MASTER'S THESIS

Interuniversity Master in Statistics and Operations Research UPC-UB

Title: Covid-19 detection based on cough analysis using statistical learning methods

Author: Matteo Perillo

Advisor: Jordi Castro Pérez

Department: Statistics and Operations Research

University: Universitat Politècnica de Catalunya

Academic year: 2020/2021

UNIVERSITAT POLITÈCNICA DE C.
BARCELONATECH
Facultat de Matemàtiques i Estadís

UNIVERSITAT DE BARCELONA

Table of Contents

Chapter 1

Introduction

1.1 COVID-19: A global emergency

COVID-19 (COronaVirus Disease of 2019), caused by the Severe Acute Respiratory Syndrome (SARS-CoV2) virus, was announced as a global pandemic on February 11, 2020 by the World Health Organization (WHO) (Cucinotta and Vanelli, 2020).
The most common symptoms of COVID-19 are fever, fatigue and a dry cough (Wang et al., 2020). Other symptoms include shortness of breath, joint pain, muscle pain, gastrointestinal symptoms and loss of smell or taste (Carfì et al., 2020). At the time of writing, the global pandemic is still ongoing, and almost 171 million COVID-19 cases have been detected, along with more than 3,550,000 deaths confirmed to be due to the disease (Johns Hopkins University, 2021). This situation has caused a lot of health systems all over the world to be overrun not only by the significant need of cares for the people affected by the disease, but also by the need of testing people to detect who has the disease. This point is crucial in order to promptly treat the patients who present COVID-19 and isolate them, avoiding the chain of infection to keep growing.
For this reason, in the last year a lot of researchers have been focusing on trying to set up a fast, economic and reliable instrument to test people for COVID-19. In particular, relying on other studies which in the past brought satisfying results in the detection of other respiratory diseases, several attempts of detecting the presence of COVID-19 using the analysis of the cough sounds have been carried out. Some of them have obtained pretty encouraging results, even if they still present some issues to overcome and need to go through more robust validation before being considered clinically reliable as COVID-19 testing tools.

1.2 Coughing: the voice of the respiratory diseases

Coughing is one of the predominant symptoms of COVID-19. A cough, also known as tussis, is a voluntary or involuntary act that clears the throat and breathing passage of foreign particles, microbes, irritants, fluids, and mucus, through the rapid expulsion of air from the lungs. A cough in composed by three phases: first the inhalation (breathing in), then the increase of the pressure in the throat and lungs with the vocal cords closed, and at the end the explosive release of air when the vocal cords open, giving a cough its characteristic sound (Newman, 2017).
Coughing is a symptom of more than 100 diseases. These diseases can be pretty different among them and therefore affect differently the respiratory system. In most cases, the infection is in the upper respiratory tract and affects the throat: examples are flu, common cold and laryngitis. In other cases, such as bronchitis or pneumonia, also the lungs and/or the airways lower down from the windpipe are infected. Some people are also affected by chronic cough, which can be due to chronic respiratory diseases like asthma or due to other chronic diseases such as gastro-oesophageal reflux disease, or even due to other conditions like smoking.
Consequently, coughs caused by different conditions can be pretty different among themselves. In particular the behaviour of the glottis has been proven to change according to the disease affecting the respiratory system, causing the cough of the sick people to be in turn different depending on their disease (Korpáš et al., 1996; Knocikova et al., 2008).
Based on these premises, several studies have been conducted to analyse the acoustics of the cough and classify people based on the disease which affects them. Some of them obtained remarkable results (Pramono et al., 2016; Rudraraju et al., 2020). All these studies tried to use machine learning algorithms of different type and complexity to process cough recordings and classify the observations, dividing the subjects in healthy and unhealthy (binary classification problems), and/or trying to identify the disease affecting the unhealthy people (multi class problems).

1.3 Detecting COVID19: A new challenge for Machine Learning

Trying to follow what has been done in the past for other respiratory diseases, in the last year a lot of researchers have analysed different type of audio data in order to understand better the effects of COVID-19 on the respiratory system and develop some tool which automatically carries out a diagnosis for the disease. Besides cough, other respiratory data such as breathing, sneezing and speech have been processed by machine learning algorithms to diagnose COVID-19. Nevertheless, for now the analysis of cough recordings seems to be the one

which better can accomplish the task. Two remarkable examples in this sense are Cough against Covid-19 (Bagad et al., 2020) and AI4COVID19 (Imran et al., 2020).
Usually the work of these research teams starts from the collection of the data, i.e. the recordings of people coughing, using websites and mobile apps created ad hoc. Here the first issues appear, as collecting a consistent number of audio data respecting certain standards of quality can be pretty difficult. This is a relevant issue, since usually the machine learning algorithms need a dataset with a pretty big number of observations to build a model which understands correctly the patterns in the data and is able to achieve good predictive performances. To increment the number of people submitting their recordings, the only possibility is to advertise these projects and raise awareness in the population about the potential which "donating a cough" could have for the purposes of scientific research. For now, one of the largest crowdsourced datasets existing is the one who is being collected for the project Coughvid (Orlandic et al., 2020).
Another issue related to these datasets is that the portion of people positive to COVID-19 which appear in them is pretty small. This has to be taken in account while trying to build a classifier on these data, since the dataset imbalance usually deteriorates the performances of the classification models. Nevertheless, the scientific literature provides a bunch of methods to deal with this situation, such as weighting, bootstrap and oversampling. The latter has already been proved effective in one study about COVID-19 cough (Pahar et al., 2020), in which the Synthetic Minority Oversampling Technique (SMOTE) has been applied in order to balance the dataset. This balancing technique, along with Artificial Neural Networks and Deep Neural Networks, has proved to be pretty effective in distinguishing the subjects with COVID-19 from subjects without COVID-19.
The final aim of the majority of the research groups which are working on the task right now is to build a website or a mobile app able to analyse an inputted audio and return a prediction about the state of the subject. Having such a tool available would increase dramatically the capacity of detecting COVID-19 for all the health systems all over the world, as in order to have a reliable diagnosis for the disease it would be enough to have a smartphone with an app installed or an internet connection. Both things (especially the former) are pretty widespread nowadays also in the developing countries, where otherwise it would be pretty difficult and expensive to get a test, due to the limitations of the health and infrastructural system. A very good result would already be to construct a high-sensitivity classifier to use as preliminary screening in order to select the people who actually could have COVID-19 and test them with the classical medical tools.

1.4 Purpose of this work

In this work I have tried to reproduce part of what has been done in the last year by the scientific community about the detection of COVID-19 through the

analysis of the cough. In particular, I have drawn inspiration from the work "COVID-19 Cough Classification using Machine Learning and Global Smartphone Recordings" (Pahar et al., 2020), in which the Coswara data set (Sharma et al., 2020) has been analysed to build a predictive tool for COVID-19 using several classification models such as Logistic Regression, Support Vector Machine, Multilayer Perceptron, Convolutional Neural Network and Resnet50 classifier. The latter showed the best results, which seem to be very promising in order to use this tool as an approved medical means.
Here only Logistic Regression, Support Vector Machine and Multilayer Perceptron have been used. The raw audio data from the Coswara repository have been treated in a way was thought as a simplified version of the state of the art for audio signal preprocessing of cough. The results obtained are pretty far from the ones of the reference work (Pahar et al., 2020), but examining the procedure can be useful to have and idea of how this kind of analysis is performed.

Chapter 2

The analysed data

2.1 The Coswara dataset

As mentioned before, the collection of useful and consistent data for this type of analysis is not trivial, and it makes pretty difficult the task of the researchers who are trying to build a diagnostic tool for COVID-19 based on cough sound. Concurrently, for who wants to engage in this type of analysis without collecting their own data it can be pretty challenging to find some reliable and publicly available data set.
One of the few which are available online and present data collected in a pretty consistent way is the CoSwara (Co = COVID-19, Swara = "sound" in sanskrit) data set. This data set has been constructed by the Indian Institute of Science of Bengaluru, and is publicly available on the corresponding GitHub repository (https://github.com/iiscleap/Coswara-Data). At the time of writing (May 2021) the repository contains observations from around 2000 subjects, but at the time in which this work has been started (February 2021) it contained 1486 subjects, which are the ones used to train the predictive models. The new subjects inserted afterwards in the study have been used as test set, guaranteeing the blindness of the model in the training phase with respect to the test set. This splitting has been done based on the assumption that the characteristics of the cough sound are not changing in time, which seem to be reasonable under an acoustic and medical point of view, but should be tested more deeply at a later stage in order to give a stronger validation to the results of the work and avoid any downwards bias in the estimation of the predictive capacity of the models.
The collected audio are of 4 types: coughing (hard and shallow), breathing (deep and shallow), counting (normal and fast) and pronouncing vowels ("a", "e" and "o"). In this work, as in the reference paper (Pahar et al., 2020), only the hard coughing has been analysed, but, as has been proven by studies for other respiratory diseases, also the other types of data could have an interesting potential in achieving good diagnosing performances for COVID-19.

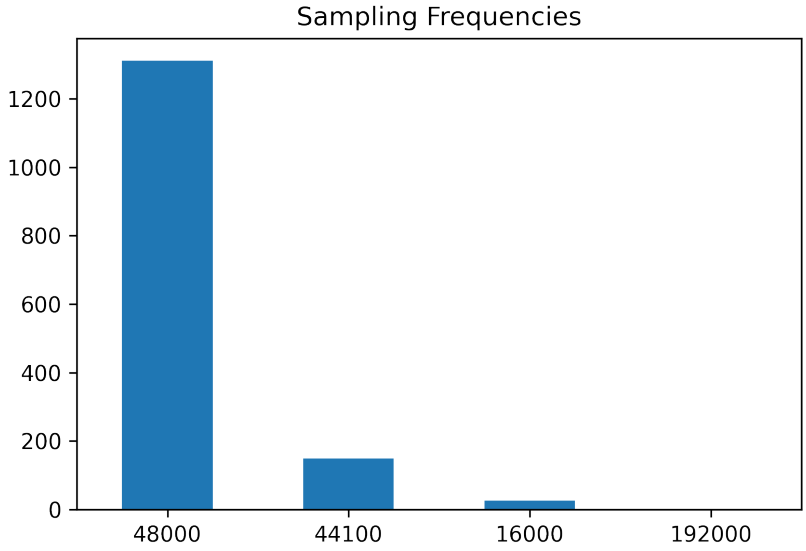


Figure 2.1: The sampling rate of the collected recordings for all the subjects composing the dataset. The majority of them (1311) has uploaded recordings sampled at 48 kHz, while the others have recorded at a sampling rate of 44100 Hz (148 subjects), 16 kHz (25 subjects) and 192 kHz (2 subjects). As described in Section 2.3, the 25 recordings sampled at 16 kHz have been excluded from the study, while the others have been resampled at 44.1 kHz.

The sampling frequency of the majority of the audio is 48 kHz, with some other audio sampled at 44.1 kHz, 16kHz and 192kHz (Figure 2.1). The recordings have been resampled in the stage of data preprocessing in order to homogenize the sampling frequencies for all the data analyzed.

2.2 Exploratory analysis: Dataset composition

For each subject also a bunch of metadata have been collected. This metadata contains general information such as home region and/or country, age and sex, and clinical information such as smoking state, symptoms at the time of recording and pre-existing diseases (e.g. asthma).
Clearly, each subject has declared their current COVID-19 status. Also people currently having some other respiratory disease different from COVID-19 appear in the data repository, but in this study they are grouped with the healthy subjects, composing the "Negative to COVID-19" group.
The Figure 2.2 show some general information about the type of subjects present in the data set. We can see that we have a pretty wide age range, even if the majority of the people is aged between 20 and 55 (Figure 2.2a). We can also see that there is a prevalence of males in the study (Figure 2.2b), and clearly a huge

portion of the people in the dataset are from India. People from all the other continents appear in the dataset, but they are significantly underrepresented compared to Asia (Figure 2.2c). As mentioned before, the subjects positive to COVID-19 are consistently outnumbered by the negative ones: in the dataset considered for training the models the subjects positive to COVID-19 are 107 against the 1379 people negative to COVID-19 (Figure 2.2d, i.e. the positive subjects represent slightly more than the 7% of the total population under study. This proportion changes in the test set, where the people positive to COVID-19 represent the 27% of the whole test (as show in Figure A.1), meaning that in the last months more people positive to COVID-19 have submitted their recording in the repository. Also for what concerns the other aspects such as gender and geographical origin the test set is slightly more balanced (plots in Appendix A).

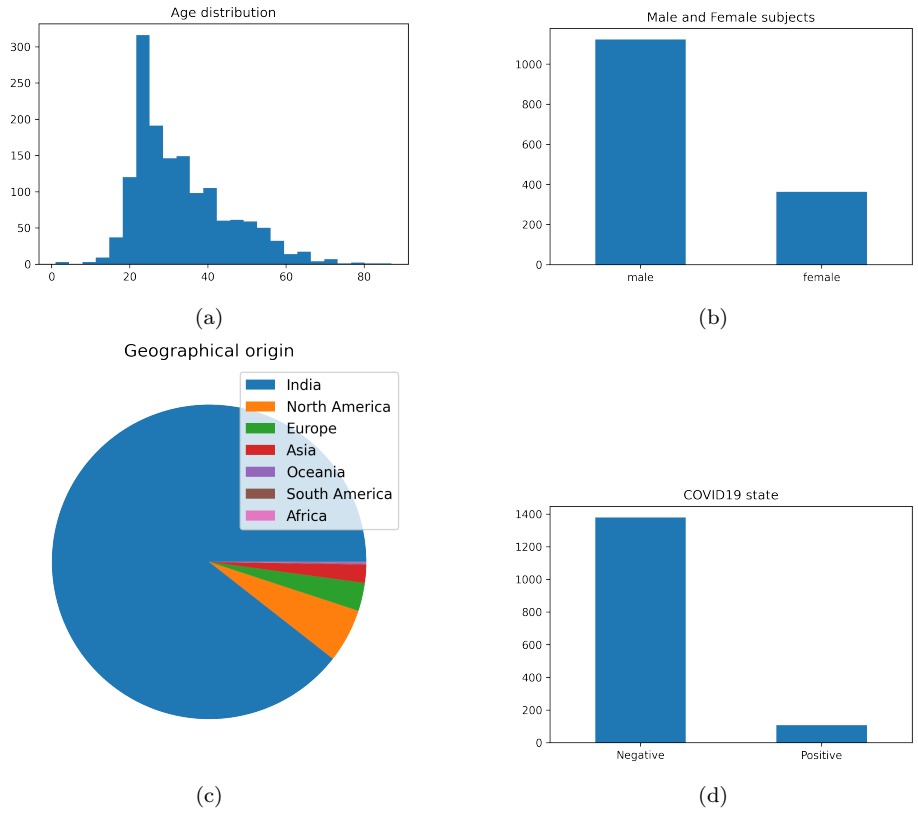


Figure 2.2: Coswara dataset composition at 15/02/2021. The 89% of the subjects is aged between 20 and 55 years, the 76% of them is male, and only the 7.2% is affected by COVID-19. The 89% of the subjects is from India, while apart from Asia only North America and Europe get to compose more than the 1% of the dataset (respectively 5.5% and 2.8%). The other continents are represented just by 1 or 2 people. These numbers undergo some slight changes after the process of selection of the subjects which are actually inserted in the dataset to analyse.

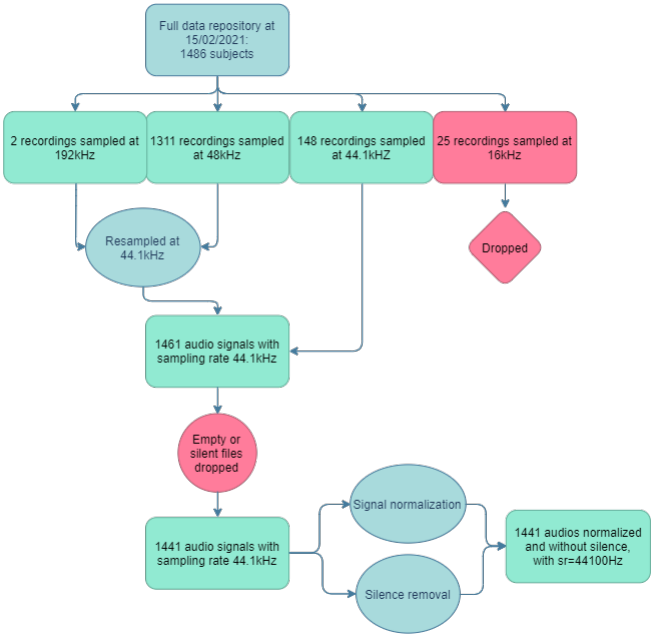
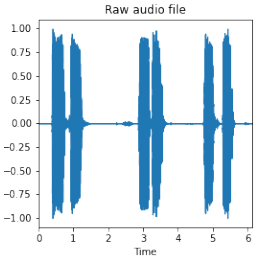


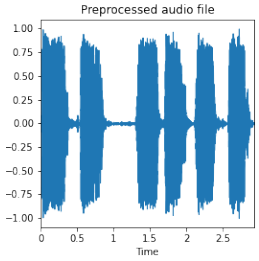
Figure 2.3: The scheme of the data preprocessing phase. Starting with 1486 recordings, after filtering the data according to their sample rate and dropping the empty or silent recordings, we end up with 1441 audio files, which are resampled at 44.1 kHz, normalized and cut to delete the silent intervals

2.3 Data preprocessing

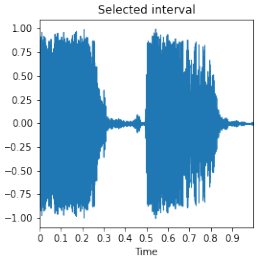
The specific acoustic characteristics of the coughing make the analysis of this kind of data pretty tricky, as only a small part of the recording, namely the intervals in which the person coughs, is actually bringing useful information.
As the first step of the work, the raw audio data (in format .wav) downloaded from the CoSwara GitHub repository, have been extracted, and the audio signals have been normalized. This has been done in order to avoid biases in the analysis due to differences in the loudness of the collected recording, which can occur when the recording is not performed in a controlled way, as the loudness of the obtained signal depends on factors such as the device used to record and the distance kept from it during the recording.
At this stage, the recordings with a sampling frequency larger than 44.1 kHz have been downsampled from their original frequency to 44.1 kHz, while the recordings with a sampling frequency smaller than 44.1 kHz (25 recordings) have been discarded from the dataset. Moreover, some recordings which resulted to be empty or totally silent have been dropped, leaving the dataset with 1441 observations.



(a) In the plot of the original audio (not preprocessed) it is possible to spot 3 pairs of coughs divided by intervals of approximately one second of silence.



(b) The silent intervals have been cut out in the preprocessing phase. The length of the preprocessed audio is more or less half of the original one.



(c) The selected 1-second interval brings contains a pair of coughs (the first pair). The audio wave is also rescaled in order to have a range 0-1 for the absolute values of the samples.

Figure 2.4: Plot of the audio waves regarding the subject with ID 0zexHIcM7tQDdnFiEj2Eb0v3g212 (recorded in April 2020).

Subsequently, the silence has been removed from the remaining audio signals, simply by cutting out all the intervals of 50ms or more presenting energy and zero crossing rate values below a specified threshold (Cohen-McFarlane et al., 2019; Bachu et al., 2010). The Figure 2.4 shows the plot of the audio signal of one recording before and after the preprocessing work. Comparing the frequency distributions of the lengths of original and preprocessed signals, it is possible to notice that during the preprocessing process the length of the recordings has been practically halved (Figure 2.6). It is also important to notice that the range of preprocessed audio duration is pretty wide, and so it is also the range of the number of coughs that each person has done in the submitted recording (it goes from 2 up to more than 7 for a single recording).

At this point, it is necessary to select intervals from each recording which carry useful information and are comparable among them. In the last years a bunch of studies has been conducted in the field of event detection through audio processing applied to medicine, and some of these studies have their main focus on the recognition of cough (Liu et al., 2014; Amoh and Odame, 2015). The field is still in development, but some satisfactory results have already been achieved. Researchers have been able to set up some algorithms which can be used to analyse audio recording from hospital rooms where the patients are under control h24. The algorithms have been proven able to detect the cough of people with their starting and ending instant, and to distinguish the sound of the cough from other types of sound (Barry et al., 2006).
In this work, not disposing of tools of this type, I have tried to mimic the rationale of these algorithms, but in a much more simplified way. The idea is to select from each recording the interval which has more acoustic energy (i.e., the loudest interval). The length of the interval is the same for all the individuals in order to make the data easily comparable. As the duration of a

Figure 2.6: Histograms of the length of original and preprocessed audio files. It is possible to see how the shape of the distribution is pretty similar, apart from an excess of recordings of length 0 in the first plot, which were detected as problematic and dropped from the dataset. The values of the distribution are more or less rescaled by 0.5, and the central 90% of preprocessed recordings has length between 0.96s and 5.14 seconds. This is an indication of the fact that some recordings have significantly more coughs or, in the worst case, more noise than others.

cough is generally between 0.4s and 0.8s, the chosen value for the length of the interval is of 1s, in order to be sure that a complete coughing enters the interval. Some other values have been tried out, but some preliminary analysis assessing the different settings through cross validation proved this one to ensure the best performance to the models. A potential drawback of using this method to select the audio intervals to analyse is that there is no way to check if external noises or multiple coughs are included in the interval. This could affect the analysis since, especially with pretty simple models, the inputted audio features should in some way be synchronized in respect to time in order to be analysed efficiently.

The extraction of the audio waves and their normalization has been performed in Python 3.7 using the librosa package (McFee et al., 2015). The silence removal and the detection of the loudest interval have been coded from scratch, relying on the numpy package.

Chapter 3

Feature extraction

After preprocessing the data, the audio features have been extracted from the recordings. These features carry important information about the audio under analysis and are supposed to show different patterns in the case in which the sounds from which they are extracted are different.
In the last years, with the arrival of the deep learning algorithms, also the "end to end" strategy has become pretty widespread for audio analysis. It consists in feeding to the model the raw or preprocessed audio signal and letting it learn how to extract meaningful features from it. Nevertheless, this kind of strategy requires powerful computing machines and complex deep learning algorithms to work, and in several aspects it is still a bleeding edge for audio processing, as it doesn't give any control to the analyst about the features to extract. For these reasons, the classical strategy has been adopted in this study, and the features have been extracted manually.
To perform the feature extraction, the selected audio intervals have been divided in frames of 2048 samples, with a hop length of 512 (i.e. the first frame goes from the 1st to the 2048th sample, the second one goes from the 513th to the 2560th sample, and so on). Hence, each interval of 1 second has been divided in 87 overlapping frames. From each frame the following features have been extracted: mel frequency cepstral coefficients, along with its first and second derivative, log energies, zero-crossing rate and kurtosis.
These features are widely used in the field of audio signal processing, and in particular when it comes to cough analysis. The first three are specific features of audio signals, while the other three are more basic "statistical" features which also carry import information about the audio signals. In this work the MFCCs, its derivatives and the zero crossing rate are extracted using the Python package librosa, which provides specific functions for this purpose, while for the kurtosis the formula from the package scipy is used, and the log energy is computed by implementing its formula in the code from scratch.

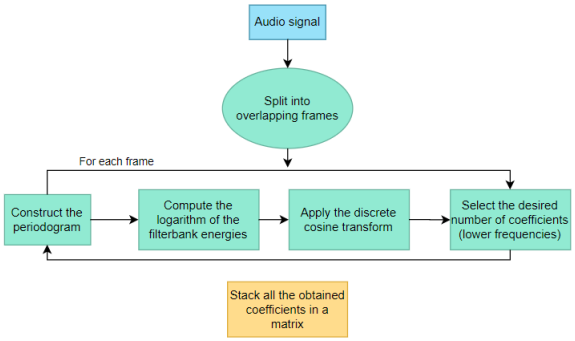


Figure 3.1: The steps for computing the MFCCs from an audio signal. The signal is split into overlapping frames of fixed dimensions and step between them, and on each frame the procedure for the extraction of the MFCCs is performed. Once the coefficients are available, the desired subset of coefficients is kept, and they are all stacked together to form the matrix of mel frequency cepstral coefficients of the audio signal.

3.1 Mel frequency cepstral coefficients, velocity and acceleration

Mel Frequency Cepstral Coefficents (MFCCs) (Davis and Mermelstein, 1980) are a feature widely used in audio signal processing, in particular in the fields of automatic speech and speaker recognition (Han et al., 2006). In the last years they have been also used in some studies about cough analysis, being proved to be able to distinguish dry cough from wet cough (Chatrzarrin et al., 2011; Huang et al., 2001).
The MFCCs are coefficients that try to describe mathematically how the sound is perceived by the human hearing starting from its short time power spectrum. These coefficients provide a description of the shape of the vocal tract, which filters and determines the sound produced by a person when speaking, coughing or breathing. The strength of these features relies in the fact that they carry at the same time information on the time domain and on the frequency domain of the signal.
The MFCCs are computed for each frame of audio signal. These frame have to last enough to bring some useful information, but still need to be short enough so that the series of samples can be considered statistically stationary. In this study each frame lasts approximately 46 milliseconds.

To compute the coefficients for a frame, first the power spectrum is computed by constructing its periodogram, describing how much each sound frequency is present in the frame. Then the periodogram bins are grouped in overlapping bands of frequency and summed up to get an idea of how much energy there is in each frequency region. Usually the number of groups fixed between 26 and

40.
These groups are not equally spaced, since it would not reproduce the way in which the human hearing perceives sounds. In fact, humans are not able to distinguish slightly different frequencies when they are both high as precisely as they are able to distinguish them when they are both low. The human ear is more "precise" at low frequencies.
To space the groups in a way that mimics this structure, the Mel filterbank is used. The filterbanks are equally spaced on the Mel scale, which is a scale of perceived frequency. The formula shown below is used to convert a frequency f to its corresponding to Mel scale value $M(f)$:

$$M(f) = 2595 \times \log_{10}(1 + \frac{f}{700})$$

Hence, in the Mel filterbank the first filters, indicating how much acoustic energy is concentrated at low frequencies, are very narrow, and as the frequencies get higher the filters get wider, giving only a rough idea of how much energy occurs at each spot.
Once obtained the filterbank energies, their logarithm is computed. This is a way of rescaling the magnitude, motivated by the fact that humans don't hear loudness on a linear scale. The final step is to compute the discrete cosine transform of the log filterbank energies. This is done because, as our filterbanks are all overlapping, the filterbank energies are quite correlated with each other and this can be a problem with the majority of the classification models. The DCT decorrelates them avoiding this issue.
Once all of this has been done, the higher order coefficients are discarded, as they are the ones who bring less useful information in the context of sound classification. Usually keeping only the first 13 coefficients is enough to build a good model in the field of speech process, as the higher order coefficients represent fast changes in the filterbank energies, which can even degrade the performance of some models.
Nevertheless, sometimes more coefficients are included in order to try to obtain better performances. In the reference study (Pahar et al., 2020) the number of coefficients has been treated as a hyperparameter to optimize in order to select the optimal number of coefficients for each model. In this work only the first 13 MFCCs have been kept to perform the analysis for the sake of simplicity.
The Figure 3.1 summarizes all the steps performed in order to compute the MFCCs.

3.2 Log Energies

The energy of an audio signal is the mean of the squared wave function. The logarithm of this value has been proven to be a feature which improves the performance of neural networks for the analysis of time series (Samiee et al., 2014). The formula below shows how the log energy is computed for a frame of

N samples in which the signal for the sample at time t is represented as $s(t)$.

$$L = log_{10}(\epsilon + \frac{\sum |s(t)|^2}{N})$$

The letter *epsilon* represents a small number (in this work 10^{-7}) used to avoid having 0 as argument of the logarithm.

3.3 Zero-crossing rate (ZCR)

The zero-crossing rate (ZCR) of a frame is the number of times the signal changes sign within the frame itself. Representing as λ) the indicator function, we can compute the zero-crossing rate for each frame through the following formula.

$$ZCR = \frac{1}{T-1} \sum_{t=1}^{T-1} \lambda(s_t * s_{t-1} < 0)$$

3.4 Kurtosis

In probability theory and statistics, the kurtosis is the fourth standardized moment of a probability distribution of a real-valued random variable. It is a measure of the "tailedness" of the probability distribution itself (DeCarlo, 1997). In the field of audio signal processing, the kurtosis of an audio signal indicates the scarceness of high amplitudes, i.e. a frame with high amplitudes (in absolute value) has low kurtosis.
The generic formula for the kurtosis for a random variable X with mean μ and 4-th moment σ^4 is the following.

$$\Lambda_x = \frac{E[(x_i(k) - \mu)^4]}{\sigma^4}$$

3.5 Merging features

After extracting and standardizing all the features, the frames are grouped and the features of frames composing the same group are averaged. In this way the dimension of the feature space is reduced, which can be helpful to improve the performances of some models. Different options for the number of frames composing a group have been tried out during some preliminary analysis, and in the end it has been decided to group 5 frames at a time. Differently from the frames themselves, the groups are not overlapping.
After performing the merging of features we end up with a features space of 756 dimensions (42 features $\times$ 18 groups of frames) for our data.

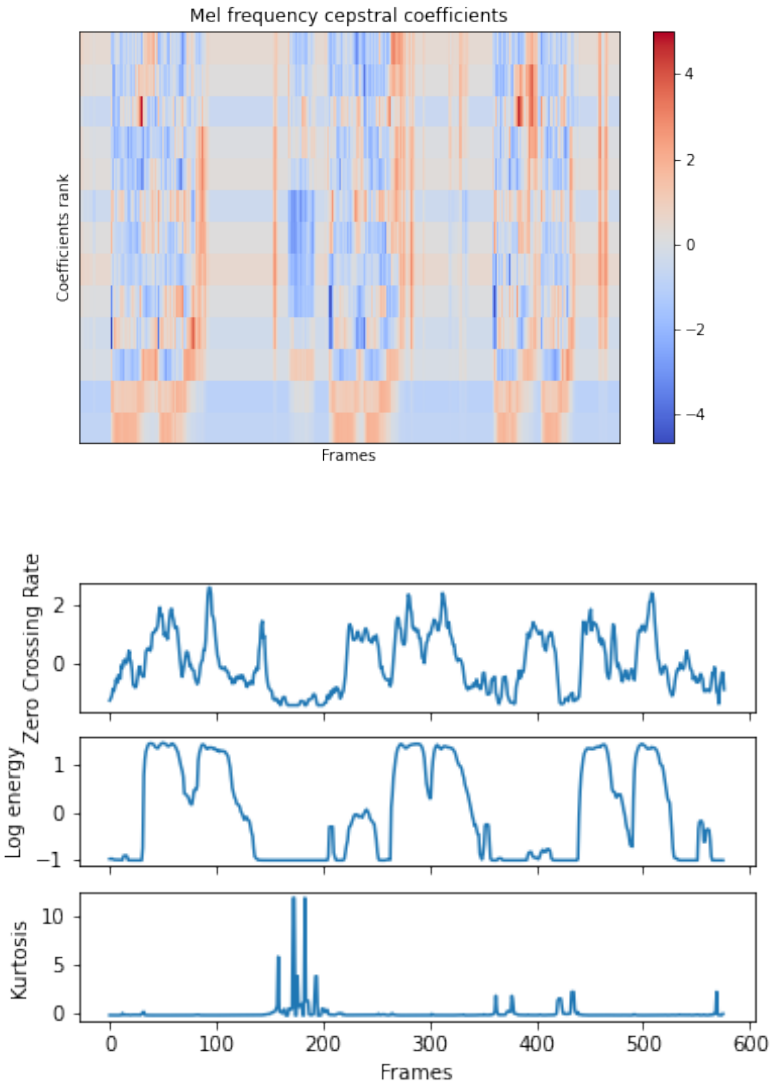


Figure 3.2: Plots of the features extracted from the raw audio of the subject with ID 0zexHIcM7tQDdnFiEj2Eb0v3g212.
Comparing the plots among them and with the plots in 2.4, it is possible to notice how the ZCR and the log energy, as expected, increase when the person coughs. The kurtosis has the opposite behaviour, showing some peaks in correspondence of the silent intervals. At the same time, also the MFCCs show a quite clear pattern based on the alternance of cough and silence. When the person coughs, also the values of the MFCCs change, with the coefficients of lower grade increasing their value.

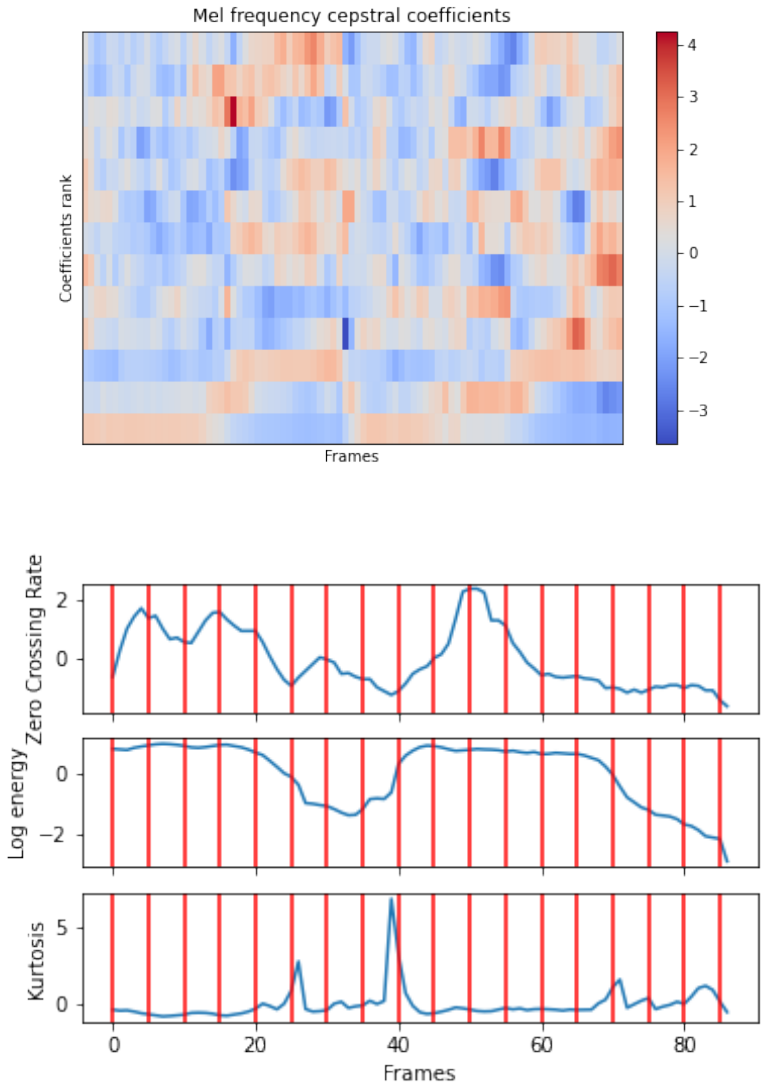


Figure 3.3: Plots of the features regarding the interval of 1 second selected from the audio of the subject with ID 0zexHIcM7tQDdnFiEj2Eb0v3g212.
Here the values of the various features are more constant through the different frames, as there is almost no silence in the whole interval. Still, it can be noticed pretty clearly how the features change throughout the different phases of the cough: from the initial "explosive" phase (high ZCR and log energy, low kurtosis, MFCCs dominated by the low frequency) to the final "bedding" phase (low ZCR and log energy, how kurtosis, MFCCs dominated by the high frequency).
The red vertical lines in the plots show how the 87 frames are grouped in the phase of features merging.

Chapter 4

Methods

As mentioned before, three different algorithms of supervised learning have been used to analyse the data and build models classifying the subjects as affected or non-affected by COVID-19. Before constructing the predictive models, the observations of the class "Positive" have been oversampled in order to balance the dataset.

4.1 Synthetic Minority Over-sampling Technique (SMOTE)

As already mentioned, the dataset under study suffers from a very strong unbalancing. This can be a big issue for machine learning techniques like the ones which have been used in this work, as the huge difference in cardinality between the two classes can induce the model to systematically misclassify subjects from the underrepresented class.
To compensate for this imbalance, Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al., 2002) has been applied. This technique, which has already been used successfully in the past for other tasks regarding cough detection and classification (Windmon et al., 2018), consists in generating synthetic samples for the minor class (or classes), based on a k-neighbours scheme.
The algorithm works in the following way: observations of the class to upsample are selected in a random order, and for each of them one of the k nearest neighbours is randomly chosen. The synthetic observation is generated constructing a new point in the features space lying in the straight line connecting the two starting points (both existing in the original dataset). The position on this line is in turn chosen randomly by drawing a number u from a Uniform distribution between 0 and 1. The formula below describes the generation of one synthetic observation, x^{SMOTE}, starting from a point x and one of its k-nearest neighbours, x^{NN} .

$$x^{SMOTE} = x + u \times (x^{NN} - x), \quad u \sim Unif(0, 1)$$

In this work the number of neighbours is fixed as 5, and the upsampling rate is above 1200 percent, meaning that each subject positive to COVID-19 has been used as base observation for the generation of synthetic samples at least 12 times. Only a subset of this group has been used for the 13th round of upsampling.
When upsampling an imbalanced dataset with SMOTE, it is crucial to do that only inside the training set after having performed the training/validation or train/test split. In this way synthetic observations generated from observations in the training set don't belong to the validation or test set (and vice versa), as this would cause an overoptimism bias Santos et al., 2018, consisting in a systematic overestimation of the performances of the model. For this reason, when the cross-validation scheme has been used to tune the hyperparameters, the SMOTE has been applied for each fold on specific the training subset of that specific fold, leaving the validation set (used to measure the performances of the model) out of the oversampling process (Figure 5.2). In the final stage of the work, when the final models have been trained on the whole training set, the SMOTE has been applied on all of it, but always keeping the test set blind with respect to this procedure (as to all the training process).

4.2 Logistic regression

The logistic regression (LR) is a statistical model belonging to the family of the generalized linear models. It is used to estimate statistical relations when the response variable is dichotomous (i.e. for binary classification problems). This model has been successfully used in the past in some clinical prediction tasks, and in this study it is used as a baseline against which we measure any improvements provided by more complex classification methods, as in some tasks in the past they have not been able to outperform the logistic regression (Christodoulou et al., 2019).
The most common version of this model, used in this work, requires the use of a logit link function, consisting in assuming the following relation between the predictors and the expected value of the response variable.

$$ln(\frac{P_X}{1-P_X}) = \alpha + X\beta$$

Considering the two possible outcomes of the experiment as baseline and case, the value P_X can be interpreted as the probability of belonging to the "case" class for a subject with observed features X. In this work, the baseline class is "Negative" and the case class is "Positive". Inverting the function, it is easy to notice that this probability is obtained by applying the so-called sigmoid function to the linear combination of the features, as shown in the following formula.

$$P_X = \frac{1}{1+e^{-(\alpha+X\beta)}}$$

At this point the response values are predicted as case if the estimated probability is larger than a specified threshold and as control otherwise. Generally

the threshold value is 0.5, but often using other thresholds can be helpful to improving the predictive performances of the model.
In this work, it has not been fitted the basic logistic regression. Two special versions of this model have been applied instead, called lasso (Tibshirani, 1996) and ridge regression (Le Cessie and Van Houwelingen, 1992). These methods, belonging to the family of regularized regression or penalized regression, consist in putting a penalty on, respectively, the absolute value and the squares of the coefficients to be estimated. In practice, the default loss function to optimize

$$L_{unreg}(\hat{\alpha}, \hat{\beta}) = \sum_{i=1} n \log(exp(-y_i(\alpha + X\beta)) + 1)$$

is changed in the following for the lasso regression (l_1 penalty):

$$L_{l1}(\hat{\alpha}, \hat{\beta}) = ||\beta||_1 + C \sum_{i=1} n \log(exp(-y_i(\alpha + X\beta)) + 1)$$

and in the following for the ridge (l_2 penalty):

$$L_{l2}(\hat{\alpha}, \hat{\beta}) = \frac{1}{2}\beta^T\beta + C \sum_{i=1} n \log(exp(-y_i(\alpha + X\beta)) + 1)$$

From the formula it is clear that as the value of C decreases the first term (the penalty) gets more importance in the computation of the loss function. As this happens, the model is forced to have null or very small coefficients for the predictors which don't have a big impact on the variability of the response variable. This can be seen as an automatic way to perform features selection, which can be very important to avoid overfitting when dealing with a dataset with a number of features fairly large if compared to the number of observations. Moreover, performing regularization improves the numerical stability of the model.
The magnitude of the regularization parameters C has been included in the models as a hyperparameter, and tuned through cross validation in the first stage of the analysis (Section 5.1).

4.3 Support Vector Machine

The support vector machine (SVM) is another algorithm which can be used for binary classification. It consists in dividing the features space in two parts with a hyperplane separating the subjects belonging to different groups: each new observed subject lying at one side of the hyperplane will be then classified as belonging to one class, while the subjects on the other side will be classified as belonging to the other class. A good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.
In order to improve the efficiency of the algorithm, the original features are

transformed through a function ϕ, and the hyperplane is used to split the space of transformed features. The scalar product between two transformed points $K(x, y) = \phi(x)^T \phi(y)$ is said kernel, and is what actually needs to be fixed in order to define the support vector machine (while it is not necessary to define explicitly the corresponding function ϕ).
Moreover, as it is practically impossible to find two groups which are perfectly linearly separable, the support vector machine usually involves the use of soft margins, a regularization technique which allows the model to mispredict some observations during the fitting procedure, or correctly predict them but letting them lie between the margins. This is a way to avoid overfitting and ensure a better capacity of generalization to the model, and it is particularly useful when the number of observations is outnumbered by the number of features, or when the observations are pretty noisy. The loss function optimized by the SVM algorithm with soft margins is:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^t w + C \sum_{i=1}^{N} \xi_i \\ \text{s.t.} \quad & y_i(w\phi(x_i) + b) + \xi_i - 1 \geq 0 \\ & \xi_i \geq 0, \qquad i = 1, ..., n \end{aligned}$$

The function ϕ is the transformation function, $sign(\phi(x_i) + b)$ is the prediction for x_i and y_i is its true value (true class, labelled with +1 or -1).
Intuitively, we're trying to maximize the margin (by minimizing $w^t w$), while incurring a penalty ξ_i when a sample is misclassified or within the margin boundary. The penalty term C controls the strength of this penalty, acting as an inverse regularization parameter: the smaller the parameter, the stronger the regularization (i.e. the algorithm allows more easily for misclassification).
In this work, the value of this parameter has been tuned through cross validation, as it has been done for the logistic regression model, and the algorithm has been used with linear, cubic and radial kernels. The formulas of the kernels are the following:

$$\begin{aligned} K_{linear}(x, y) &= x^T y \\ K_{cubic}(x, y) &= [\gamma(x^T y) + r]^3 \\ K_{radial}(x, y) &= exp(-\gamma ||x - y||^2) \end{aligned}$$

Here the intercept r for the cubic kernel has been fixed as 0, while the value of γ in the cubic and the radial kernels has been fixed as

$$\gamma = \frac{1}{p * var(X_{train})}$$

where p is the number of columns of X ($p = 756$) and $var(X_{train})$ depends on the specific subset of the training set on which the model is being fitted, and on how this subset has been oversampled.
In the literature there are already different studies available in which the SVM has been used as a binary or multiclass classifier to detect and classify cough events (Tracey et al., 2011; Bhateja et al., 2019).

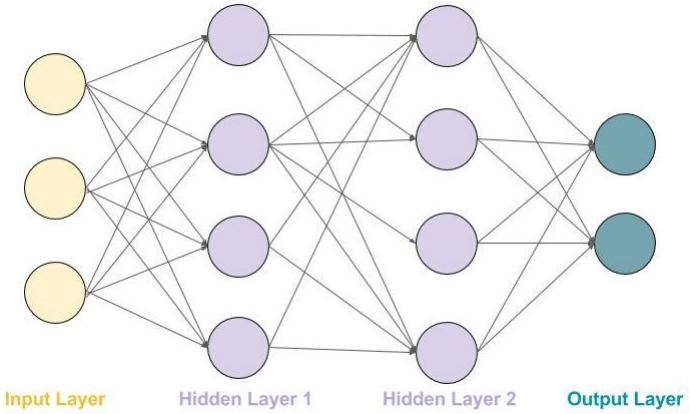


Figure 4.1: The scheme of a fully connected multilayer perceptron neural network. The one represented has two hidden layers and more than one neuron in the output layer. In this work the output layer has only one neuron, while the number and the dimensions of the hidden layers are treated as a hyperparameter. The dimensions of the input layer depend on the duration of the analysed audio and on the length of a single frame.
The figure has been taken from https://learnopencv.com/image-classification-using-feedforward-neural-network-in-keras/

4.4 Multilayer Perceptron

A multilayer perceptron (MLP) is a neural network with input layer and output layer connected by multiple hidden layers of neurons. These models can be used for classification and regression tasks, having much more flexibility than the classical statistical models in learning nonlinear patterns. The price to pay for this flexibility is an increased number of parameters to optimize, and a way reduced interpretability of the parameters themselves (Olmedo et al., 2018).
In the past these models have already been used with good results for cough detection and to discriminate influenza cough from other coughs (Sarangi et al., 2016).
The structure of the multilayer perceptron is the one observed in the figure 4.1. Each layer has a number of neurons which are connected with all the neurons of the previous and next layer. Given the vector of values x observed in a layer, the value of one neuron of the next layer is computed as the linear combination of these values, to which an activation function is applied, as the formula below shows.

$$y = \phi(w^T x + b)$$

The weights w and b are optimized in the supervised training phase, while the activation function ϕ is chosen a priori.
In this study the rectified liner unit (ReLu) has been used as activation function for the hidden layers, while the sigmoid function is used for the output

layer. Hence, the output of the neural network is a value between 0 and 1 for each observation, which can be interpreted exactly as the output of the logistic regression. In fact, it is interesting to notice that the logistic regression is an artificial neural network without hidden layers, in which the information directly passes from the input layer to the output neuron through a linear combination and a sigmoid transformation.
As done for the linear regression, also for this model a l_2 regularization procedure has been used for each neuron, together with the dropout technique. The dropout (Srivastava et al., 2014) is a method which consists in randomly inactivating, or dropping, some neurons for one or more layer at each epoch of the training process, in order to force the fitted model not to give too much importance to a small set of neurons, which could be a source of overfitting. This technique approximates training many neural networks with different architectures in parallel.
The different candidate values for dropout rate, regularization strength, learning rate and batch size are specified in the Table 5.2. The best combination of all these hyperparameters has been found through cross-validation during the hyperparameters tuning phase (Section 5.1). The number of layers and the number of neurons in each layer have been fixed before this stage, based on the results of some preliminary analysis. The model used contains 10 hidden layers, with the number of neurons exponentially decreasing from 1024 (2^{10}) to 2 (2^1).

Chapter 5

Classification process

5.1 Hyperparameters tuning

As mentioned before, all the stages of this work (data preprocessing, features extraction, classifiers fitting) present some hyperparameters to be optimized. For the hyperparameters regarding data preprocessing and features extraction (i.e. frame dimensions, hop length, number of MFCCs, length of the intervals and number of frames to merge in each group), it has not been carried out a real hyperparameter tuning procedure, for sake of computational time. What as been done is to perform some preliminary analysis with the logistic regression model, using the cross validation or a bunch of different simple train/validation splits. These analyses, together with some knowledge drawn from the theory of audio signal processing and from previous studies on this dataset (Pahar et al., 2020), are used to fix the values of the non-tuned hyperparameters. These values are summarized in the Table 5.1 below.

Hyperparameters	Stage	Fixed value
Samples per frame	Data preprocessing	2048
Hop length	Data preprocessing	512
Length of the intervals	Data preprocessing	1 second
Number of MFCCs	Features extraction	26
Number of frames to merge	Features extraction	4

Table 5.1: Hyperparameters related to data preprocessing and features extraction. These hyperparameters have not been included in the systematic tuning process carried out through 5-fold cross validation, but they have been fixed relying on some well known result regarding audio signal processing and some preliminary analysis conducted using the logistic regression model.

On the other hand, the hyperparameters regarding the classifiers have been tuned using the 5-fold cross validation scheme shown in the Figure 5.1. The training set (N = 1441) has been divided in 5 folds of dimension (n_f = 288

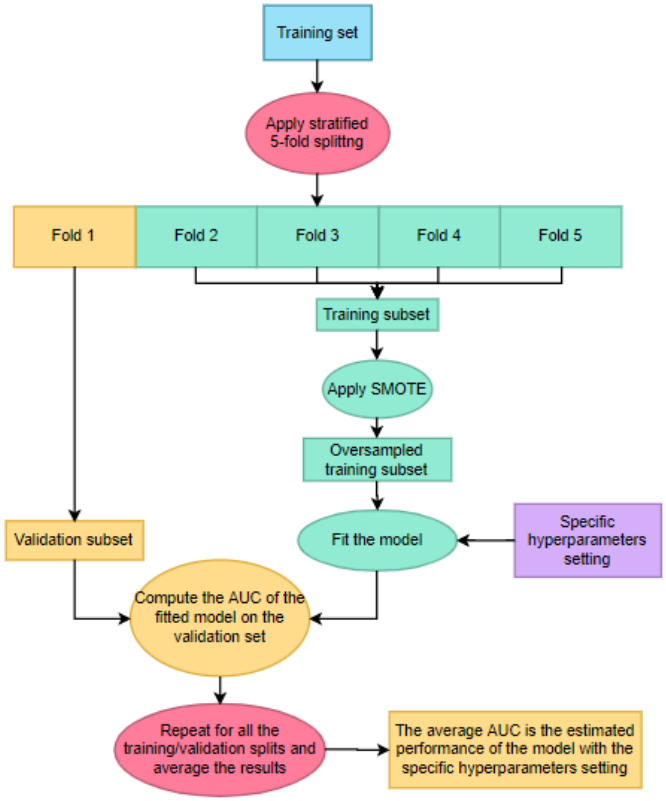


Figure 5.1: The process used to estimate through cross validation the performance of a model with a given setting of hyperparameters. The SMOTE is applied on the training subset after each splitting, and the procedure is repeated 5 times. Each estimated model is evaluated computing the area under ROC curve on the validation subset, and the evaluation of a specific setting of hyperparameters is obtained by averaging the 5 results. This procedure is repeated for each combination of hyperparameters, and at the end the combination providing the best performance is kept to fit the model on the whole dataset.

for $f = 1, 2, 3, 4$ and $n_f = 289$ for $f = 5$). The process has been stratified with respect to the response variable, in order to have a constant percentage of subjects positive to COVID-19 across the 5 folds. Subsequently, one fold at a time has been taken out from the training set and the models have been trained on the remaining observations after oversampling them using SMOTE. This has been done with all the possible combinations of hyperparameters. Table 5.2 provides an overview of the search sets using for each of them.
Hence, each model has been fitted 5 times on 5 different (overlapping) training sets, and evaluated on 5 different validation sets (non overlapping among them). The combination of hyperparameters chosen to fit the final model on the whole training set is the one which has performed better across the 5 train/validation splits. As specified before, for each training/validation split the SMOTE has been applied on the training subset as first step, before fitting the model.

Hyperparameters	Classifier	Range
l_1 regularization penalty (C_1)	LR	10^{-7} to 10^{7} in steps of 10^i
l_2 regularization penalty (C_2)	LR	10^{-7} to 10^{7} in steps of 10^i
Soft margin reg. penalty (C_{SVM})	SVM	10^{-7} to 10^{7} in steps of 10^i
Kernel function	SVM	Linear, cubic, radial
l_2 regularization penalty (C_{MLP})	MLP	10^{-7} to 10^{-3} in steps of 10^i
Learning rate	MLP	10^{-5} to 10^{-3} in steps of 10^i
Dropout rate	MLP	0.15 to 0.3 in steps of 0.05
Batch size	MLP	2^k with $k = 6, 7, 8$

Table 5.2: Search set for the hyperparameters of the three classifiers, optimized using 5-fold cross validation as explained in Figure 5.1.

5.2 Area under ROC curve (AUC)

The receiver operating characteristic (ROC) curve is a graph showing the performance of a classification model at all classification thresholds. In particular, this curve plots the true positive rate (TPR) or recall against the false positive rate (FPR) for all the possible classification thresholds. This is a more general way of assessing a binary classifier than simply measuring its accuracy, and it has the strength of suggesting to the researcher the threshold which gives the best trade off between specificity and sensitivity. A common choice for the threshold is the one for which the Equal Error Rate is obtained, i.e. the false positive rate and false negative rate are equal. Clearly, the choice of the threshold is arbitrary, and having the ROC curve can be helpful to understand how a certain change in the threshold would affect the specificity and sensitivity of the model.
Moreover, computing the area under the curve of a model it is possible to obtain a very useful metric to evaluate a binary classifier. In this way the performance of the model with all the possible thresholds is measured at the same time. This metric has been used in this work to measure the performances of the models

both in the hyperparameters tuning phase and in the analysis on the test set. In the last stage of the analysis, the threshold corresponding to the Equal Error Rate and another arbitrary threshold based on the ROC curve have been computed for each model.

Chapter 6

Results

All analysis has been performed using Python 3.7 and Jupyter Notebook. It has been carried out on a laptop Asus VivoBook S, with a processor Intel Core i7-8550U @ 1.80GHz 1.99GHz, 8 GB of RAM and operating system at 64 bit.

6.1 Training and validation

In this phase the models are fitted with all the combination of hyperparameters listed in the Table 5.2, following the scheme described in the Figure 5.1.

6.1.1 Logistic regression

The logistic regression has been carried out using the function LogisticRegression of the Python package scikit-learn. As explained before, two different types of logistic regression are performed, using two different regularization techniques, i.e. Lasso and Ridge regression. Internally, the function uses two different solvers (saga and lbfgs, respectively) in order to solve the two different problems. The parameter C is used to determine the strength of the regularization, following the formula mentioned in the section about the methods.

	1e-7	1e-6	1e-5	1e-4	1e-3	1e-2	1e-1	1e0	1e1	1e2	1e3	1e4	1e5	1e6	1e7
l_1	0.50	0.50	0.50	0.50	0.50	0.59	0.54	0.51	0.50	0.50	0.50	0.50	0.5	0.50	0.50
l_2	0.57	0.56	0.55	0.58	0.57	0.54	0.52	0.53	0.53	0.52	0.52	0.52	0.5	0.49	0.49

Table 6.1: Results obtained by cross validation with the logistic regression using 14 different parameters setting. In general, the model works better with a l2 penalty than with a l1 penalty, even if the best result has been obtained with a l1 penalty and regularization parameter $C = 0.01$, for which the score of the model was $AUC = 0.59$. The best result for the model with l2 penalty has been obtained with $C = 0.0001$ ($AUC = 0.58$).

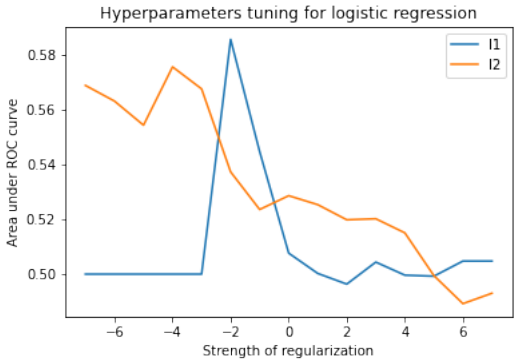


Figure 6.1: Results obtained by cross validation with l1 and l2 penalized logistic regression and different values of the regularization strength. It is possible to see very clearly that the l2 penalized regression benefits from a strong regularization, as its performances get worse as the parameter C increases. The l1 penalized regression obtains the best results with a medium-strong regularization, i.e. with $C = 0.001$ and 0.01. This is due to the fact that a too strong regularization with l1 penalty can cause underfitting. It could be interesting to combine the two regularization techniques using the elastic net with $C < 1$ and see if it can bring some improvement to the performances of the model.

The two models with different values of the regularization parameters C are assessed using the cross validation schema explained in the previous section, and the results are displayed in the Table 6.1.
As we can see from the table, the best performance has been obtained by the model using the ridge regularization scheme with the parameter $C_2 = 0.001$. For what concerns the models fitted using lasso regularization, the best one turned out to be the one with $C_1 = 0.01$, which is still performing worse than the majority of the models fitted with the ridge regression.
From the Figure 6.1 we can see better how the two regularization schemes perform as their regularization strength changes. What is clear is that for $C_1 < 0.01$, i.e. very strong regularization, the model is not able at all to correctly classify the subjects in the validation sets (AUC=0.5 corresponds to the performances of a classifier which randomly guesses the state of a subject). The performance slightly improves with $C_1 \geq 0.01$: as already said, $C_1 = 0.01$ gives the best results, but also the other values bring very similar results.
For what concerns the models fitted with l2 penalty, the behaviour is pretty different: the best performances are obtained with the settings determining stronger regularization, and the value of the AUC decreases as the regularization is weakened.
This behaviour is probably related to the fact that a strong regularization on the absolute value of the parameters generally forces the model to completely ignore a lot of predictors, maybe too many in this case for our purpose. On the other hand, in the case of a l2 penalty, the model is usually more "flexible" in

this sense and tends to include more predictors, even though it still forces a lot of them to have a very low corresponding coefficient.

6.1.2 Support Vector Machine

The support vector machine has been built using the formula SVC of the Python package svm, specific for support vector machines. The package provides the possibility to fit the model with different kernels, while fixing also the strength of the regularization setting the parameter C (as explained in the section about methods). The different combinations tried for the parameter C and the kernel function are listed in the table 5.2.
The Table 6.2 summarizes the results for the Support Vector Machine. From the table it is clear that this model was not able at all to learn the structure of the data analysed, or at least it was not able to build a prediction model able to generalize its activity and correctly predict subjects not seen during the phase of training.

	1e-7	1e-6	1e-5	1e-4	1e-3	1e-2	1e-1	1e0	1e1	1e2	1e3	1e4	1e5	1e6	1e7
linear	0.503	0.503	0.503	0.503	0.503	0.503	0.498	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
poly	0.503	0.503	0.503	0.503	0.503	0.503	0.498	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
rbf	0.503	0.503	0.503	0.503	0.503	0.503	0.498	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Table 6.2: The performances of the SVM on the classifying problem with 45 different combinations of kernel functions and regularization strength. With none of these combinations the model has been able to achieve performances better than a "random guess" (AUC=0.5) on the validation sets.

The analysis of the performances of the support vector machine on the training subsets has shown how the model was suffering from a very strong overfitting, as the AUC value for the train subsets was always consistently higher than 0.5. This means that the model was learning patterns from the training data which were not generalizable to the validation ones.
Due to the bad performance shown in this phase, this model has not been trained on the whole dataset to measure its performances on the test set.

6.1.3 Multilayer Perceptron

The multilayer perceptron has been built using the functions provided by the Python library TensorFlow through its interface Keras, which is one of the most used for the construction and fitting of neural networks.
It has been constructed a neural network with 10 hidden layers of dimensions 2^i with i ranging from 10 (the first hidden layer had 1024 neurons) to 1 (the last hidden layer had 2 neurons). The number of neurons of the input layer is equal to the cardinality of the feature space (i.e. 756), while the output layer has one neuron, as this is a neural network constructed for the binary classification.

The activation function chosen for the hidden layers is the rectified linear unit function, while for the output layer the sigmoid function has been set, as it is usually done in neural networks for binary classification.
In this phase the early stopping method is used, with patience of 20 epochs. This means that the neural network stops training if for 20 epochs in a row the performances on the validation set are not improving. At that point the weights that allowed to get the best performances are kept as fitted weights of the network, and the AUC of that model on the validation set is used as estimate for the performances of the model itself.

C	LR	DR	BS	Avg AUC	AVG Ep
1.000000e-03	0.00010	0.25	128	0.653997	32.8
1.000000e-06	0.00010	0.15	256	0.644895	38.4
1.000000e-06	0.00001	0.30	128	0.642774	39.2
1.000000e-03	0.00100	0.15	128	0.641747	26.8
1.000000e-03	0.00100	0.20	64	0.638234	26.2
1.000000e-06	0.00010	0.25	64	0.638218	30.4
1.000000e-03	0.00010	0.30	256	0.637396	45.0
1.000000e-03	0.00010	0.20	128	0.637343	45.0
1.000000e-06	0.00100	0.20	128	0.636996	31.8
1.000000e-04	0.00001	0.15	256	0.636870	78.4
1.000000e-07	0.00100	0.30	64	0.635344	31.6
1.000000e-07	0.00010	0.20	64	0.633503	32.2
1.000000e-03	0.00100	0.30	128	0.633076	36.0
1.000000e-04	0.00010	0.20	256	0.632655	29.6
1.000000e-04	0.00001	0.15	64	0.632414	32.8
1.000000e-03	0.00010	0.15	64	0.631706	24.0
1.000000e-06	0.00010	0.25	128	0.630149	27.0
1.000000e-03	0.00001	0.15	64	0.630104	41.4
1.000000e-07	0.00010	0.30	64	0.630095	40.6
1.000000e-04	0.00001	0.20	64	0.630071	58.8

Table 6.3: The 20 best combinations of hyperparameters of the multilayer perceptron in terms of performances on the test set. The best combination is: regularization parameter $C = 0.0001$, learning rate $LR = 0.0001$, dropout rate $DR = 0.25$, batch size $BS = 128$. All the first 20 combinations exceed the value of 0.63 for the AUC, which is far from the desirable results but already shows an improvement with respect to the results obtained with the Logistic Regression. It is interesting to see how, as expected, the average number of epochs before the stop of the algorithm is linked with the fixed learning rate.

It has been used Adam as a solver, setting its learning rate as a hyperparameter. As shown in the table 5.2, also the strength of the regularization parameter l_2, the dropout rate and the batch size are used as hyperparameters. The whole 5-fold cross validation procedure is designed to check 180 different combinations of hyperparameters.

The table 6.3 shows the evaluation of the performances of the multilayer perceptron for the 20 best combinations of hyperparameters. From this table we can see that this model seems to be able to improve the performances of the Logistic Regression, since all the first 20 combination exceed the AUC value of 0.63, outperforming consistently the results obtained with the logistic regression.
The best combination of hyperparameters for this model on these data turned out to be the following: regularization parameter $C = 0.0001$, learning rate $LR = 0.0001$, dropout rate $DR = 0.25$, batch size $BS = 128$. This combination returns an average AUC on the 5 folds of 0.65, while the average number of epochs is equal to 33, which means that the best weights for the model were found after 13 epochs.

6.2 Retraining and testing on new data

At this point, the logistic regression and the multilayer perceptron have been trained again on the whole dataset using the combinations of hyperparameters identified as the best by the tuning through cross validation.
The support vector machine has been excluded by this phase, since it returned very poor results during the hyperparameters tuning phase, showing that no one of the hyperparameters' combination tried out was actually useful to build a model able to learn a pattern from the training subset which could be generalizable on the unseen validation subset.
The models built on the whole training set showed performances slightly worse than the ones obtained during the hyperparameters tuning phase. The analysis of these results can help in understanding better the characteristics of the two models and how they are built and assessed.

6.2.1 Logistic regression

The model has been fitted on the whole training set using the l_1 penalty with regularization parameter $C = 0.01$. The obtained classifier has been used to predict the class of the subjects in the test set.
Since the training set has been oversampled with SMOTE before fitting the model, the whole procedure has a random nature. So the training and assessing schema has been repeated 10 times setting 10 different seeds on the SMOTE procedure in Python in order to understand how much the model depends on this randomness, and to assess the model in a way which would not depend too much on the oversampling performed in a single case. The results are shown in the Table 6.4, from where we can see that the performances of the model do not depend that much on the random nature of the SMOTE procedure. On the other hand, the performances seem to be consistently worse than the ones obtained by the same model on the training set with 5-fold cross validation: the average AUC through the 10 runs is $AUC = 0.51$, which is only slightly better than a random guess.

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Avg	Std
AUC	0.513	0.51	0.514	0.521	0.505	0.511	0.502	0.524	0.52	0.513	0.513	0.0065

Table 6.4: The values of the AUC for the final logistic regression model with 10 different seeds for the SMOTE function. The table displays also the average and the standard deviation of the 10 AUC value.

It can be useful to focus on one single "fitting and assessing" procedure and to analyse it in details. To do so, the run which performed better (the 7th run) is considered.
This classifier shows an area under the ROC curve of $AUC_{test} = 0.524$ on the test set, while re-classifying with it the subjects of the training brings an $AUC_{training} = 0.702$. This means that the models suffers from overfitting, which is a problem that had already been noticed during the hyperparameters tuning phase.
The False Positive Rate and the False Negative Rate with different thresholds are considered in order to build the curve the ROC curve of the model, which is shown in Figure 6.4, and to find the best threshold γ that one would use to optimize the performance of this particular classifier. Two different methods are used: the Equal Error Rate and the "largest distance from the random classifier line".

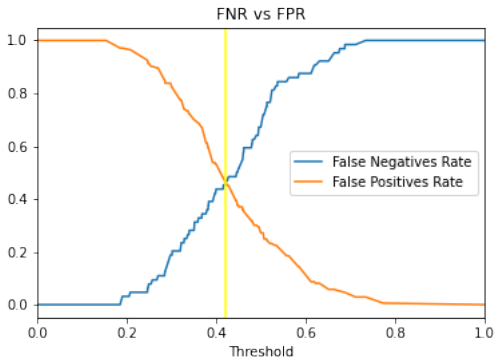


Figure 6.2: The values of FPR and FNR for one fitted logistic regression. The plot shows the typical "X" shape, with EER represented by the value of the two functions where the two curves cross. The lowest estimated probability in this case is around 0.2, while the highest is around 0.8, and this is reflected in the plot, from which we can see that thresholds below 0.2 and above 0.8 would just predict all the subjects as Positive or as Negative, respectively.

The threshold at Equal Error Rate, as explained in the section 5.2, is the threshold for which the FNR and the FPR are equal. For this particular problem it is equal to $\gamma_{EER} = 0.42$, which brings a False Positive Rate of $TPR = 0.465$ and, by definition, the same value of the False Negative Rate. This means that

with this classifier and this threshold, each subject (positive or negative) of the test set would have a probability of being correctly classified of about 53.5%.
The other method consists in finding the point on the ROC curve which has the largest distance from the ROC curve of a random classifier ($AUC = 0.5$), represented in the Figure 6.4 at the end of the section as an orange dashed line. In some sense, this is the point which guarantees the best possible improving with respect to the performances of a "coin toss" classification method. The best threshold according to this method is $\gamma_{DRC} = 0.45$, which brings a True Positive Rate of $TPR = 0.500$ and a True Negative Rate of $TNR = 0.625$. This means that using this threshold, a subject with COVID-19 has the 50% of probability of being correctly diagnosed of the disease, while a healthy subject has the 37.5% of probability of being incorrectly classified as affected by COVID-19. Basically, choosing γ_{DRC} over γ_{EER} would mean to trade the slight improvement in terms of sensitivity (capacity of correctly identify unhealthy subjects) that γ_{EER} ensures with respect to a random classifier, obtaining in exchange a bigger improvement in terms of specificity (capacity of not misclassifying healthy subjects). Knowing this, one can choose the threshold who prefers and that fits better their objectives while using this model (which can also be different from the two proposed here).

6.2.2 Multilayer perceptron

The model has been refitted with the hyperparameters found to be optimal in Section 6.1.3. In this model the sources of randomness are numerous and are intrinsic to the structure of the model itself. Aside from the randomness brought by the SMOTE, the model used in this work presents randomness due to the random nature of the initialization of the values for the weights and to the use of the dropout regularization technique.
As done for logistic regression, the multilayer perceptron has been fitted 10 times with different seeds in order to estimate its performances more precisely, and simultaneously estimate the variability of the performances, which are due to the random components of the model. The results are shown in the Table 6.5, from which we can see that on the test set the model performed way worse than during the hyperparameters tuning phase ($AUC = 0.53$), and also it has an increased variability of performances with respect to the logistic regression classifier.

	Run 0	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Avg	Std
AUC	0.512	0.517	0.5	0.537	0.531	0.535	0.528	0.555	0.527	0.56	0.53	0.0175

Table 6.5: The values of the AUC for the final multilayer perceptron with 10 different seeds for the SMOTE function. The table displays also the average and the standard deviation of the 10 AUC value.

Analysing in details the scores predicted by these models for the observations of the test set, it is possible to identify a strange behaviour of the models

which could be a source of issues when it comes to practically apply it to perform prediction. Basically, in almost all the constructed models, the majority of the predicted scores are about 0.45 and 0.55. With predicted probabilities that are so squeezed and close between each other, it can be very difficult to find an optimal threshold for the models, as a very small change in the value of the threshold can totally change the behaviour of the model.
The Figure 6.3 shows the evolution of the AUC value and of the loss function for one trained model across all the epochs. It is possible to notice how in the beginning the performances on the training and test set are very similar, and as the model trains it improves a lot its capacity of predicting elements of the training set, but the same does not happen on the test set. So in this case the model is strongly overfitting the training data despite the use of the regularization techniques.
The Figure 6.4 in the next page shows the ROC curve constructed for one particular model, together with two proposed thresholds, computed as done before for the logistic regression classifier. In this particular case, the two proposed thresholds coincide, and we have $\gamma_{EER} = \gamma_{DRC} = 0.52$. This brings to a very low True Positives Rate ($TPR = 0.109$ and a very high True Negatives Rate ($TNR = 0.981$), meaning that one person without COVID-19 has the 98.1% of probability to be classified as negative, but also a person with COVID-19 has a very high probability (89.1%) of being classified as negative. Basically, using that threshold, the model will predict almost everyone as not affected by COVID-19. This is a behaviour that we want to avoid, so it could be worth it to look for another threshold for this model, even if given the low AUC value every threshold will determine a model with some big drawback.

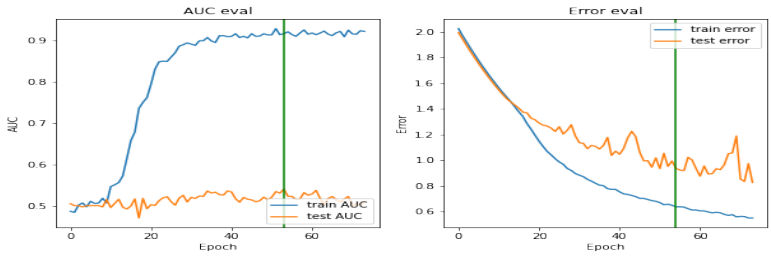


Figure 6.3: The evolution of the AUC and loss function values of one multilayer perceptron during its training. The green vertical lines represent the epoch in which the highest AUC on the test set has been observed ($AUC = 0.54$, at epoch 54). It is possible to see how in both plots after the first 10 epochs the curve for training and test set start to diverge, with the model learning very good how to classify the subjects of the training set, but struggling a lot with the ones of the test set. The loss function on the test set keeps decreasing for the whole training, but its behaviour shows more and more fluctuations as the training proceeds. For what concerns the AUC, the test set reaches its highest value after improving pretty constantly but very slowly for 54 epochs. The training lasted 74 epochs, which is way more than what it had taken in the phase of hyperparameters tuning with the same hyperparameters.

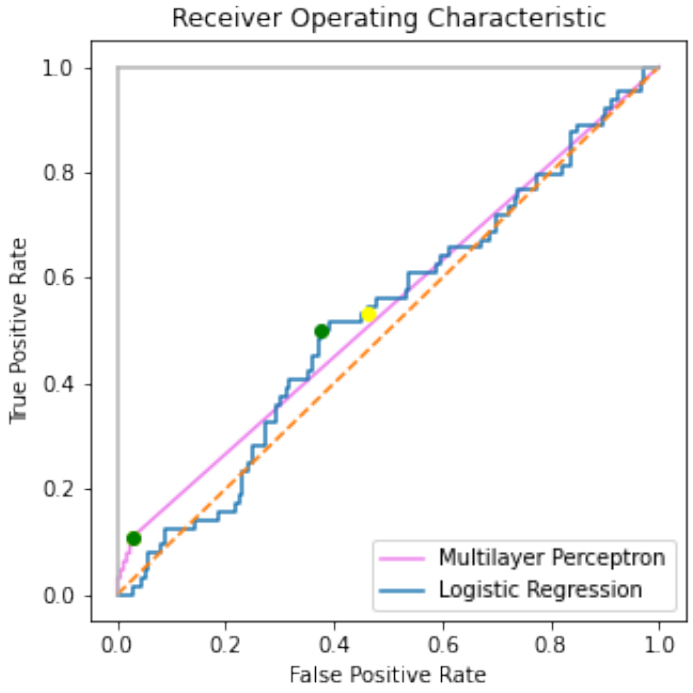


Figure 6.4: The ROC curves of one of the 10 fitted models for each classifier.
The curve of the logistic regression describes a classifier with performances very similar to the random classifier for extreme thresholds (bottom-left and top-right corners of the plot) and some improvement for intermediate values of the threshold. Here it is possible to see how the threshold γ_{EER} is a bit better than γ_{DRC} in terms of sensitivity (the yellow point on the blue curve is slightly higher than the green point), but the latter guarantees a good improving in terms of specificity (it is significantly more at the left).
The curve of the multilayer perceptron shows the largest improvements over the random classifier for thresholds who favour the specificity over the sensitivity (bottom-left corner). The MLP classifier gets closer and closer to a random classifier as the threshold is lifted. The performances of the MLP, differently from the ones of the LR, never get worse than the ones of the random classifier (the curve never crosses the orange line).

Chapter 7

Discussion

The results obtained in this work are far from being as promising as other results which have been obtained in different published works during the last year.
The logistic regression model, which has been used as a baseline to compare the results obtained by other models, has performed poorly ($AUC = 0.58$ in the hyperparameters tuning phase, $AUC = 0.51$ on the test set), but has been able to recognize some useful patter in order to perform better than a "toss of coin" predictor (random guess of the class).
The support vector machine has not been able at all to learn some generalizable patterns from the test set, proving itself unable to predict better than a "toss of coin predictor" on the validation set (AUC = 0.5 in the hyperparameters tuning phase). This result a bit of surprising, since in the study of Pahar et al. the SVM performed consistently better than the logistic regression. One possible cause for this poor performance with respect to the logistic regression could have been the way in which the data set has been standardized. In fact, each of the 42 features has been standardized on an individual basis, in order to obtain for each subject 42 vectors of 18 entries (number of groups of frames) with zero mean and unitary standard deviation. This is the standard practice for audio signal processing and ensures also to have a good grade of comparability for the magnitude of different features.
Nevertheless, relying on the robustness of this first standardization, it has not been performed a proper standardization of the features across the whole dataset before performing the analysis, which would have rescaled each of the 756 columns of the dataset in order to obtain mean zero and unitary standard deviation for each of them. This could have deteriorated the performances of the support vector machine, as it is a method based on the distances between data points, and its performances can be affected consistently from a range of magnitude of the features which is too wide. The Figure 7.1 shows the value of the mean and standard deviation for all the 756 columns of the training data set.

On the other hand, the multilayer perceptron has shown some interesting improvements in the performances with respect to the logistic regression in the

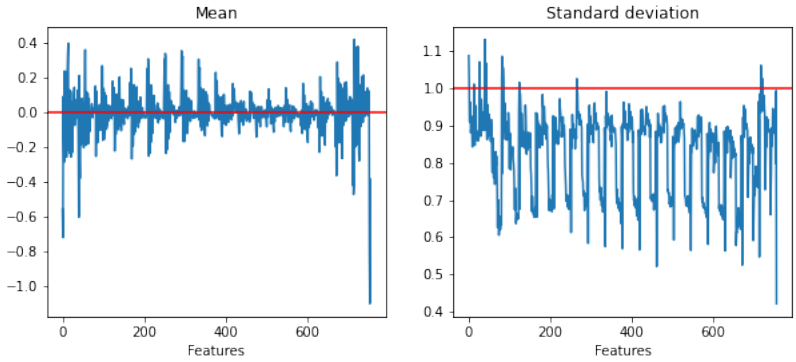


Figure 7.1: Mean and standard deviation for each column of the final dataset. They seem to follow a pretty systematic periodic scheme, with 18 well identifiable repetitions of it. This is easily explained by the structure of the dataset, which can be considered as 18 repetitions of the same set of features.
For what concerns the standard deviation, which is the one that can affect the performance of the classification algorithms, we can notice that its range is between 0.5 and 1.1, i.e. the column with minimum variability has half the s.d. of the one with the maximum one. This does not seem an exaggeratedly extreme situation, but still could be improved by standardizing each column before performing the analysis. The red lines in the plots represent the mean and standard deviation that one would obtain by doing that.

hyperparameter tuning phase ($AUC = 0.65$), but it turned out to perform a lot worse on the test set ($AUC = 0.53$). Moreover, in the several performed runs, the model often showed some strange values in terms of predicted probabilities on the validation or test set, which were usually all very close to 0.5, and almost always not less than 0.4. This is an undesirable behaviour, and it could be useful to change something in the structure of the classifier in order to avoid this "squeezing" of estimated probabilities.
Also, the causes of the dramatic deterioration of the performances on the unseen data should be subject of deeper analysis, as not only it could be due to some issue of the model, but it could also be caused by some systematic difference in the features of the audio of the training and test set. In that case, it could be worth to consider re-designing the analysis adding a stratification based on the time of submission when splitting the recordings between train and test set. Clearly, this was not possible in this work as the data used as test set were not available at the time in which the work has become.
The gap between the performances of the models in this study and in the one of Pahar et al., 2020 is most likely due to the fact that in this work several simplifications with respect to what is done in high-level studies has been performed at different stages of the work, from the data preprocessing to the feature extraction and the fit of the model.
In particular, the data preprocessing, as already mentioned before, has been per-

formed in a way that mimics only superficially the optimal way of proceeding to analyse recordings of people coughing. To overcome the issue of not having a good algorithm of cough detection at my disposal, I merely selected the loudest interval of one second from each recording, since in general it should include the loudest coughing for each person.
This way of proceeding can be a source of issues at different levels. First, even if the structure of the recordings ensures with some confidence that in the loudest second appears a cough, there is a high risk that in the audio recorded in noisy conditions the loudest second will also include a high level of noise, which results in feeding some bad data to the models. At the same time, even if we are pretty sure that one cough will be selected by finding the loudest 1-second window in the recording, every cough will appear in a different part of the interval, i.e. it can appear in the first half second, in the middle, in the end, in the worst case, it could even appear in the interval but be cut, as the last part of the cough is usually pretty quiet. On the other hand, in some case one interval of one second could contain two coughing, if they were short enough and not too far in time. This general lack of synchrony between the selected intervals which are used to build the models can be a problem especially for pretty simple models such as logistic regression and support vector machine, for which it is difficult to expect that they would learn from the data which part of the selected intervals is the useful one and how to automatically synchronize this part for different observations.
Another issue related with this way of selecting the data is that it discards a lot of potentially useful information from some observations. In fact, as seen in the Figure 2.6, more than a half of the recordings have a length larger than 5s, which means that selecting just 1s out of that recording corresponds to automatically discard the 80% or more of the recording. A way to avoid this issue could be to select multiple intervals for each observation and perform the analysis using some extensions of the models presented in this work which are built for datasets containing multiple observations from the same subject. This solution could also be a good way to increase the number of observations actually fed to the models, but at the same time it would force us to drop from the dataset the subjects which have uploaded recordings too short to provide the desired number of (non-overlapping) intervals, diminishing the already small number of different subjects included in the study.
For what concerns the features extraction and the training of the models, in order to avoid obtaining computational times exaggeratedly long, it was necessary to reduce the space of the hyperparameters tried out for the different models. As already mentioned, the number of mel frequency cepstral coefficients to keep in the model has been fixed to 13 were after some preliminary analysis proved that larger numbers did not improve the performances of the logistic regression. Anyway, the results of the study of Pahar et al., 2020 would suggest that it could be interesting to fit the multilayer perceptron keeping more MFCCs to see if this brings some improvement.
For what concerns the logistic regression model, the number of hyperparameters tried out has been significantly reduced with respect to the study of Pahar

et al., 2020, as in that paper in addition to l_1 and l_2 penalties it has been tried out also the linear combination of the two (elastic net) with the weight to give to each penalty used as additional hyperparameter to optimize.
Speaking of the multilayer perceptron, for the sake of computational time, the optimal dimension of the net, i.e. the optimal number of layers and neurons for each layer, have been fixed a priori. At the same time, also the set of potential values for the other hyperparameters has been reduced with respect to the one explored by the study of Pahar et al., 2020. In order to select which values to keep in the parameters set to explore, some preliminary analysis with different possible combination of hyperparameters have been performed.

Chapter 8

Conclusions and future works

In this work I have tried to use logistic regression, support vector machine and multilayer perceptron to build a prediction tool for the COVID-19 based on the sound of the cough of people.
Among the three classifiers, the one which has worked better is the multilayer perceptron with 10 hidden layers, as it was expected at the beginning. The logistic regression has performed worse than the MLP, but still better than the support vector machine, which was not able to classify the unseen subjects better than a model which randomly guesses the correct class.
As already stated before, this work is inspired on some works already published on the topic, but did not have the aim to obtain the same results, due to some obvious differences from the point of view of the availability of tools, knowledges and resources of the people involved. Nevertheless, this could be an interesting starting point for some work on audio signal processing able to actually classify coughs in a robust manner and to correctly identify people affected by COVID-19 based on this type of analysis.
The crucial change to perform in order to improve the results of the model could be the introduction of an algorithm able to detect the coughs from the recording. In this way the selection of intervals of fixed length, which is a very naive method, would be substituted by the selection of coughs, which is what actually matters for the purpose of this work.
At that point, as the coughs are of different length, the frames would be merged by fixing the number of groups of frames to obtain, as it has been done by Pahar et al. in their work. In this way the same number of features would be extracted from all the coughs, regardless their duration, and at the same time the dimensionality of the feature space would be reduced. Moreover, this style of merging would divide the features according to the phase of the cough to which they correspond, ensuring a synchronization of the features among different coughs which can be an important attribute for the dataset in order

to improve the performances of the predicting model.

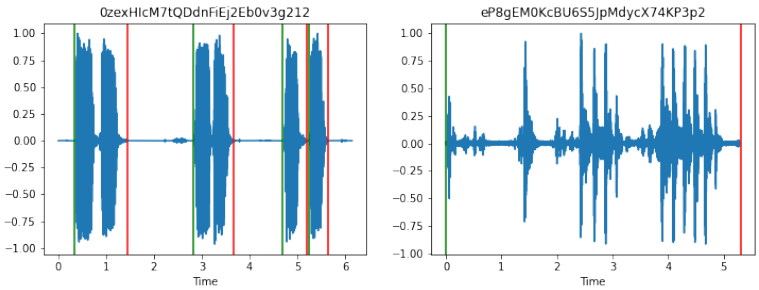


Figure 8.1: An overview of how the algorithm for the selection of the coughs currently works. The green lines represent the beginning of a cough, while the red lines represent the end. On the left we can see how the algorithm is able to identify the coughs on the audio with ID 0zexHIcM7tQDdnFiEj2Eb0v3g212, but it is able to correctly annotate the pair of coughs as two different events just for one pair out of three (the third); the other two pairs are considered as two unique events. This could seem a trivial issue, but can actually be very harmful for the analysis, and probably it is not solvable if not considering other features in the algorithm more than the loudness of each frame. On the right there is an example of a pretty noisy audio (ID: eP8gEM0KcBU6S5JpMdycX74KP3p2), in which the presence of continuous background noise prevents the algorithm to identify when one cough ends, so the whole audio is (wrongly) annotated as a cough. To overcome this problem it could be enough to tune properly the threshold below which the signal is considered silent, in order to discard more noise, but this could bring some other problems, so it needs to be tackled carefully.

Here too, a further improvement for the work could be obtained by selecting more than one cough for each subject. This should be doable without discarding any observations (or very few) since almost all the recordings have at least 2 coughs. Once extracted the desired number of coughs per subject, the data would be analysed to some extension of the logistic regression and the multilayer perceptron binary classifier for repeated observations.
A very basic algorithm for the identification of the coughs has already been constructed, but it was impossible to introduce it in this work since it is still unable to properly distinguish cough from continued noise, neither to identify two coughs not intercut by a pretty long silence as two different events. Anyway, it has already been proven effective on some audio of the dataset, and hopefully it will be a good starting point to improve this work. The Figure 8.1 shows how this algorithm for cough identification works on two different audio present in the CoSwara data set.

References

Amoh, J., & Odame, K. (2015). Deepcough: A deep convolutional neural network in a wearable cough detection system. *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 1–4.

Bachu, R., Kopparthi, S., Adapa, B., & Barkana, B. D. (2010). Voiced/unvoiced decision for speech signals based on zero-crossing rate and energy. *Advanced techniques in computing sciences and software engineering* (pp. 279–282). Springer.

Bagad, P., Dalmia, A., Doshi, J., Nagrani, A., Bhamare, P., Mahale, A., Rane, S., Agarwal, N., & Panicker, R. (2020). Cough against covid: Evidence of covid-19 signature in cough sounds. *arXiv preprint arXiv:2009.08790*.

Barry, S. J., Dane, A. D., Morice, A. H., & Walmsley, A. D. (2006). The automatic recognition and counting of cough. *Cough*, *2*(1), 1–9.

Bhateja, V., Taquee, A., & Sharma, D. K. (2019). Pre-processing and classification of cough sounds in noisy environment using svm. *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, 822–826.

Carfì, A., Bernabei, R., Landi, F., et al. (2020). Persistent symptoms in patients after acute covid-19. *Journal of the American Medical Association*, *324*(6), 603–605.

Chatrzarrin, H., Arcelus, A., Goubran, R., & Knoefel, F. (2011). Feature extraction for the differentiation of dry and wet cough sounds. *2011 IEEE international symposium on medical measurements and applications*, 162–166.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357.

Christodoulou, E., Ma, J., Collins, G. S., Steyerberg, E. W., Verbakel, J. Y., & Van Calster, B. (2019). A systematic review shows no performance benefit of machine learning over logistic regression for clinical prediction models. *Journal of Clinical Epidemiology*, *110*, 12–22.

Cohen-McFarlane, M., Goubran, R., & Knoefel, F. (2019). Comparison of silence removal methods for the identification of audio cough events. *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 1263–1268.

Cucinotta, D., & Vanelli, M. (2020). Who declares covid-19 a pandemic. *Acta Bio Medica: Atenei Parmensis*, *91*(1), 157.

Davis, S., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *28*(4), 357–366.

DeCarlo, L. T. (1997). On the meaning and use of kurtosis. *Psychological Methods*, *2*(3), 292.

Han, W., Chan, C.-F., Choy, C.-S., & Pun, K.-P. (2006). An efficient mfcc extraction method in speech recognition. *2006 IEEE international symposium on circuits and systems*, 4–pp.

Johns Hopkins University. (2021). Covid-19 dashboard by the center for systems science and engineering (csse).

Huang, X., Acero, A., Hon, H.-W., & Reddy, R. (2001). *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice hall PTR.

Imran, A., Posokhova, I., Qureshi, H. N., Masood, U., Riaz, M. S., Ali, K., John, C. N., Hussain, M. I., & Nabeel, M. (2020). Ai4covid-19: Ai enabled preliminary diagnosis for covid-19 from cough samples via an app. *Informatics in Medicine Unlocked*, *20*, 100378.

Knocikova, J., Korpas, J., Vrabec, M., & Javorka, M. (2008). Wavelet analysis of voluntary cough sound in patients with respiratory diseases. *J Physiol Pharmacol*, *59*(Suppl 6), 331–40.

Korpáš, J., Sadloňová, J., & Vrabec, M. (1996). Analysis of the cough sound: An overview. *Pulmonary Pharmacology*, *9*(5-6), 261–268.

Le Cessie, S., & Van Houwelingen, J. C. (1992). Ridge estimators in logistic regression. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, *41*(1), 191–201.

Liu, J.-M., You, M., Wang, Z., Li, G.-Z., Xu, X., & Qiu, Z. (2014). Cough detection using deep neural networks. *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 560–563.

McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E., & Nieto, O. (2015). Librosa: Audio and music signal analysis in python. *Proceedings of the 14th python in science conference*, *8*, 18–25.

Newman, T. (2017). All about coughs and their causes.

Olmedo, M. T. C., Paegelow, M., Mas, J.-F., & Escobar, F. (2018). *Geomatic approaches for modeling land change scenarios*. Springer.

Orlandic, L., Teijeiro, T., & Atienza, D. (2020). The coughvid crowdsourcing dataset: A corpus for the study of large-scale cough analysis algorithms. *arXiv preprint arXiv:2009.11644*.

Pahar, M., Klopper, M., Warren, R., & Niesler, T. (2020). Covid-19 cough classification using machine learning and global smartphone recordings. *arXiv preprint arXiv:2012.01926*.

Pramono, R. X. A., Imtiaz, S. A., & Rodriguez-Villegas, E. (2016). A cough-based algorithm for automatic diagnosis of pertussis. *PloS One*, *11*(9), e0162128.

Rudraraju, G., Palreddy, S., Mamidgi, B., Sripada, N. R., Sai, Y. P., Vodnala, N. K., & Haranath, S. P. (2020). Cough sound analysis and objective correlation with spirometry and clinical diagnosis. *Informatics in Medicine Unlocked*, *19*, 100319.

Samiee, K., Kovacs, P., & Gabbouj, M. (2014). Epileptic seizure classification of eeg time-series using rational discrete short-time fourier transform. *IEEE Transactions on Biomedical Engineering*, *62*(2), 541–552.

Santos, M. S., Soares, J. P., Abreu, P. H., Araujo, H., & Santos, J. (2018). Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches. *IEEE Computational Intelligence Magazine*, *13*(4), 59–76.

Sarangi, L., Mohanty, M. N., & Pattanayak, S. (2016). Design of mlp based model for analysis of patient suffering from influenza. *Procedia Computer Science*, *92*, 396–403.

Sharma, N., Krishnan, P., Kumar, R., Ramoji, S., Chetupalli, S. R., Ghosh, P. K., Ganapathy, S., et al. (2020). Coswara–a database of breathing, cough, and voice sounds for covid-19 diagnosis. *arXiv preprint arXiv:2005.10548*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288.

Tracey, B. H., Comina, G., Larson, S., Bravard, M., López, J. W., & Gilman, R. H. (2011). Cough detection algorithm for monitoring patient recovery from pulmonary tuberculosis. *2011 Annual international conference of the IEEE engineering in medicine and biology society*, 6017–6020.

Wang, D., Hu, B., Hu, C., Zhu, F., Liu, X., Zhang, J., Wang, B., Xiang, H., Cheng, Z., Xiong, Y., et al. (2020). Clinical characteristics of 138 hospitalized patients with 2019 novel coronavirus–infected pneumonia in wuhan, china. *Journal of the American Medical Association*, *323*(11), 1061–1069.

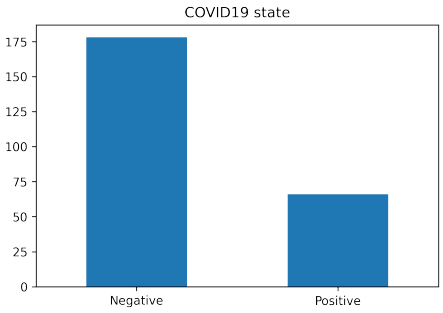
Windmon, A., Minakshi, M., Bharti, P., Chellappan, S., Johansson, M., Jenkins, B. A., & Athilingam, P. R. (2018). Tussiswatch: A smart-phone system to identify cough episodes as early symptoms of chronic obstructive pulmonary disease and congestive heart failure. *IEEE Journal of Biomedical and Health Informatics*, *23*(4), 1566–1573.

Appendices

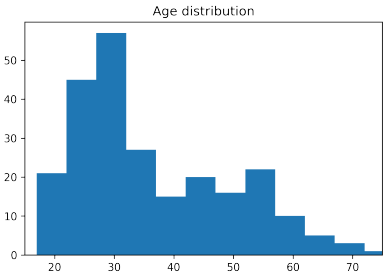
Appendix A

Plots about the test set

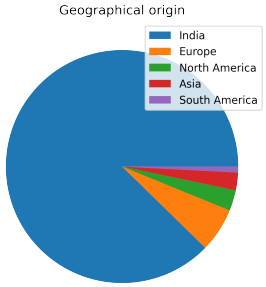
The plots below show the general information on the subjects in the test set. From them, it is possible to notice that the test set is slightly more balanced in terms of gender, and that the proportion of subjects positive to COVID-19 is higher than the proportion on the training set (27%).



(a) State of Covid-19 on the test set.



(b) Distribution in the age of the test set.



(c) Origin of the subjects in the test set.

Figure A.1

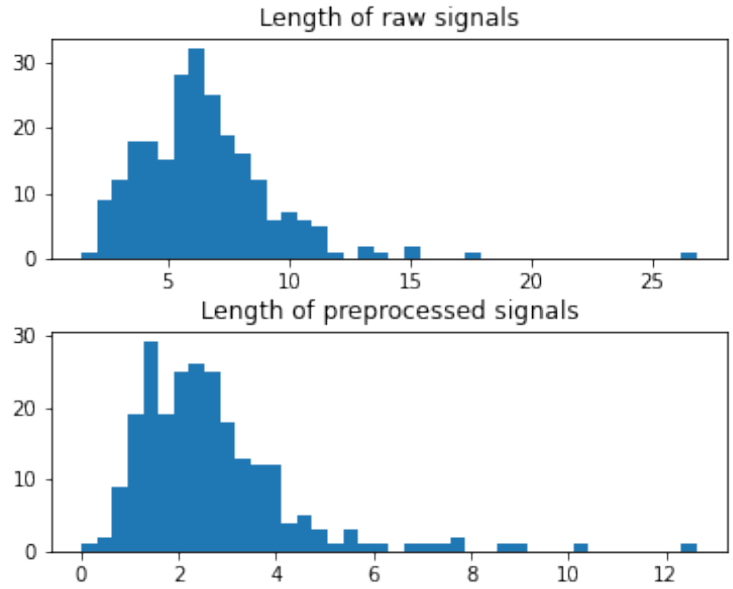


Figure A.2: Length of the recordings in the test set before and after the preprocessing.

Appendix B

Python code

The following pages contain the majority of the Python code used for the analysis, including the exploratory analysis, data preprocessing, features extraction and hyperparameters tuning phase for logistic regression and multilayer perceptron.
The lines of code used to produce plots have not been included for the sake of brevity.

B.1 Exploratory analysis

```
import json
import pandas as pd
import numpy as np
from pandas import read_csv
import matplotlib.pyplot as plt
import os
import pycountry_convert as pc

date_dir = [o for o in os.listdir(d)
                    if os.path.isdir(os.path.join(d,o))]
#where d the directory with all the subfolders
#containing the audio files

id_dirs = [os.path.join(d, b, o)
                for b in os.listdir(d)
                    if os.path.isdir(os.path.join(d,b))
                for o in os.listdir(b)
                    if os.path.isdir(os.path.join(d,b,o))]

metadata = {}
```

```
for id_dir in id_dirs:
    file = id_dir + "\metadata.json"
    with open(file) as json_file:
        metadata[id_dir[-28:]] = json.load(json_file)

combined_data = pd.DataFrame(metadata).T.drop('id', axis = 1)

combined_data['test_status']
[np.array([type(combined_data['test_status'][i])
           for i in range(len(combined_data))]) == float] = 'na'

which_pos = np.where([combined_data['covid_status'][i][0] == 'p'
                      for i in range(len(combined_data))])[0]
combined_data['Label'] = np.zeros(len(combined_data), int)
combined_data['Label'][which_pos] = int(1)
```

B.2 Data preprocessing

```
import librosa
import librosa.display

heavy_cough = "\\cough-heavy.wav"

def access_audio(audio_type):
    paths = [direct + audio_type for direct in id_dirs]
    audios = [librosa.load(file, sr=None) for file in paths]
    signals = [audios[i][0] for i in range(len(audios))]
    sample_rates = [audios[i][1] for i in range(len(audios))]
    lengths = [len(audios[i][0])/audios[i][1] \
               for i in range(len(audios))]

    return signals, sample_rates, lengths

#FUNCTION FOR REMOVING SILENCE USING AUDIO ENERGY
def remove_silence(signal, threshold=0.01, frame_dim=2048, \
                   hop_length=512):
    start_frame = np.arange(0, signal.shape[0], hop_length)
    end_frame = start_frame+frame_dim
    #zero padding
    signal = np.pad(signal, (0, end_frame[-1]-signal.shape[0]))

    to_remove = np.array([])

    for i in range(len(start_frame)):
```

```
            frame_idx = range(start_frame[i], end_frame[i])
            if np.max(np.abs(signal[frame_idx])) <= threshold:
                to_remove = np.append(to_remove, frame_idx)

    to_remove = np.unique(to_remove).astype(int)

    signal = np.delete(signal, to_remove)

    return signal

id_codes = combined_data.index

hc_signals, hc_sr, hc_lengths = access_audio(heavy_cough)
#access all the files called "cough-heavy.wav"

hc = pd.DataFrame(list(zip(hc_signals, hc_sr, hc_lengths)),
                  columns = ["Signal", "Sampling Rate", \
                                  "Recording length"],
                  index = id_codes)
hc = hc.loc[hc["Sampling Rate"] >= 44100, :]
hc = hc.loc[[len(a)>0 for a in hc["Signal"]], :]
hc = hc.loc[[np.abs(a).max()>0 for a in hc['Signal']], :]

proc_signals = []
proc_lengths = []

for idx in hc.index:
    signal = hc.loc[idx, "Signal"]
    sr = hc.loc[idx, "Sampling Rate"]
    if sr > 44100:
        signal = librosa.resample(signal, sr, 44100)
    signal = remove_silence(librosa.util.normalize(signal))
    length = librosa.get_duration(signal, sr = 44100)
    proc_signals.append(signal)
    proc_lengths.append(length)

hc["Processed signal"] = proc_signals
hc["Processed audio length"] = proc_lengths

hc = pd.concat([combined_data['Label'], hc], axis = 1, join = "inner")
```

B.3 Features extraction

```
import librosa
```

```
import librosa.display
from scipy.stats import kurtosis
from sklearn.preprocessing import StandardScaler

def features(signal, frame_length = 2048, hop_length=512, n_mfcc=13):

    #mfcc, delta, delta delta and zcr extracted using librosa
    mfccs = librosa.feature.mfcc(signal,
            n_fft=frame_length, hop_length=hop_length, n_mfcc=n_mfcc)
    mfccs = StandardScaler().fit_transform(mfccs.T).T
    mfccs_delta = librosa.feature.delta(mfccs)
    mfccs_delta =
    StandardScaler().fit_transform(mfccs_delta.T).T
    mfccs_delta2 = librosa.feature.delta(mfccs, order=2)
    mfccs_delta2 = StandardScaler().fit_transform(mfccs_delta2.T).T
    zcr = librosa.feature.zero_crossing_rate(signal,
            frame_length=frame_length, hop_length=hop_length)
    #log energy and kurtosis extracted recursively
    log_energy = np.array([
        np.log(1e-7 + sum(abs(signal[i:i+frame_length]**2))/frame_length)
        for i in range(0, len(signal), hop_length)
    ])
    kurt = np.array([
        kurtosis(signal[i:i+frame_length])
        for i in range(0, len(signal), hop_length)
    ])
    zcr = StandardScaler(). \
    fit_transform(zcr.reshape(-1, 1)).reshape(1, -1)
    log_energy = StandardScaler(). \
    fit_transform(log_energy.reshape(-1, 1)).reshape(1, -1)
    kurt = StandardScaler(). \
    fit_transform(kurt.reshape(-1, 1)).reshape(1, -1)

    return mfccs, mfccs_delta, mfccs_delta2, zcr, log_energy, kurt

def search_best_segment(processed_signals, sr=44100, dur=1, hop=0.5):
    best_segments = []
    for signal in processed_signals:
        if signal.shape[0]<sr*dur:
            signal = np.pad(signal, (0, int(1+sr*dur-signal.shape[0])))
        d = signal.shape[0]
        st = np.arange(0, int(d-dur*sr), int(hop*sr))
        energies = [np.abs(signal[int(s):int(s+(dur*sr))]).sum() \
                                                    for s in st]
        p = np.argmax(energies)
        s = st[p]
```

```
        best_segments.append(librosa.util.normalize( \
        signal[int(s):int(s+(dur*sr))]))
    return best_segments

def extract_features(best_segments):
    extracted = []
    for processed_signal in best_segments:
        mfccs, mfcc_delta, mfcc_delta2, zcr,
        log_energy, kurt = features(processed_signal)
        if log_energy.shape[1] != mfccs.shape[1]:
            print(mfccs.shape[1]-log_energy.shape[1])
        extracted.append(np.concatenate((mfccs,
            mfcc_delta, mfcc_delta2, zcr, log_energy, kurt)))

    return extracted

def merge_features(features_list, segment_width=5):
    merged = []
    sw = segment_width
    fs = [f.shape[1] for f in features_list][0]
    for features in features_list:
        merged.append(np.array([features[:, i:i+sw].mean(axis=1)
        for i in np.arange(0, fs, sw)]).reshape(1, -1))

    return merged

l = search_best_segment(hc['Signal'], dur = 1)
e = extract_features(l)
m = merge_features(e, segment_width = 5)
X = np.concatenate(m)
y = hc['Label']
```

B.4 Logistic regression (cross validation)

```
from sklearn.model_selection import StratifiedKFold,
cross_val_score, train_test_split, GridSearchCV, cross_validate
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn.metrics import roc_auc_score, \
confusion_matrix, accuracy_score

#function to print the metric of the model
def prediction_summary(y, y_pred):
```

```
    set_dimension = len(y)
    tn, fp, fn, tp = confusion_matrix(y, y_pred).ravel()
    accuracy = accuracy_score(y, y_pred)
    auc = roc_auc_score(y, y_pred)

    print("Set dimension: {}".format(set_dimension))
    print("True Positives: {} \nFalse Positives: {} \
    \nTrue Negatives: {} \nFalse Negatives: {}".format(tp, fp, tn, fn))
    print("Accuracy: {}".format(round(accuracy, 3)))
    print("Area Under ROC Curve: {}".format(round(auc, 3)))

#loop for the cross validation (oversample, fit, assess)
def score_model(model, params, X, y, cv, rs=777):
    smoter = SMOTE(random_state=rs)

    scores = []

    for train_fold_index, val_fold_index in cv.split(X, y):
        X_train_fold, y_train_fold = X[train_fold_index], \
                                     y[train_fold_index]
        X_val_fold, y_val_fold = X[val_fold_index], y[val_fold_index]

        X_train_fold_upsample, y_train_fold_upsample = \
        smoter.fit_resample(X_train_fold, y_train_fold)
        model_obj = model(**params).\
        fit(X_train_fold_upsample, y_train_fold_upsample)
        score = roc_auc_score(y_val_fold, model_obj.predict(X_val_fold))
        scores.append(score)
    return np.array(scores)

skf = StratifiedKFold(n_splits=5, random_state=7, shuffle=True)

lr_tracker = []
Cs = []
pens = ['l1', 'l2']
lr = LogisticRegression
for C in np.logspace(-7, 7, 15):
    lo = []
    Cs.append(str(C))
    for solver in ['saga', 'lbfgs']:
        if solver == 'saga':
            penalty = 'l1'
        else:
            penalty = 'l2'
        print('C: ' + str(C) + ', Penalty: ' + penalty)
        sc = score_model(lr, {'C': C, 'max_iter': 10000, \
```

```
        'solver': solver, 'penalty': penalty}, X, y, cv = skf)
        print(sc.mean())
        lo.append(sc.mean())
    lr_tracker.append(lo)

lr_trk = pd.DataFrame(lr_tracker)
lr_trk.index = Cs
```

B.5 Multilayer perceptron (cross validation)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, \
Dropout, Conv2D, BatchNormalization, MaxPool2D, Flatten
from tensorflow.keras.utils import plot_model
from tensorflow.keras.metrics import AUC
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2

skf = StratifiedKFold(n_splits=5, random_state=7, shuffle=True)

def built_mlp(dr = 0.2, C = 1e-2, lr = 1e-4, input_shape = X.shape[1]):

    model = Sequential()
    model.add(Dense(1024, input_dim = input_shape, activation = 'relu'))

    model.add(Dense(512, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
    model.add(Dropout(dr))
    model.add(Dense(256, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
    model.add(Dropout(dr))
    model.add(Dense(128, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
    model.add(Dropout(dr))
    model.add(Dense(64, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
    model.add(Dropout(dr))
    model.add(Dense(32, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
    model.add(Dropout(dr))
    model.add(Dense(16, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
```

```
    model.add(Dropout(dr))
    model.add(Dense(8, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
    model.add(Dropout(dr))
    model.add(Dense(4, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
    model.add(Dropout(dr))
    model.add(Dense(2, kernel_regularizer=l2(C), \
            bias_regularizer=l2(C), activation = 'relu'))
    model.add(Dropout(dr))

    model.add(Dense(1, activation = 'sigmoid'))

    model.compile(loss = 'binary_crossentropy', \
                  optimizer = Adam(learning_rate = lr), \
                  metrics = ['accuracy', AUC(name = 'auc')])

    return model

def cv_score(X, y, dr_ = 0.2, C_ = 1e-2, lr_ = 1e-4, \
             cv = skf, ep = 20, bs_ = None):
    if bs_ == None:
        bs_ = X.shape[1]

    aucs = []
    epochs = []

    for train_fold_index, val_fold_index in skf.split(X, y):
        X_train_fold, y_train_fold = \
        X[train_fold_index], y[train_fold_index]
        X_val_fold, y_val_fold = \
        X[val_fold_index], y[val_fold_index]

        X_tr2, y_tr2 = \
        SMOTE().fit_resample(X_train_fold, y_train_fold)
        monitor = EarlyStopping(monitor='val_auc', \
                                min_delta=0, patience=20, verbose = 0,
                                mode = 'max', restore_best_weights = True)
        model = built_mlp(dr = dr_, C = C_, lr = lr_)
        model_obj = model.fit(X_tr2, y_tr2, epochs = ep, verbose = 0 \
                    batch_size = bs_, callbacks = [monitor], \
                    validation_data = (X_val_fold, y_val_fold))
        auc = np.array(model_obj.history['val_auc']).max()
        aucs.append(auc)
        epoch = len(model_obj.history['loss'])
        epochs.append(epoch)
```

```
    return aucs, epochs

drs = np.arange(0.15, 0.31, 0.05)
Cs = np.logspace(-7, -3, 5)
lrs = np.logspace(-5, -3, 3)
bss = [64, 128, 256]

mlp_tracker = pd.DataFrame(columns = ['C', 'LR', \
            'DR', 'BS', 'AUC', 'Avg_AUC', 'Ep', 'AVG_Ep'])

for C in Cs:
    for lr in lrs:
        for dr in drs:
            for bs in bss:
                res = cv_score(X = X, y = y, dr_ = dr, C_ = C, \
                               lr_ = lr, ep = 1000, bs_ = bs)
                mean_auc = np.array(res[0]).mean()
                mean_epochs = np.array(res[1]).mean()
                new_row = {'C': C, 'LR': lr, 'DR': dr, 'BS': bs, \
                           'AUC': np.array(res[0]), 'Avg_AUC': mean_auc, \
                           'Ep': np.array(res[1]), 'AVG_Ep': mean_epochs}
                mlp_tracker = mlp_tracker.append(new_row,
                                                 ignore_index = True)
```