible splits over fraction of features (that usually is its square root). Second, extra randomized trees model uses the whole learning sample, rather than bootstrap technique. As every ensemble method in classification problems, it yields the final prediction by majority vote. These two differences allow to reduce a bit the variance of the model, at the expense of a slight increase in bias.

In table 4.1 the performance of tree-based models are shown, sorted by decreasing values of F_1 score. The best method is Extra trees, which is trained with one hundred eighty-five trees and entropy as the function to measure the quality of a split. The other parameters that can be tuned, like the maximum depth of the tree, the number of features etc... are set to the default values.

The performance of these methods can easily improved by tuning the hyperparameters. It can be seen in figure 4.11 that with more than two hundred trees the F_1 score is always almost greater than 0.77.

After that, using the python module StackingClassifier from mlxtend library, the stacking procedure is developed. It allows to perform a 5-fold cross-validation and, of course, to choose the learners of the first level and the meta classifier. The models used are five: k-nearest neighbors, random forest, multinomial naive bayes, logistic regression and decision tree. The second learner is logistic regression model (figure 4.12). The result is competitive, but probably due the short size of input data set one cannot fully appreciate the performance of this technique. It is a good method to improve the accuracy of some models but, in this case, it cannot outperform support vector machine or extra randomized trees.

Algorithm	F1 score	Accuracy	Precision	Recall
Extra trees	0.787	0.788	0.788	0.787
XGBoost	0.741	0.742	0.751	0.737
Random forest	0.739	0.74	0.74	0.739
Bagging tree	0.727	0.725	0.728	0.727
Decision tree	0.686	0.686	0.686	0.688

Table 4.1: *Tree-based methods with evaluation of performance, sorted by F_1 score*

A summary of all the algorithms used in the project can be seen in the table 4.2.

4.3.3 Twitter sentiment analysis

An application is proposed: some tweets, collected from a famous Twitter profile, are labelled with an emotion using the best method among the previous ones (the ensemble method with sixteen models). They were downloaded so that they have a minimum length of one hundred characters and no more than three hashtags. Otherwise, they would have been too difficult to classified.

For each emotion in table 4.3 the five most recent tweets are shown. Each tweet has three columns of reference: the expected emotion, the detected one and a probability value for each class. The blue value is the highest in the vector and it determines the detected class. The ambiguous predictions are colored in

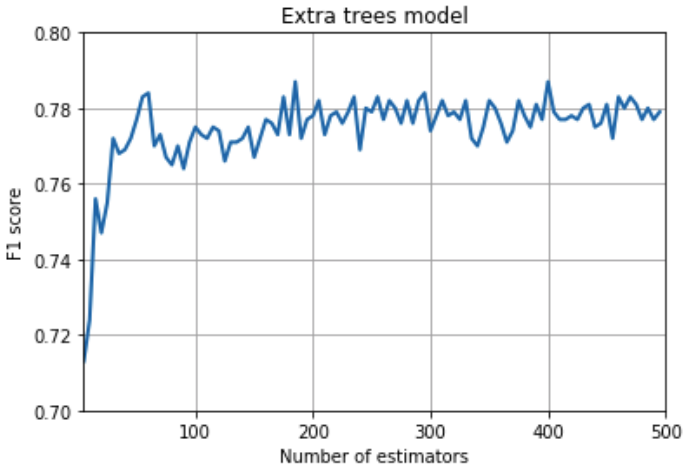


Figure 4.11: F_1 *score of extra trees method depending on the number of trees*

```
Stacking model:
    5-fold cross validation
    Logistic regression as meta classifier

Accuracy: 0.739 (+/- 0.020) [K-nearest neighbors]
Accuracy: 0.735 (+/- 0.015) [Random forest]
Accuracy: 0.711 (+/- 0.012) [Multinomial Naive Bayes]
Accuracy: 0.686 (+/- 0.019) [Decision tree]
Accuracy: 0.772 (+/- 0.019) [StackingClassifier]
```

Figure 4.12: *Stacking method*

Algorithm	F1 score	Accuracy	Precision	Recall
Ensemble, 16 models	0.796	0.797	0.796	0.797
Extra trees	0.787	0.788	0.788	0.787
Stochastic gradient descent, svm	0.787	0.788	0.788	0.786
Support vector machine	0.784	0.785	0.785	0.783
Stochastic gradient descent, logistic	0.779	0.78	0.78	0.778
Stacking of knn, rf, mnb, dt	0.772	0.772	0.774	0.772
Logistic regression	0.769	0.77	0.77	0.769
XGBoost	0.741	0.742	0.751	0.737
Random forest	0.739	0.74	0.74	0.739
Adaboost mnb	0.734	0.734	0.737	0.734
Bagging tree	0.727	0.725	0.728	0.727
K-nearest neighbors	0.725	0.725	0.727	0.725
Multinomial Naive Bayes	0.708	0.708	0.709	0.709
Decision tree	0.686	0.686	0.686	0.688

Table 4.2: *Algorithms used in Python with evaluation of performance, sorted by* F_1 *score*

yellow, while the mistakes are in red. It should be noted that, sometimes, the algorithm is pretty sure about the forecast. For example, in the eighth tweet the value of the predicted class is 0.9, while in the fifth, representing a wrong classification, the model is uncertain about joy or sadness (probability of 0.41 versus 0.46).

The table is created as a pandas data frame. The column "Emotion" is added to the previous ones after exporting the file to Excel.

Considering the not ambiguous tweets of table 4.3, just two tweets among eleven seem to be misclassified. It means that about 82% are correctly classified and this reflects the results previously obtained.

Emotion	Emotion detected	Probabilities f/j/s	Tweet
sadness	sadness	[0.15 0.25 0.6]	*They refuse to see or acknowledge the Death, Crime, Drugs and Human Trafficking at our Southern Border!*
sadness	sadness	[0.19 0.21 0.6]	*The Fake News photoshopped pictures of Melania, then propelled conspiracy theories that it's actually not her by my side in Alabama and other places. They are only getting more deranged with time!*
sadness	sadness	[0.09 0.14 0.77]	*Defying voters, the Governor of California will halt all death penalty executions of 737 stone cold killers. Friends and families of the always forgotten VICTIMS are not thrilled, and neither am I!*
sadness	sadness	[0.11 0.42 0.47]	*....Scott has helped us to lower drug prices, get a record number of generic drugs approved and onto the market, and so many other things. He and his talents will be greatly missed!*
joy	sadness	[0.13 0.41 0.46]	*Congratulations @Toyota! BIG NEWS for U.S. Auto Workers! The USMCA is already fixing the broken NAFTA deal.*
joy	joy	[0.24 0.56 0.2]	*More people are working today in the United States, 158,000,000, than at any time in our Country, history. That is a Big Deal!*
joy/sadness	joy	[0.18 0.5 0.32]	*Donald Trump's Approval Rating is at or near his highest level ever. The media is not being honest about what is happening in this Country.*
joy	joy	[0.05 0.9 0.05]	*Aluminum prices are down 12% since I instituted Tariffs on Aluminum Dumping - and the U.S. will be taking in Billions, plus jobs. Nice!*
joy	joy	[0.17 0.56 0.27]	*On International Women's Day, we honor women worldwide for their vital role in shaping and strengthening our communities, families, governments, and businesses...*
sadness/joy	joy	[0.02 0.84 0.14]	*I cannot believe the level of dishonesty in the media. It is totally out of control, but we are winning!*
fear	fear	[0.44 0.16 0.4]	*Democrats will have a unanimous vote on a 20% issue in opposing Republican Senators tomorrow. The Dems are for Open Borders and Crime!*
joy/fear	fear	[0.41 0.37 0.22]	*My Administration looks forward to negotiating a large scale Trade Deal with the United Kingdom. The potential is unlimited!*
fear/sadness	fear	[0.57 0.02 0.41]	*I agree with Rand Paul. This is a total disgrace and should NEVER happen to another President!*
fear	fear	[0.51 0.19 0.3]	*Just spoke with Jacinda Ardern, the Prime Minister of New Zealand, regarding the horrific events that have taken place over the past 24 hours. I informed the Prime Minister....*
joy	fear	[0.44 0.36 0.2]	*....and I did not increase their second traunch of Tariffs to 25% on March 1st. This is very important for our great farmers - and me!*

Table 4.3: *Emotion analysis: expected and detected feelings for fifteen tweets. The ambiguous tweets are colored in yellow, while those wrongly classified are red.*

Chapter 5

Conclusions

The aim of this work was to carry out a sentiment analysis. With the purpose of doing it as precisely as possible, I used what are commonly called ensemble methods. The idea of combining multiple classifiers starts from the principle that there is no perfect machine learning algorithm. Every model has its limitations and weaknesses, so the goal is to consider the best possible decision taken overall in order to overcome these boundaries.

After the preprocessing phase, I trained several ensemble classifiers. Among the most famous there are certainly bagging, boosting, stacking and the voting methods. In this case, it emerged that the best algorithm is a voting one with heterogeneous models. It is very important to remember that the choice of the algorithm to apply is strongly dependent on the data to analyze. It also must be chosen on the basis of the input data set.

I applied the stacking method with two levels to my study, but the results were not as significant as in voting methods. Another input data set could be used, and the development of new procedures based on more levels will be the aim of further studies.

Once the best algorithm was chosen, a sentiment analysis was performed on the tweets of a famous politician. The results obtained showed that in about 80% of cases the algorithm is able to correctly classify the tweets.

This work could also be extended to other social networks and different aspects of sentiment analysis could be considered, such as customer satisfaction on some commercial products.

Appendix A

Programs instructions

This appendix helps the user who wants to launch the programs. The following instructions concern the program about politicians (*Twitter_politicians*) and the program of emotion analysis (*Twitter_emotion*).

A.1 Twitter_politicians

1. First of all, download the last version of Anaconda and select "Add Anaconda to my PATH environment variable" during the installation phase. Anaconda is used to launch the .ipynb program from Jupyter Notebook [1].
2. Download the last version of Python to launch the .py program during scheduling. This can be generated from the .ipynb file, by running "Jupyter nbconvert -to script <file.ipybn>" from the terminal. In this case, comment the portion of code referred to the word cloud, since some libraries, as Wordcloud and PIL, cannot be used with the native Python interpreter.
3. Download the required libraries: it can be done by terminal running "py -m pip install <library name>" or "pip install <library name>" statement.
4. Enter the login credentials of the twitter application, as explained later.
5. Set the directory where the output files will be printed.
6. Enter the date of the first day of scheduling.
7. Run all.

A.2 Twitter_emotion

1. The same first four points as in A.1.
2. Enter the path of the emotion data set.

[1] Jupyter Notebook is an interactive Web-based computing environment for creating documents. A Jupyter Notebook document is a JSON type, which is composed by an ordered list of cells that may contain code, text, math and graphs.

3. Enter the path for the output data sets.
4. Run all.

Appendix B

Python codes

This appendix shows some of the most interesting pieces of code developed in Python.

Listing B.1: Twitter class

```
class TwitterClient(object):

    def __init__(self):

        #Credentials
        consumer_key='HTdVyKp4I2SZAtJt3BjxyFs1e'
        consumer_secret='Q221FPjVZrPvtxCOBeN4sUehytU4Rjs8E0CM4BQSKwmBHxbiCD'
        access_token='977105053227118592-NSPbMDxFsHtOTMkqHUIo8Ibn93I8cf4'
        access_secret='mmykGQLvzIOBAXd11ZyfPOnCsB0ByUYxBYfCTzKzDyQrZ'

        #Authentication
        try:
            self.auth = OAuthHandler(consumer_key, consumer_secret)
            self.auth.set_access_token(access_token, access_secret)
            self.api = tweepy.API(self.auth)

        except:
            print("Error: Authentication Failed")

    def get_tweets(self, query, count = 10):

        tweets = []

        try:
            #user_timeline for user, otherwise search
            load_tweets = self.api.user_timeline(screen_name = query,
                                                 include_rts = False,
                                                 tweet_mode = 'extended',
                                                 count=count, language='en')

            mytweet_hash = []
```

```
            mytweet_at = []
            for status in load_tweets:
                if len(status.full_text)>100 and len(status.full_text)<200:
                    mytweet=status.full_text
                    if re.findall(r'\#\w+', mytweet):
                        mytweet_hash.append(re.findall(r'\#\w+', mytweet))
                        if str(mytweet_hash).count('#') < 4:
                            tweets.append(mytweet)
                    if re.findall(r'\@\w+', mytweet):
                        mytweet_at.append(re.findall(r'\@\w+', mytweet))
                        if str(mytweet_at).count('@') < 4:
                            tweets.append(mytweet)
                    else:
                        tweets.append(mytweet)


        except tweepy.TweepError as e:
            print("Error : " + str(e))

        return tweets
```

Listing B.2: Data preparation and feature extraction

```
###### DATA PREPARATION AND FEATURE EXTRACTION ######

Emotion_list=[]
Tweet_list=[]

#mysent= [joy fear anger sadness disgust shame guilt]
mysent= ['joy', 'sadness', 'fear']

with open('/Users/Gianluca/Desktop/Tesi/SENTIMENT/emotion.csv', 'r') as f:
    reader = csv.DictReader(f, delimiter=';')
    for row in reader:
        if row['Emotion'] in mysent:
            Emotion_list.append(row['Emotion'])
            Tweet_list.append(row['Tweet'])


Emotion_df = pd.DataFrame(
    {'Emotion': Emotion_list,
     'Tweet': Tweet_list
    })

Emotion_df = Emotion_df.sample(frac=1).reset_index(drop=True)

cont=Emotion_df['Tweet']
sent=Emotion_df['Emotion']
```

```
train_X, val_X, train_y, val_y = train_test_split(cont,sent,random_state=1)

# data preparation
norm_train_tweet = normalize_corpus(train_X,lemmatize=True,only_text_chars=True)

# feature extraction
vectorizer, train_features = build_feature_matrix(documents=norm_train_tweet,
                                                  feature_type='tfidf',
                                                  ngram_range=(1, 2),
                                                  min_df=0.0, max_df=1.0)

# normalize tweets
norm_test_tweet = normalize_corpus(val_X,lemmatize=True,only_text_chars=True)

# extract features
test_features = vectorizer.transform(norm_test_tweet)
```

Listing B.3: Calibrated probabilities and ROC curves

```
model_0 = SGDClassifier(loss='hinge', penalty='elasticnet', alpha=0.0001,
                                        max_iter=100, random_state=1, tol=1e-3)

model = CalibratedClassifierCV(model_0, cv=5, method='sigmoid')

model.fit(train_features, train_y)

y_predict = model.predict(test_features)
predicted_probas = model.predict_proba(test_features)

# Plot curves for multi-classification
import matplotlib.pyplot as plt
import scikitplot as skplt
skplt.metrics.plot_roc(val_y, predicted_probas, plot_micro=False, plot_macro=False,
                       cmap=plt.get_cmap('Spectral') )
plt.show()
```

Listing B.4: Models and confusion matrices

```
def models():
    ''' algorithms '''
    svm=LinearSVC()
    sgd_svm = SGDClassifier(loss='hinge', penalty='elasticnet', alpha=0.0001, max_iter=100,
                        random_state=1, tol=1e-3)
    log = SGDClassifier(loss='log', penalty='elasticnet', alpha=0.0001, max_iter=100,
                        random_state=1, tol=1e-3)
    log_reg = LogisticRegression(random_state=1)
    tree = DecisionTreeClassifier(random_state=1,min_samples_leaf=10)
    mnb = MultinomialNB(alpha=1.0e-10)
    n_neighbors = 50
    knc = neighbors.KNeighborsClassifier(n_neighbors, weights='distance')
```

```
    rf = RandomForestClassifier(min_samples_leaf=10, n_estimators=100)
    et = ExtraTreesClassifier(n_estimators=375, criterion='entropy')
    n_estim = 5
    bagS = OneVsRestClassifier(BaggingClassifier(SVC(kernel='linear', probability=True),
                                                  max_samples=1.0 / n_estim,
                                                  n_estimators=n_estim))
    bagT = BaggingClassifier(base_estimator=tree, n_estimators=50)
    xgb = GradientBoostingClassifier()
    ada = AdaBoostClassifier()

    myclf= {
            "Linear Support vector machines" : svm,
            "Stochastic gradient descent, svm" : sgd_svm,
            "Stochastic gradient descent, logistic" : log,
            "Logistic regression" : log_reg,
            "Decision Tree" : tree,
            "Multinomial Naive Bayes" : mnb,
            "K-nearest neighbors" : knc,
            "Random forest" : rf,
            "Extra trees" : et,
            "Bagging TREE" : bagT,
            "Bagging SVM" : bagS,
            "Adaboost" : ada,
            "XGBoost" : xgb
            }

    return myclf


#Fit and predict
my_eval=[]
for clf in models():
    time0=0
    time1=0
    time0 = time()
    myclf=models()[clf].fit(train_features, train_y)
    time1 = time()
    predicted_sentiments = myclf.predict(test_features)
    #print('Time to fit', clf, 'is %f second' %(time1-time0))
    my_perf=display_evaluation_metrics(true_labels=val_y,
                          predicted_labels=predicted_sentiments)
    my_eval.append(f1_score_performance(true_labels=val_y,
                          predicted_labels=predicted_sentiments))
    print('\n')

#CLASSIFICATION REPORT and CONFUSION MATRIX
index, value = max(enumerate(my_eval), key=operator.itemgetter(1))
print('The best algorithm is', list(models())[index])
best_model=models()[list(models())[index]]
my_best=best_model.fit(train_features, train_y)
```

```
best_prediction=my_best.predict(test_features)
display_classification_report(val_y, best_prediction, dec=3, classes=mysent)


#CONFUSION MATRIX
classes=mysent

plt.figure()
plot_confusion_matrix(create_confusion_matrix(val_y, best_prediction, labels=classes),
                      classes=classes,
                      title='Confusion matrix',
                      cmap=plt.cm.Greens)

plt.figure()
plot_confusion_matrix(create_confusion_matrix(val_y, best_prediction, labels=classes),
                      classes=classes, normalize=True,
                      title='Normalized confusion matrix',
                      cmap=plt.cm.Blues)

plt.show()
```

Listing B.5: Voting classifier and ensemble model with weighted mean

```
#Voting classifier
model = VotingClassifier(estimators=[('MODEL1', model1),
                                     ('MODEL2', model2),
                                     ('MODEL3', model3),
                                     ('MODEL4', model4),
                                     ('MODEL5', model5),
                                     ('MODEL6', model6),
                                     ('MODEL7', model7),
                                     ('MODEL8', model8),
                                     ('MODEL9', model9),
                                     ('MODEL10', model10),
                                     ('MODEL11', model11),
                                     ('MODEL12', model12),
                                     ('MODEL13', model13),
                                     ('MODEL14', model14),
                                     ('MODEL15', model15),
                                     ('MODEL16', model16)],
                                     voting='soft')

model.fit(train_features, train_y)
myVoting = model.predict(test_features)

##############################################################################
#Weighted mean
model1 = neighbors.KNeighborsClassifier(n_neighbors, weights='distance')
model2 = DecisionTreeClassifier(random_state=1,min_samples_leaf=10)
```

```
model3 = MultinomialNB(alpha=1.0e-10)

model1.fit(train_features, train_y)
my_model1 = model1.predict(test_features)
print("\nEvaluating the accuracy for each model:")
print("\nK-nearest neighbors:", accuracy_performance(val_y, my_model1,
                                        positive_class=1))

model2.fit(train_features, train_y)
my_model2 = model2.predict(test_features)
print("Decision Tree:", accuracy_performance(val_y, my_model2,
                                        positive_class=1))

model3.fit(train_features, train_y)
my_model3 = model3.predict(test_features)
print("Multinomial Naive Bayes:", accuracy_performance(val_y, my_model3,
                                        positive_class=1))

pred1=model1.predict_proba(test_features)
pred2=model2.predict_proba(test_features)
pred3=model3.predict_proba(test_features)

myweight = []
myscore = []

for i in range(1,100):
    a=np.random.dirichlet(np.ones(3),size=1).round(2)
    myweight.append(a)

    my_array = []
    mediapredw=pred1*a.item(0) + pred2*a.item(1) + pred3*a.item(2)

    for row in mediapredw:
        index, value = max(enumerate(row), key=operator.itemgetter(1))
        if index == 0:
            my_array.append('fear')
        if index == 1:
            my_array.append('joy')
        if index == 2:
            my_array.append('sadness')

    myscore.append(accuracy_performance(val_y, my_array, positive_class=1))

index_w, value_w = max(enumerate(myscore), key=operator.itemgetter(1))

print('The ensemble model with WEIGHTED MEAN:', value_w)
print('The weights used are', myweight[index_w])
print('\n')

plt.xlabel('Index of weights vector')
```

```
plt.ylabel('Accuracy')
plt.title('Ensemble model with weighted mean')

plt.plot(range(1,100), myscore, linewidth=2.0)
plt.axis([0, 100, 0.68, 0.78])
plt.grid(True)
plt.show()
```

Listing B.6: Models on CSV

```
#WRITING ON CSV
my_models=[]
my_f1=[]
my_acc=[]
my_prec=[]
my_rec=[]

for clf in models():
    my_models.append(clf)
    myclf=models()[clf].fit(train_features, train_y)
    predicted_sentiments = myclf.predict(test_features)

    my_f1.append(f1_score_performance(true_labels=val_y,
                        predicted_labels=predicted_sentiments))
    my_acc.append(accuracy_performance(true_labels=val_y,
                        predicted_labels=predicted_sentiments))
    my_prec.append(precision_performance(true_labels=val_y,
                        predicted_labels=predicted_sentiments))
    my_rec.append(recall_performance(true_labels=val_y,
                        predicted_labels=predicted_sentiments))

mypd = pd.DataFrame(np.column_stack([my_models, my_f1, my_acc, my_prec, my_rec]),
             columns=['Algorithm', 'F1 score', 'Accuracy', 'Precision', 'Recall'])
mypd_sort=mypd.sort_values(by=['F1 score'],ascending=False)
#print(mypd_sort)

mypd_sort.to_csv("/Users/Gianluca/Desktop/models.csv",
                                encoding='utf-8', index=False, sep=';')
```

Listing B.7: Stacking classifier

```
meta = LogisticRegression(solver='lbfgs', multi_class='auto')
sclf = StackingClassifier(classifiers=[model5, model14, model2, model16],
                          use_probas=True,
                          average_probas=False,
                          meta_classifier=meta)

print('Stacking model:')
print('    5-fold cross validation')
print('    Logistic regression as meta classifier')
```

```
print()

for clf, label in zip([model5, model14, model2, model16, sclf],
                      ['K—nearest neighbors',
                       'Random forest',
                       'Multinomial Naive Bayes',
                       'Decision tree',
                       'StackingClassifier']):

    scores = model_selection.cross_val_score(clf, train_features, my_train_y,
                                             cv=5, scoring='accuracy')
    print("Accuracy: %0.3f (+/— %0.3f) [%s]"
          % (scores.mean(), scores.std(), label))
```

Listing B.8: Sentiment analysis

```
if __name__ == "__main__":

    query = '@realDonaldTrump'
    api = TwitterClient()

    my_best_model = CalibratedClassifierCV(best_model_0, cv=5, method='sigmoid')
    my_best_model.fit(train_features, train_y)

    tweets = api.get_tweets(query = query, count = 200)

    tweets_norm=normalize_corpus(tweets,
                            lemmatize=True,
                            only_text_chars=True)

    tweets_test_features = vectorizer.transform(tweets_norm)

    predicted_tweets = my_best_model.predict(tweets_test_features)

    probab = my_best_model.predict_proba(tweets_test_features).round(2)

    proba = []
    for el in probab:
        proba.append(str(el))

    joy_tweets = []
    sadness_tweets = []
    fear_tweets = []
    joy_sent = []
    sadness_sent = []
    fear_sent = []
    joy_prob = []
    fear_prob = []
    sadness_prob = []
```

```
    for i in range(0,len(predicted_tweets)):
        if predicted_tweets[i] == 'joy':
            joy_tweets.append(tweets[i])
            joy_sent.append('joy')
            joy_prob.append(proba[i])

        if predicted_tweets[i] == 'sadness':
            sadness_tweets.append(tweets[i])
            sadness_sent.append('sadness')
            sadness_prob.append(proba[i])

        if predicted_tweets[i] == 'fear':
            fear_tweets.append(tweets[i])
            fear_sent.append('fear')
            fear_prob.append(proba[i])

def output_sent(sent,tweets,prob,emotion):
    mypd = pd.DataFrame(np.column_stack([sent, prob, tweets]),
                                    columns=['Emotion', 'Probabilities', 'Tweet'])
    return mypd
```

#Output

```
myf = output_sent(fear_sent,fear_tweets,fear_prob,'fear')
myj = output_sent(joy_sent,joy_tweets,joy_prob,'joy')
mys = output_sent(sadness_sent,sadness_tweets,sadness_prob,'sadness')

mys2 = mys[-5:]
myj2 = myj[-5:]
myf2 = myf[-5:]

mypd.concat([mys, myj, myf])

mypath="/Users/Gianluca/Desktop/"

mypd.to_csv(mypath + query + ".csv", encoding='utf-8', index=False, sep=';')
```

Appendix C

SAS codes

This appendix shows some of the most interesting pieces of code written in SAS.

Listing C.1: Import and charts

```
PROC IMPORT OUT= WORK.parole_
            DATAFILE= "C:\\Users\gfino\Desktop\SEMANTIC\OUTPUT\\
        2018-05-16_Twitter_Politici_Data_Frame.txt"
            DBMS=DLM REPLACE;
     DELIMITER='3B'x;
     GETNAMES=YES;
     DATAROW=2;
RUN;

/*Data frame dopo 2 giorni di simulazione */
PROC IMPORT OUT= WORK.parole_2g
            DATAFILE= "C:\\Users\gfino\Desktop\SEMANTIC\OUTPUT\0_HASH\\
        2018-05-09_2018-05-11_Twitter_Politici_Data_Frame_app.txt"
            DBMS=DLM REPLACE;
     DELIMITER='3B'x;
     GETNAMES=YES;
     DATAROW=2;
RUN;

proc sort data=parole_ out=parole_plus;
by descending Totale;
run;

title 'Frequenza delle parole';

axis1 stagger label=none;
axis2 label=(a=90 'Frequenza');
pattern1 v=solid color=blue; /* Per barre colore rosso */

/* Create space at the bottom of the graph */
footnote h=.01 in ' ';

ods rtf file='temp.rtf' style=HTMLBLUE;

data parole_60;
```

```
        set parole_plus;
        where Totale > 60;
        if parole in ("#forzaitalia", "@emmabonino" , "@pietrograsso")
                then delete;
proc print data=Parole_60;
run;
ods rtf close;

ods rtf file='temp.rtf' style=HTMLBLUE;
proc gchart data=parole_60 ;
   vbar parole / sumvar=Totale maxis=axis1 raxis=axis2 levels=all
        descending;
run;
quit;
ods rtf close;
```

Listing C.2: Correlation

```
/* CORRELAZIONE */
TITLE 'CORRELAZIONE POLITICI';
ods rtf file='temp.rtf' style=HTMLBLUE;
TITLE 'correlazione 9 politici';
proc corr data=Parole_2g;
        var Berlusconi Bonino Di_Maio Grasso Grillo
                Maroni Meloni Renzi Salvini;
run;
ods rtf close;

/*****************************************************************/
/* GRAFICO SOVRAPPOSTO */
TITLE 'Andamento frequenza parole';

data parole_2g_60;
        set parole_2g;
        where Totale > 60;
        if parole in ("#forzaitalia", "@emmabonino" , "@pietrograsso")
                then delete;
proc print data=Parole_2g_60;
run;

/* Disegno 9 politici */

axis1 stagger label=none;
axis2 label=(a=90 'Frequenza') order=(0 to 75 by 10);

symbol1 interpol=join
        value=C
                font=marker
                width=6
        color=_style_;
symbol2 interpol=join
        value=C
        font=marker
                width=2
```

```
        color=_style_ ;
symbol3 interpol=join
        value=C
        font=marker
                width=2
        color=_style_ ;
symbol4 interpol=join
        value=C
        font=marker
                width=2
        color=_style_ ;
symbol5 interpol=join
        value=C
                width=2
        font=marker
        color=_style_ ;
symbol6 interpol=join
        value=C
        font=marker
                width=2
        color=_style_ ;
symbol7 interpol=join
        value=C
        font=marker
                width=6
        color=_style_ ;
symbol8 interpol=join
        value=C
        font=marker
                width=2
        color=_style_ ;
symbol9 interpol=join
        value=C
        font=marker
                width=2
        color=_style_ ;

legend1 label=none
        position=(top center inside)
        mode=share;

ods rtf file='temp.rtf' style=HTMLBLUE;
proc gplot data=parole_7g_60;
   plot berlusconi*parole bonino*parole di_maio*parole
        grasso*parole grillo*parole maroni*parole
        meloni*parole renzi*parole salvini*parole/
                        overlay legend=legend1
                          vref=1000 to 5000 by 1000
                          lvref=2
                          haxis=axis1 hminor=4
                          vaxis=axis2 vminor=1;
run;
quit;
ods rtf close;
```

Bibliography

[1] S.Bird, and E. Klein. *Natural Language Processing with Python.* O'Reilly Media, 2009.

[2] Sas Institute. *Step-by-Step Programming with Base SAS 9.4.* SAS Publishing, 2013.

[3] I. Idrid. *Python Data Analysis.* Packt Pub Ltd, 2014.

[4] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms.* Taylor & Francis Group, LLC, 2012.

[5] T. Hastie, R. Tibshirami, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009.

[6] G. James, D. Witten, T. Hastie, and R. Tibshirami. *An Introduction to Statistical Learning: with Applications in R.* Springer, 2013.

[7] K. Deng. *Omega: on-line memory-based general purpose system classifier.* PhD Thesis, Tech. Report, CMU-RI-TR-98-33, 1998.

[8] M. Parsian. *Data Algorithms.* O'Reilly Media, 2015.

[9] X. He, Z. Wang, C. Jin, Y. Zheng, and X. Xue. *A simplified multi-class support vector machine with reduced dual optimization.* Pattern Recognition Letters archive, 33, 2012.

[10] S. Ruder. *An overview of gradient descent optimization algorithms.* arXiv:1609.04747, 2017.

[11] C. Ferri, J. Hernández-Orallo, M. A. Salido. *Volume under the ROC Surface for Multi-class Problems.* Machine Learning: ECML 2003, pp. 108-120.

[12] A. Srinivasan. *Note on the Location of Optimal Classifiers in N-dimensional ROC Space.* Technical Report PRG-TR-2-99, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, 1999.

[13] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms.* A JOHN WILEY & SONS, 2004.

[14] M. Kuhn, and K. Johnson. *Applied Predictive Modeling.* Springer, 2013.

[15] A. Niculescu-Mizil, and R. Caruana. *Predicting Good Probabilities With Supervised Learning.* ICML '05 Proceedings of the 22nd international conference on Machine learning, 2007, pp. 625-632.

[16] L. Breiman. *Bagging Predictors*. Machine Learning, 1996, 24, pp. 123-140.

[17] L. Breiman. *Random Forests*. Machine Learning, 2001, 45, pp. 5–32.

[18] R. Schapire, and E. Freund *Boosting: Foundations and Algorithm*. MIT Press, 2012.

[19] J. Zhu, H. Zou, S. Rosset, and T. Hastie *Multi-class AdaBoost*. Statistics and Its Interface, 2009, 2, pp. 349–360.

[20] S. Džeroski, and B. Ženko. *Is Combining Classifiers with Stacking Better than Selecting the Best One?* Machine Learning, 2004, 54, pp 255–273.

[21] D. Sarkar. *Text Analytics with Python*. Springer, 2016.

[22] F. Jeffrey. *Mastering Regular Expressions*. O'Reilly Media, 2009.

[23] P. Geurts, D. Ernst, and L. Wehenkel. *Extremely Randomized Trees* Machine Learning, 2006, 63, pp 3-42.