Politecnico di Torino

Cardiotocography Binary Classification

Mathematics in Machine Learning

Master degree in Data Science and Engineering

Author: Chiara Tomei s277790

Professors: Francesco Vaccarino, Mauro Gasparini

May 8, 2022

Contents

1 Introduction

The aim of this study is to introduce the main concepts in supervised machine learning by presenting techniques and strategies to diagnose potentially ill patients from the Cardiotocography exam. Cardiotocography (CTG) is a continuous recording of the fetal heart rate obtained via an ultrasound transducer placed on the mother's abdomen. CTG is widely used in pregnancy as a method of assessing fetal well-being, predominantly in pregnancies with increased risk of complications.

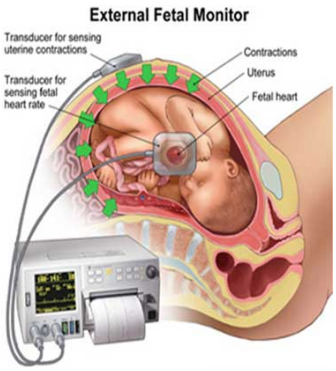


Figure 1: Cardiotocography Machine

The study of the parameters derived from a Cardiotocography can be very useful in recognizing warning signs that could endanger the life of the fetus. The primary purpose of fetal surveillance by CTG is to prevent adverse fetal outcomes.
The goal is to build an automated model for a **binary diagnosis** (normal, pathologic), based on the fetal cardiotocography. The general concepts of the supervised machine learning paradigm are later reported, together with a detailed explanation of all techniques and algorithms used to build the models. In particular KNN, Decision Trees, Random Forest, Support Vector Machine algorithms have been applied.

2 Envinronment

The analysis has been carried out with Python programming language, using different machine learning and statistical frameworks and free libraries such as *sklearn, numpy, pandas, imblearn* and some data visualization libraries (*matplotlib and seaborn*).
The different libraries have allowed to implement more easily both the classification methods and the preprocessing parts such as oversampling, undersampling, k fold cross validation, dimensionality reduction or splitting of the database.
The entire code of this project is public available at the following link: https://github.com/tomeichiara/CardiotocographyDataSet

3 Dataset exploration

The dataset is composed of 2126 samples from different patients and 20 features which are preprocessed statistics, derived from the analysis made by the Cardiotocography exam and which describe the key parameters of Uterine contraction and Fetal Heart Rate (FHR).
The dataset is available at the following link: https://archive.ics.uci.edu/ml/datasets/cardiotocography

3.1 Feature details

1. **LB**: baseline FHR → is the mean level of the FHR when this is stable, excluding accelerations and decelerations. It is determined over a time period of 5-10 minutes, expressed as beats per minute (bpm). Preterm fetuses tend to have values towards the upper end of the normal range.
2. **AC**: number of accelerations per second → Accelerations are transient increases in FHR of 15bpm or more above the baseline and lasting 15 seconds. Accelerations in preterm fetuses may be of lesser amplitude and shorter duration.
3. **FM**: number of fetal movements recorded by a fetal monitor per second. → An excessively low number of fetal movements may be a risk factor.
4. **UC**: number of uterine contraction per second → Uterine contractions reduce the blood flow to the placenta thus reducing fetal oxygenation. Consideration must be given to the frequency of contractions with particular attention paid to the rest between them.
5. **DL**: number of light decelerations per second → decelerations are transient episodes of decrease of FHR below the baseline of more than 15 bpm lasting at least 15 seconds.The fetus will not decelerate its heart rate without physiological provocation. Uterine activity, even in its mildest form, will result in decelerations in a fetus whose oxygenation is already compromised. Some of them are benign and associated with sleep cycles.
6. **DS**: number of severe decelerations per second
7. **DP**: number of prolonged decelerations per second → are defined as a decrease of FHR below the baseline of more than 15 bpm for longer than 90 seconds but less than 5 minutes.
8. **ASTV**: percentage of time with abnormal short term variability →Baseline variability is the minor fluctuation in baseline FHR. It is assessed by estimating the difference in bpm between the highest peak and lowest trough of fluctuation in one minute segments of the trace.
9. **MSTV**: mean value of short term variability
10. **ALTV**: percentage of time with abnormal long term variability
11. **MLTV**: mean value of long term variability
12. **Width**: width of FHR histogram
13. **Min**: minimum of FHR histogram
14. **Max**: maximum of FHR histogram
15. **Nmax**: number of histogram peaks
16. **Nzeros**: number of histogram zeros
17. **Mode**: histogram mode
18. **Mean**: histogram mean
19. **Median**: histogram median
20. **Variance**: histogram variance

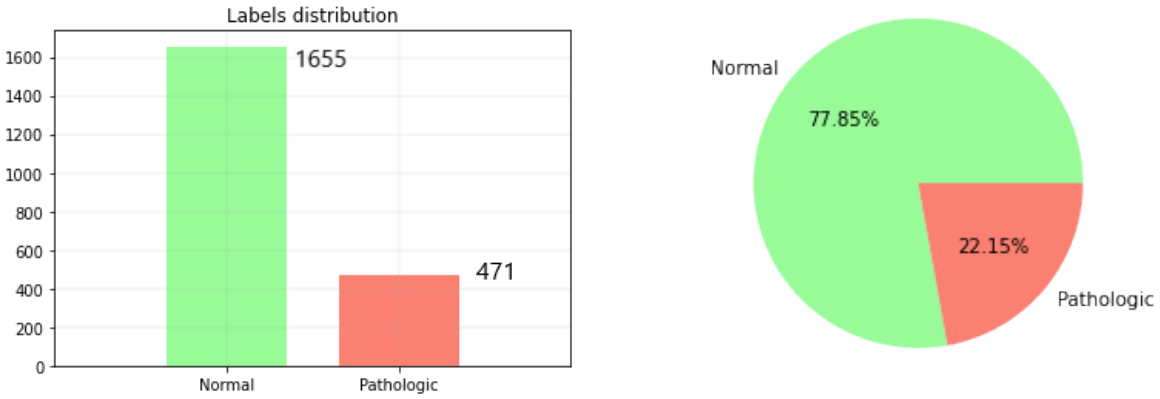
Using these features and the help of 6 experienced midwives, two types of classification were performed: the first concerns the FHR pattern classification with 10 possible labels. The second, the one considered in this project, concerns the status of the fetus, which has been reduced to a binary classification with two possible classes: **normal fetus or fetus in pathological condition**.

3.2 Missing values

No missing or null values were detected within the entire dataset. All features present have 2126 values.

3.3 Class Distribution

The first important statistic calculated concerns the distribution of the two labels. A correct and balanced distribution of the classes is fundamental to define the problem as balanced and not to run into problems during the training phase of the classification algorithm.
Through a simple analysis it is clear that in our case the classes are rather unbalanced. We observe 1655 samples belonging to the class of normal fetuses and only 471 to the class of pathologic fetuses.



(a) Histogram (b) Pie Chart

Figure 2: Class distribution (Normal-Pathological)

This means that our starting dataset is composed of 77.85% of samples belonging to the normal class and 22.15% to the pathological class. For this reason it will be necessary to implement techniques to mitigate the imbalance, through re-sampling action like under or over sampling.

3.4 Statistics

Statistics for all other remaining features were then computed. It was chosen to analyze:

- Mean: sample mean $\rightarrow \overline{x} = \frac{1}{n}\sum_i^n x_i$;
- Variance: sample variance $\rightarrow \sigma^2 = \frac{1}{n-1}\sum_i^n (x_i - \overline{x})^2$;
- First, second, and third quartiles
- The minimum value
- The maximum value
- Skewness Index: computed as follows:

$$skewness = \frac{\sum_i^N (x_i - \overline{x})^3 / N}{s^3} \tag{1}$$

where $\overline{x}$ is the mean, s is the standard deviation and N the number of data points. Note that in computing the skewness, the standard deviation is computed using N in the denominator rather than N - 1.

- Kurtosis Index: computed with the Fisher definition as

$$kurtosis = \frac{\sum_i^N (x_i - \overline{x})^4 / N}{s^4} - 3 \tag{2}$$

where $\overline{x}$ is the mean, s is the standard deviation and N the number of data points. Note that in computing the kurtosis, the standard deviation is computed using N in the denominator rather than N - 1. The Fisher's definition, subtracting -3 makes the normal distribution have zero Kurtosis.

	mean	std	min	25%	50%	75%	max	skewness	kurtosis
LB	133.303857	9.840844	106.0	126.000000	133.000000	140.000000	160.000000	0.020312	-0.292943
AC	0.003170	0.003860	0.0	0.000000	0.001630	0.005631	0.019284	1.210085	0.784374
FM	0.009474	0.046670	0.0	0.000000	0.000000	0.002512	0.480634	7.812157	64.266963
UC	0.004357	0.002940	0.0	0.001876	0.004482	0.006525	0.014925	0.156377	-0.649461
DL	0.001885	0.002962	0.0	0.000000	0.000000	0.003264	0.015385	1.721570	2.506369
DS	0.000004	0.000063	0.0	0.000000	0.000000	0.000000	0.001353	17.811702	321.084735
DP	0.000157	0.000580	0.0	0.000000	0.000000	0.000000	0.005348	4.278801	20.077857
ASTV	46.990122	17.192814	12.0	32.000000	49.000000	61.000000	87.000000	-0.011829	-1.051030
MSTV	1.332785	0.883241	0.2	0.700000	1.200000	1.700000	7.000000	1.657339	4.700756
ALTV	9.846660	18.396880	0.0	0.000000	0.000000	11.000000	91.000000	2.195075	4.252998
MLTV	8.187629	5.628247	0.0	4.600000	7.400000	10.800000	50.700000	1.331998	4.131254
Width	70.445908	38.955693	3.0	37.000000	67.500000	100.000000	180.000000	0.314235	-0.902287
Min	93.579492	29.560212	50.0	67.000000	93.000000	120.000000	159.000000	0.115784	-1.290422
Max	164.025400	17.944183	122.0	152.000000	162.000000	174.000000	238.000000	0.577862	0.632769
Nmax	4.068203	2.949386	0.0	2.000000	3.000000	6.000000	18.000000	0.892886	0.504211
Nzeros	0.323612	0.706059	0.0	0.000000	0.000000	0.000000	10.000000	3.920287	30.365084
Mode	137.452023	16.381289	60.0	129.000000	139.000000	148.000000	187.000000	-0.995178	3.009531
Mean	134.610536	15.593596	73.0	125.000000	136.000000	145.000000	182.000000	-0.651019	0.933427
Median	138.090310	14.466589	77.0	129.000000	139.000000	148.000000	186.000000	-0.478414	0.667259
Variance	18.808090	28.977636	0.0	2.000000	7.000000	24.000000	269.000000	3.219974	15.131589

Figure 3: Multiple statistics

Skewness and **Kurtosis** are two indices that help us understand the structure and shape of the data distribution.

In particular, *Skewness* is a degree of asymmetry observed in a probability distribution that deviates from the symmetrical normal distribution in a given set of data. The symmetrical distribution has zero skewness, where mean, median and mode are the same value. We talk of positive Skewness when the mean is greater than the median and a large number of data are on the right-side of the curve. The negative Skewness occurs when the mean of the dataset is less than the median and a large number of data lies on the left-side (figure 4).

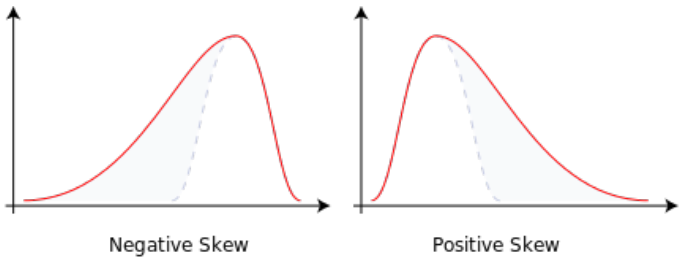


Figure 4: Skewness representation

The *Kurtosis* index refers to the degree of presence of outliers in the distribution. It is a statistical measure, it indicates whether the data is heavy-tailed or light tailed respect to a normal distribution. It has been used the Fisher's Kurtosis formula, according with, distributions which have 0 or very close to zero kurtosis are called Mesokurtic distributions. Normal distribution falls under this bucket. Distributions that are uniform or flat-topped, have negative Fisher's kurtosis and are also called Platykurtic distributions. Distributions with high positive Fisher's kurtosis are called Leptokurtic distributions. Leptokurtic distributions are tail-heavy distributions that suffer from outliers that may require handling or processing depending on the use case.

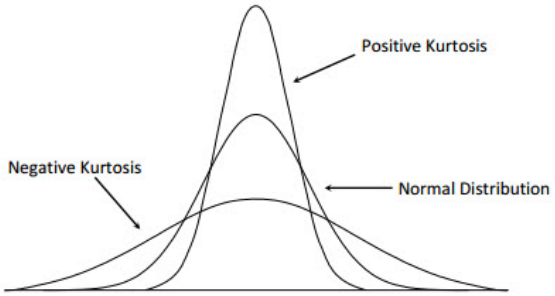


Figure 5: Kurtosis representation

From the calculated statistics, it can be seen that the data space and the range of means and variances is very large. Therefore a data preprocessing is necessary to prevent misleading output.

3.5 Boxplot

For further feature analysis, boxplots were created, in order to understand how the data are distributed. First, the boxplots were plotted leaving the features as they were originally.
However, many of the measurements that are '*per second*' are very small and close to zero, so the scales are very different and it is impossible to compare the different data.

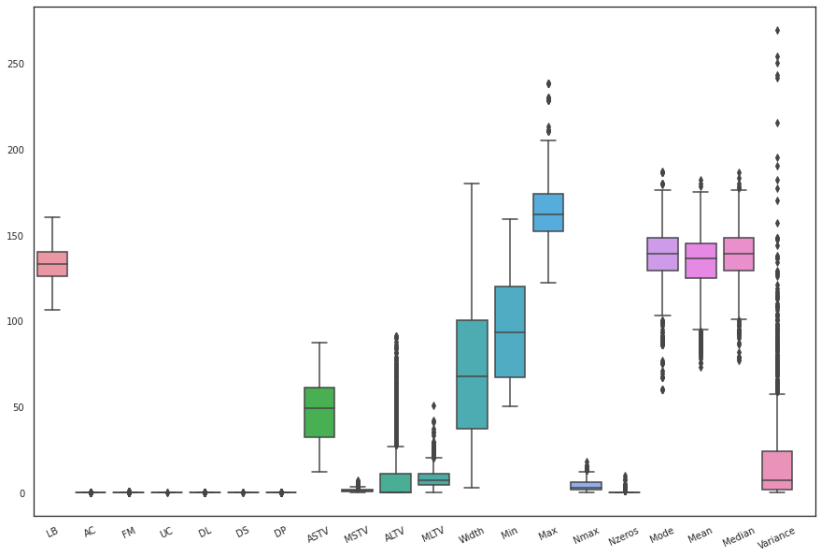


Figure 6: Boxplot with original data

For this reason it was decided to also create boxplots of normalized features, in order to better appreciate a direct comparison. From the graph below we can see for each feature the 1°, 2° and 3° quartile as well as a minimum and maximum and those that are considered outliers. Defining the interquartile range as $IQR = (Q_3 - Q_1)$, every sample placed beyond $Q_1 - 1.5IQR$ and $Q_3 + 1.5IQR$ is considered an outlier. It is possible to change the coefficient of 1.5 in order to change the threshold of outliers.

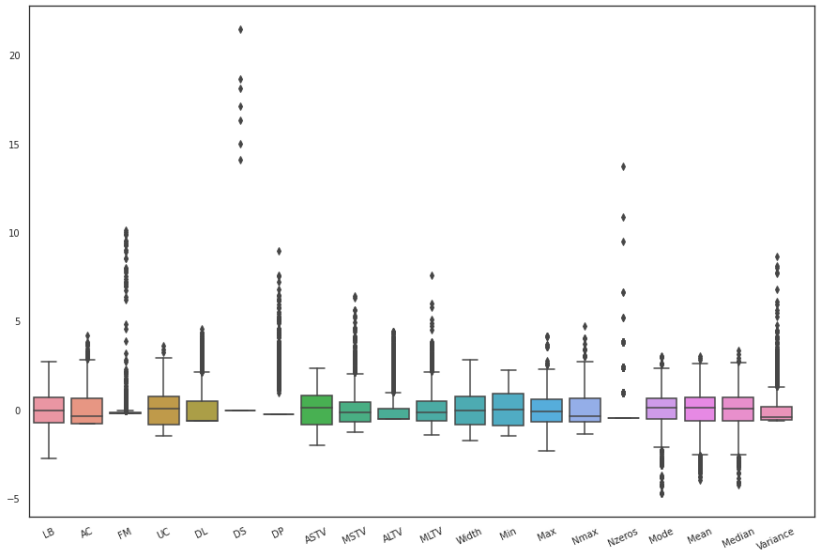


Figure 7: Boxplot with normalized data

3.6 Outliers

As for outliers, a more in-depth study was needed. From what was emerged through the calculation of the Kurtosis index it is clear that there are much data potentially in the range of outliers. It is necessary to understand if they are really a sign of noise and troubles or if they have an important and useful meaning.

An **Isolation Forest** was used for outlier detection. Isolation Forest, like any tree ensemble method, is built on the basis of decision trees. While with normal decision trees we try to partition the data so that the partitions are as homogeneous as possible, using isolation trees we look for a way to partition the space so that we isolate the points. Clearly an 'anomalous' point is much easier to been isolated than a regular point. In the example below, you can see that many more partitions are required to locate a regular point than an anomalous one.

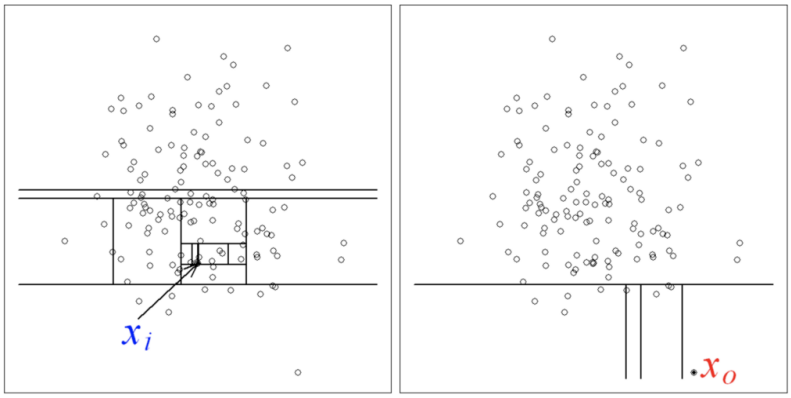


Figure 8: Isolation algorithm graphic representation

In the Isolation Forest you take a sample of the dataset and build isolation trees until the points are isolated. In these trees, partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature.

For the test part it is necessary to calculate an anomaly score for each new point. The calculation is done with the following formula:

$$S(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{3}$$

where $h(x)$ is the path length of observation x, $c(n)$ is the average path length and n is the number of external nodes. With this particular formula we compare the required depth-path for point x with the average required depth-path.

- If $E(h(x)) << c(n)$ we obtain a $S(x,n)$ near to 1 that indicates anomalies and so outliers.
- If $E(h(x)) \simeq c(n)$ we obtain a $S(x,n)$ near to 0.5. Score close or smaller than 0.5 indicates normal observations.

Once we have run the code and made the prediction we can observe the scores obtained on each data point (figure 9).

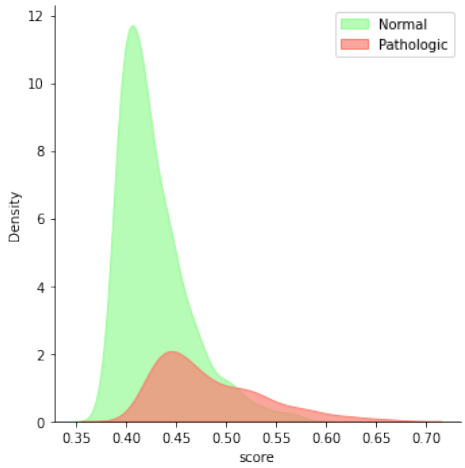


Figure 9: Isolation Forest scores

We observe that in the pathological class many more data points have high scores and therefore are considered as outliers. We can then insert as a parameter the *Contamination Index* that selects the percentage of dataset to be considered outliers. If we set, for example, the contamination index equal to 5% we have obtained a total of 107 outliers. As can be seen from the histograms (10), most of them belong to the class of pathological fetuses.

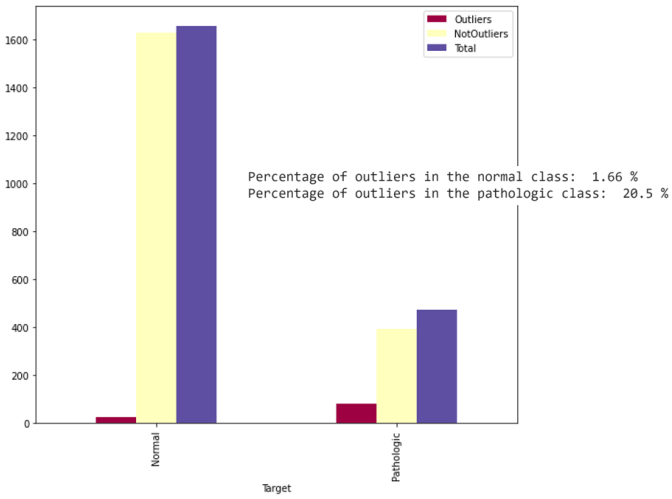


Figure 10: Outliers distribution between the two classes

In particular it was calculated that only 1.66% of normal samples were recognized as outliers while 20.5% of pathological samples would be negligible according to the algorithm. This would clearly affect our algorithm in a negative way, eliminating a large part of the pathological class, already rather under-represented. It is useful to investigate the real meaning of these outliers. For example, when we have a LB particularly out of range is a symptom of premature fetuses and therefore a very relevant information for the classification of the disease. Wanting to define the anomalies of CTG, which lead to the diagnosis of pathologies in the fetus, it would be self-defeating to remove the information useful to identify these abnormal values.

3.7 Feature distribution

To understand the relationship between features and the target of our classification, it has been useful investigate how the distribution of features varies as the labels change. In this way we can analyze if there is a relationship between the trend of a certain feature and the change of class or if instead the distribution remains unchanged.

(a) AC (b) ALTV (c) ASTV (d) DL

(e) DP (f) DS (g) FM (h) LB

(i) Max (j) Mean (k) Median (l) Min

(m) MLTV (n) Mode (o) MSTV (p) NMax

(q) NZeros (r) Variance (s) Width (t) UC

Figure 11: Kde distributions for each class

In order to analyze the distribution we used a **density function** estimated using the *kernel*

density estimation (KDE) method which is a parametric method for visualizing the distribution of observations in a dataset, analagous to a histogram. KDE represents the data using a continuous probability density curve.

The estimated density function f_h at any given x is calculated as

$$f_h(x) = \frac{1}{Nh} \sum_i K(\frac{x - x_i}{h}) \tag{4}$$

where K is a kernel function(in this plot, it is the gaussian kernel $g(x) = e^{-\pi * x^2}$) and h is a smoothing parameter called bandwidth.

From the analysis carried out we note that there are some features that maintain almost unchanged the distribution while others deviate significantly as the $ASTV$ (% of time with abnormal short term variability). It is easy to understand the reason since a higher percentage of abnormalities leads to a more likely pathological fetus. For this reason the distribution of the pathological class is more shifted towards higher percentages on the right of the graph. Another example regards the feature AC that is the number of acceleration of the Fetal Heart Rate (FHR). Acceleration in preterm fetuses are less or absent and for this reason there is a large peak above zero in the pathological class, while the normal class, despite also having the presence of zero values, has a distribution that reaches even higher values.

3.8 Correlation

The last analysis on the data has been to calculate the correlation between the variables. This analysis is very important for the robustness of the classification model that will be used. Multicollinearity can be quite negative for some models which assume that the predictors are all independent and so, sometimes, it is useful to drop some redundant features.

For this calculation was chosen the **Pearson coefficient**. The correlation between variables indicates the type of the relationship between them. A weaker relationship means a low correlation value. The maximum value of correlation is equal to 1 while for a data totally not correlated the value is equal to 0. A negative correlation is possible, and his maximum value is -1. The Pearson correlation formula is the following:

$$\rho_{x,y} = \frac{Cov(x, y)}{\sigma_x \sigma_y} \in [-1, 1] \tag{5}$$

where $Cov(x, y)$ is the Covariance and it compares two variables in terms of deviation from their mean. σ_x and σ_y are the standard deviation of x and y.
In that formula the denominator squeezes the covariance to be a number from -1 to 1 and ensure that the scale of the data does not effect the correlation value. If data fall on a straight line (as can be seen in different example in figure 13), then the covariance and the product of variance terms are the same and the division gives us 1 or -1. When data do not fall in a straight line then the covariance accounts for less of the variance in the data, and the correlation is closer to 0.
Note that the Pearson coefficient is only able to capture **linear trends** and for this reason, for a more depth analysis it could be useful to try also other type of correlation, like the quadratic ones through R^2.

For an easier visualization of the correlation between features it has been plotted the correlation matrix using a *heatmap*. Since the correlation matrix is symmetrical it is reported just the lower diagonal matrix.

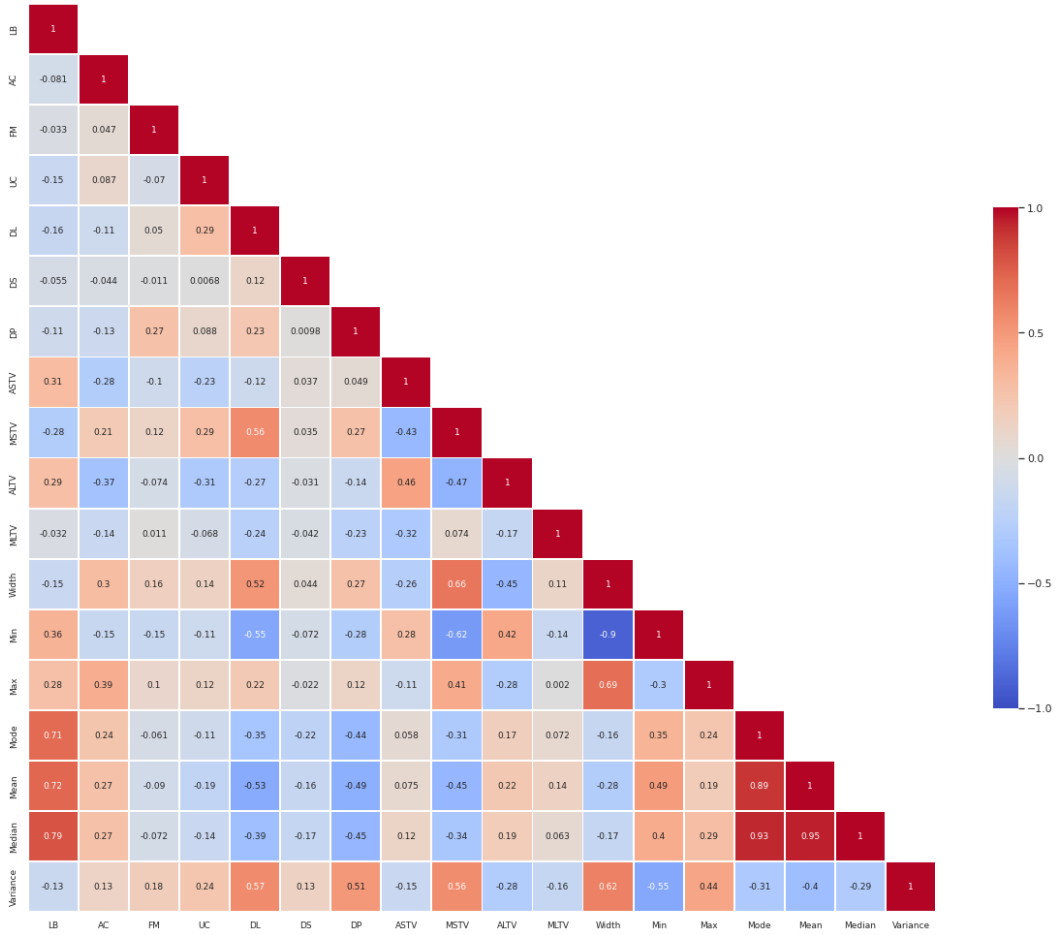
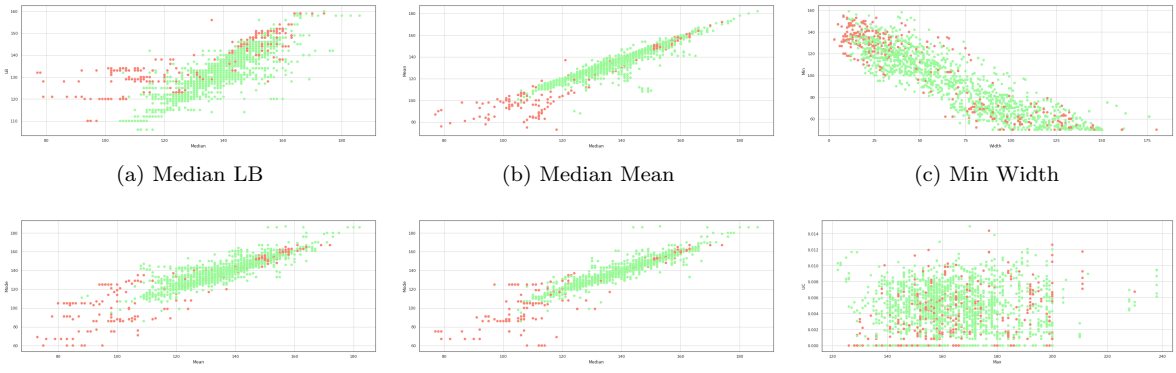


Figure 12: Correlation Matrix

Here we see how some variables actually appear with a rather high correlation index. These are the *mode*, *median* and the *mean*, which appear of course correlated. There is also a fairly high correlation between these three variables and the *baseline fetal heart rate (LB)* and this is easily explained by the fact that the *LB* is a kind of average of the FHR described by the histogram. *Width* and *min* also appear to be negatively correlated and this can be explained by the fact that the FHR histogram is wider the lower it is and the tighter the higher it (and therefore with a large minimum).

For further evidence of the correlation between these variables we can also plot some of their scatter plot which shows us the distribution in space of these particular pairs.



(a) Median LB (b) Median Mean (c) Min Width

(d) Mode Mean (e) Mode Median (f) UCMax

Figure 13: Scatter Plot of highly correlated features, except for (f) that is an example of two uncorrelated features

This analysis will be the basis of choices related to feature selection and feature reduction, since the same information could be encoded through fewer attributes.

4 Data Preprocessing

4.1 Feature selection

In order to limit the problem related to the multicollinearity of the variables, it was decided to take into account the correlation indices previously obtained to implement an appropriate feature selection. A correlation higher than 85% was chosen as a range to be considered. In this way we obtained all the pairs which are above this threshold. In particular they are:

- Median;Mean (95%)
- Mode;Median (93%)
- Min;Width (90%)
- Mean;Mode (89%)

It would clearly be disadvantageous, however, to eliminate all these pairs of variables because we would lose potentially very relevant information. It was therefore decided to eliminate one of the two variables in each pair and to maintain the one with the highest correlation index with the target variable, in order to keep the most significant one.

```
ASTV        0.493391
ALTV        0.489400
AC          0.394988
DP          0.340895
UC          0.263823
LB          0.251875
MSTV        0.207717
MLTV        0.172519
Min         0.158171
Width       0.142182
Mode        0.092320
DS          0.090782
Variance    0.085948
FM          0.060913
Nmax        0.060354
Mean        0.059107
Max         0.048106
Median      0.047890
Nzeros      0.031163
DL          0.030166
```

Figure 14: Correlation with the target variable

In detail for the variables selected the correlation with the label is equal to:

- Min: 15.8%
- Width: 14.2%
- Mode: 9.1%
- Mean: 5.9%
- Median: 4.7%

Therefore, according to these findings, three variables were eliminated: *Width, Mean, Median.*

4.2 Dataset splitting and validation

In order to train the classification models, it was necessary to split the dataset in a part for training and one for testing. In particular it has been chosen to maintain a percentage of 67% for the training and 33% for the test. It has been preserved the same proportion of the two classes in both divisions (stratified split). Note that the split has been randomly generated but kept the same for all the following methods and for the different runs in order to make consistent the results among the different trials. (random_state = 42)

In order to optimize the classification algorithms and to test different hyper-parameters it was chosen to use a grid search implemented through K-fold cross validation. The training dataset was divided into 5-folds. The cross validation involves, in this case, 5 cycles in which each time 4 splits are chosen to be the training ones and a split is used to validate the chosen hyper-parameters. At the end of the process the average on the performance metrics will be calculated to evaluate the goodness of that particular set of hyper-parameters.

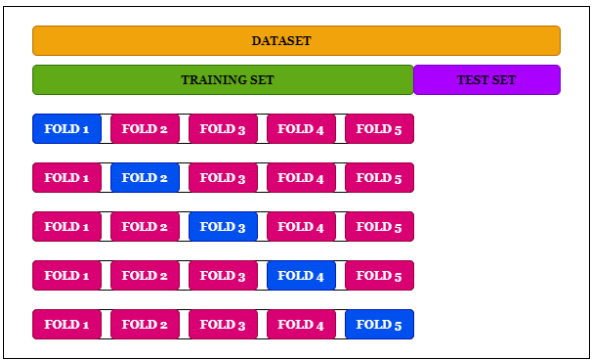


Figure 15: Cross Validation split

As well as for the test-set it is important that also the validation sets within the cross-validation are not affected by pre-processing activities in order to keep the verification phase as agnostic and independent as possible. For this reason resampling methods will be applied only on the training fold and not on the validation, as well as standardization and PCA.

4.3 Standardization

As can be seen from the analysis done earlier with the boxplots, regarding the distribution of data, it is clear that a normalization operation is necessary to make the data space more uniform and make sure that all have $mean = 0$ and $variance = 1$.
For each column:

$$x' = \frac{x - \mu_x}{\sigma_x} \tag{6}$$

where μ_x is the sample standard deviation of feature X.

The ultimate goal of performing standardization is to bring down all the features to a common scale without distorting the differences in the range of the values.

For many classification techniques it is important that features are zero-centered or have similar variances. For example, the *K-Nearest-Neighbor*, being based on the calculation of distances, is very susceptible to the variances of the data and could give more importance to certain attributes rather than others. The standardization is carried out only on the values of the samples of the training dataset and is then applied on the basis of the mean and variance to the other test samples as well. It is an important process to keep data transformation formally correct and not spoil them with information not available at the time of training.

4.4 PCA

Dimensionality reduction is a technique used to reduce the size of large datasets by transforming a large set of variables into a smaller one that still retains most of the information of the large one. One way to do this is to use the PCA algorithm (Principal Component Analysis).

Reducing the number of variables comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Smaller data sets are easier to explore and visualize and this make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.

So overall we can say that dimensionality reduction is useful for:

- **reduce the computational cost** that negatively affect algorithm
- a better **generalization** and **avoid overfitting** or **curse of dimensionality**
- data easier to be **visualized**

PCA wants to find the optimal value of the coefficient of each feature that maximizes the variance of the linear combination of all features.
More formally, the principal component exploiting the maximum variance, corresponding to the **first** principal component, is calculated as follow:

$$\begin{gathered} X \in \mathbb{R}^{n \times d},\ dataset\, centered\, in\, zero \\ \Sigma, sample\, covariance\, matrix \\ find\, \vec{z_1} = a_1\vec{e_1} + ... + a_d\vec{e_d}\ subject\, to \\ \vec{z_1} = argmax_{\vec{z_1}}\, \vec{z_1}^T \Sigma \vec{z_1}\ s.t. : ||\vec{z_1}|| = 1 \end{gathered} \tag{7}$$

The above maximization problem can be easily computed through the eigenvectors of the covariance matrix as follow:

1. Data have been normalized. We compute the covariance matrix Σ to spot covariance and variance of all the features of the dataset:

$$var(f1) = \frac{1}{n-1}\sum_{i=1}^{n}(f1_i - \overline{f1})^2 \tag{8}$$

$$cov(f1, f2) = \frac{1}{n-1}\sum_{i=1}^{n}(f1_i - \overline{f1})(f2_i - \overline{f2}) \tag{9}$$

Cov (x,x)	**Cov(x,y)**	**Cov(x,z)**
Cov(y,x)	**Cov(y,y)**	**Cov(y,z)**
Cov(z,x)	**Cov(z,y)**	**Cov(z,z)**

Figure 16: Covariance matrix for 3 features

2. We calculate the eigenvalues of the covariance matrix by setting:

$$det|\Sigma - \lambda I| = 0 \tag{10}$$

   Solving this equation of $d - th$ degree, as much as the number of features, we get all the eigenvalues.

3. We obtain the first eigenvector $PC1$ by solving the following equation, where λ_1 is the largest eigenvalue and Σ is the covariance matrix. The first eigenvector found also represents the *first principal component*.

$$\Sigma PC1 = \lambda_1 PC1 \tag{11}$$

   In the same way we also get the other eigenvectors which will all be orthogonal.

4. Multiplying the original matrix with the two principal eigenvectors we transform the initial space on the new two principal components PC1 and PC2. The eigenvalues corresponding to the eigenvectors found are exploitative of the variance represented by these new components.

By calculating the variance explained by each principal component we can compare it with the total variance of the dataset and select only a subset of PCs that reduces dimensionality but retains most of the information and variance. In the plot we can see the single variance explained by each principal component and the cumulative explained variance curve. We usually choose a threshold equal to 90%. In our case, wanting to maintain 90% of the variance, only 11 Principal Components will be needed. The first three components alone already cover 55% of the total variance.

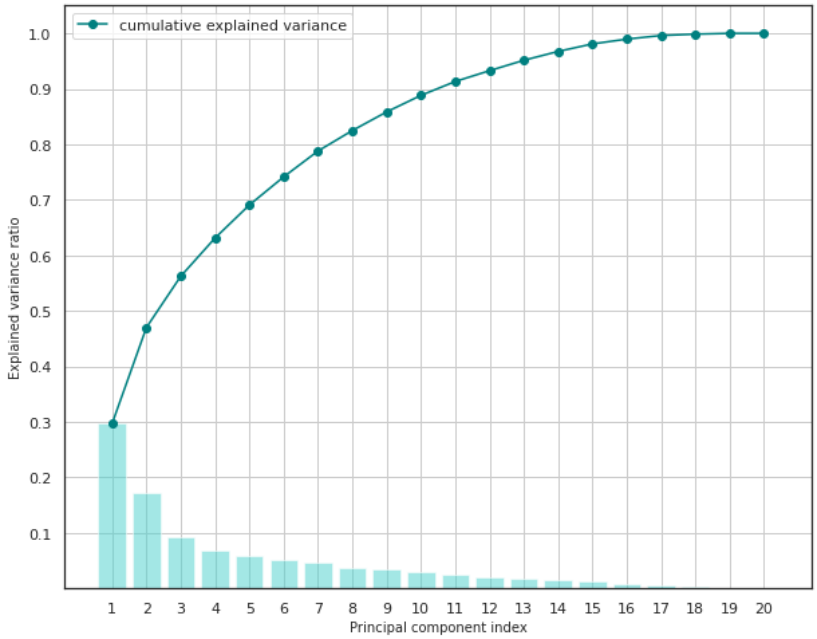


Figure 17: Singular and cumulative variance explained by Principal Components

4.5 Resampling

As anticipated, the analyzed dataset shows a strong imbalance for the class related to pathological fetuses. For this reason it is necessary to try to balance the classes by resampling.
The resampling operations have been implemented always and only on the training dataset and not on the test, in order not to compromise the reality of the data and because the algorithm must remain blind to the possible data that will be given to him as testing data.

4.5.1 Random Oversampling

The easiest method to increase the minority class samples is by random oversampling.
Random oversampling involves randomly selecting with replacement examples from the minority class and adding them to the training dataset. This means that examples from the minority class can be chosen and added to the new "more balanced" training dataset multiple times; they are selected from the original training dataset, added to the new training dataset, and then returned or "replaced" in the original dataset, allowing them to be selected again. This process is carried on until an equal number of samples is obtained for both classes.

However, the random oversampling may increase the likelihood of occurring overfitting, since it makes exact copies of the minority class examples.

4.5.2 SMOTE

Another useful technique for doing data augmentation is to create synthetic samples using the Synthetic Minority Oversampling Technique (SMOTE). Also in this case the objective is to balance the classes and obtain an equal number of samples for the normal and pathological class.

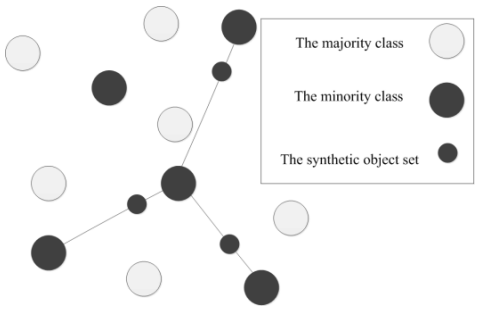


Figure 18: SMOTE graphical schema

Considering a sample x_i of the minority class and one of its k neighbors $x_{i,k}$ of the same class, a new sample is generated that stations on the segment that connects x_i with $x_{i,k}$.
For example:

$$x_{new} = x_i + \lambda(x_i - x_{i,k}), \lambda \in (0, 1) \tag{12}$$

The approach is effective because new synthetic examples from the minority class are created that are plausible and relatively close in feature space to existing examples from the minority class.
A general downside of the approach is that synthetic examples are created without considering the majority class, possibly resulting in ambiguous samples if there is a strong overlap for the classes.

4.5.3 Undersampling with centroids

This third method under-samples the majority class by replacing a cluster of majority samples by the cluster centroid of a K-Means algorithm. This algorithm keeps N majority samples by fitting the K-Means algorithm with N cluster to the majority class and using the coordinates of the N cluster centroids as the new majority samples.
The newly generated set is synthesized with the centroids of the K-means method instead of the original samples. It transforms the majority class, the minority class preserves itself.

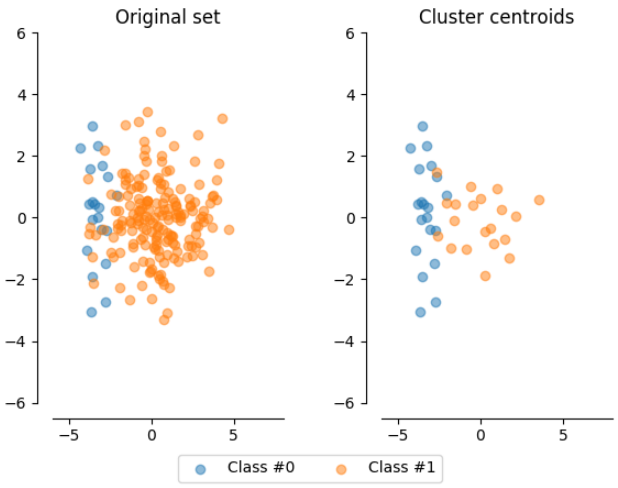


Figure 19: Undersampling with centroids graphical schema

4.5.4 SMOTEENN

The last technique analyzed is SMOTEENN which combines **SMOTE** oversampling and **Edited Nearest Neighbors** cleaning technique.

Frequently, class clusters are not well defined since some majority class examples might invading the minority class space. The opposite can also be true because interpolating minority class examples can expand the minority class clusters, introducing artificial minority class examples too deeply in the majority class space. Inducing a classifier under such a situation can lead to overfitting.

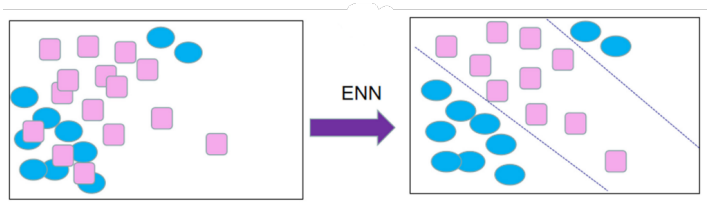


Figure 20: Cumulative variance explained by Principal Components

ENN is used in order to create better-defined class clusters and clean data by removing samples close to the decision boundary. ENN is used to remove examples from both classes. Thus, any example that is misclassified is removed from the training set. The algorithm of ENN can be explained as follows.

1. Determine K, as the number of nearest neighbors.
2. Find the K-nearest neighbor of the observation among the other observations in the dataset, then return the majority class from the K-nearest neighbor.
3. If the class of the observation and the majority class from the observation's K-nearest neighbor is different, then the observation and its K-nearest neighbor are deleted from the dataset.
4. Repeat step 2 and 3 until the proportion of each class is the same.

The following result was obtained by employing ENN to eliminate noise samples caused by the process of SMOTE.

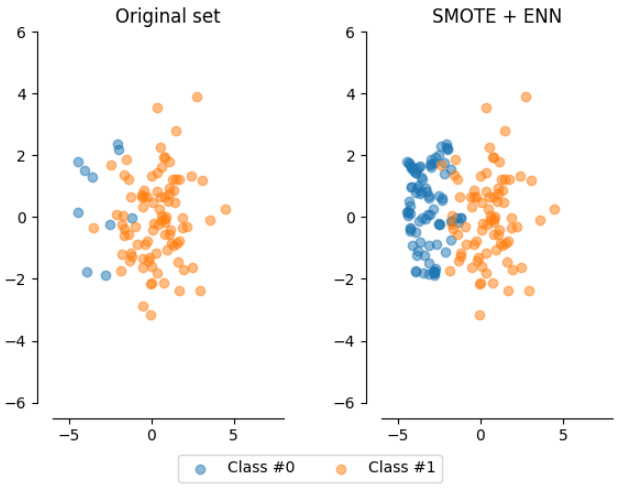


Figure 21: SMOTEENN resampling technique graphical representation

5 Metrics

In order to effectively evaluate the tuning of the different parameters and the goodness of the different classification methods used, it is necessary to introduce some evaluation metrics:

1. **Accuracy** $= \frac{\text{correctly classified samples}}{\text{total number of samples tested}} = \frac{TP+TN}{TP+TN+FP+FN}$
2. **Precision** $= \frac{\text{samples correctly assigned to positive}}{\text{total number of samples assigned to positive}} = \frac{TP}{TP+FP}$ which measure how often the model is correct when it predicts positive.
3. **Recall** $= \frac{\text{samples correctly assigned to positive}}{\text{total number of samples actually belonging to positive}} = \frac{TP}{TP+FN}$ which measure how often the model predict positive samples among all true positive sample.
4. **F1 score** $= \frac{2*\text{precision}*\text{recall}}{\text{precision}+\text{recall}}$, which is the harmonic mean of precision and recall and it increases with both precision and recall.

Accuracy is a popular metric. However, in the case of unbalanced datasets, especially those containing medical diagnoses, it is not always the best choice. Recall is often a much-appreciated metric in the medical field because if it is high it indicates a high ability to identify samples as pathological. However, if we had only the recall indication, we would not know if the model actually predicts correctly or if there is a particular disproportion in classifying the samples as pathological. For this reason the best solution is to use F1-score as the main reference metric, which incorporates both information and gives us a more reliable global view.

5.1 Confusion matrix

A good way to analyze the performance can be through the confusion matrix, within which we find the information for all 4 reference values (TruePositive, TrueNegative, FalsePositive, FalseNegative) and from which it is very easy to extrapolate the data to calculate the reference metrics.

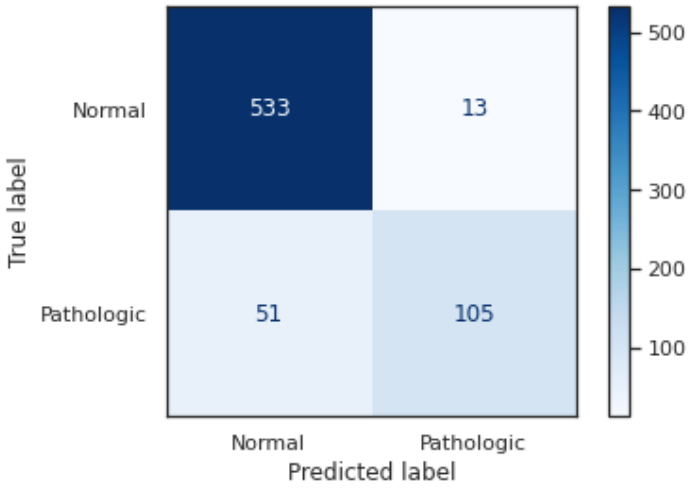


Figure 22: Example of the confusion matrix of KNN before any type of resampling or dimensionality reduction

6 Classification Methods

6.1 Random Forest

The first classification method that has been used is the *Random Forest*. To explain the structure of the Random Forest we start by outlining the single component that is the *Decision Tree*.
The original idea behind is to find a particular space partition such that the misclassification error is minimized. However, there are too many possible solution to this problem, that is NP-hard. For this reason we use a local heuristic to make greedy best choice locally, for each node, through the measure of impurity
With a Decision Tree the classes are predicted through simple decision rules inferred from data.

At the first step a new feature creates the root, which branches will lead to a new node composed of a new feature or, if the combination of features is sufficient to delineate the classification, a new leaf. Nodes are chosen by taking the feature that always best divides the classification space. Hardly a single feature is able to divide the space with 100% precision. For this reason, features have an *impurity level* that defines how well they fit for the purpose. The impurity level is calculated using the ***GINI Index*** or the ***Entropy Index***.

$$G(t) = \sum_{k=1}^{C} p_t(k)(1 - p_t(k)) = 1 - \sum_{k=1}^{C} p_t(k)^2 \tag{13}$$

$$E(t) = -\sum_{k=1}^{C} p_t(k) log(p_t(k)) \tag{14}$$

where $p_t(k)$ is the frequency of the class k at the node t, with C the total number of classes.
The impurity is more negligible the lower the gini index is. For this reason those features with the lowest possible impurity value are chosen, at each step, as nodes.

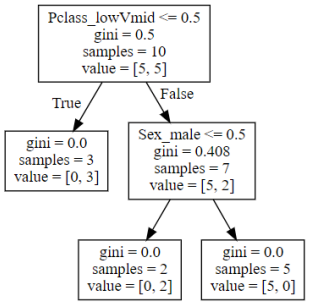


Figure 23: Decision Tree splitting example

It is clear that in the Decision Tree the order of the features is very important and with only one Decision Tree the risk of overfitting is very high because the variability in the model is low.
For this reason it is chosen, as a classification model, a Random Forest, composed of several Decision Trees. The following are the steps to build a Random Forest Classifier:

1. Creation of a **Bootstrapped** Dataset. A dataset with the same size of the original is created, taking the data in a random way and with replacement, with the possibility of having several copies of the same data.

2. Creation of the Decision Tree using the bootstrapped dataset and only a **random subset of fixed size features** (columns) at each step, for each node.

3. At this point the process is **repeated** from step 1., until all the desired trees have been created.

4. Finally, the classification is done by **majority voting** of all the results obtained from each tree.

Bootstrapping the data plus using the aggregate to make a decision is called **Bagging**. The boostrapping technique and considering only a subset of variables at each step results in a wide variety of trees, with a high variance. The variety is what makes Random Forest more effective and robust than individual decision trees, stronger because uncorrelated but at the same time less interpretable.

Specifically, in this analysis, hyperparameters have been tuned as follows:

```
"criterion":["gini", "entropy"],
 #the cardinality of the set of random features extracted at each split
"max_features":["auto", "sqrt", "log2"],
"n_estimators":[10, 50, 100, 200],
"max_depth":[None,5,10,20,25]
```

It was analyzed, through a summary graph, the *f1-score* performance, compared to all possible pipelines chosen. More detail can be found in the appendix A.1 where the confusion matrix is located.

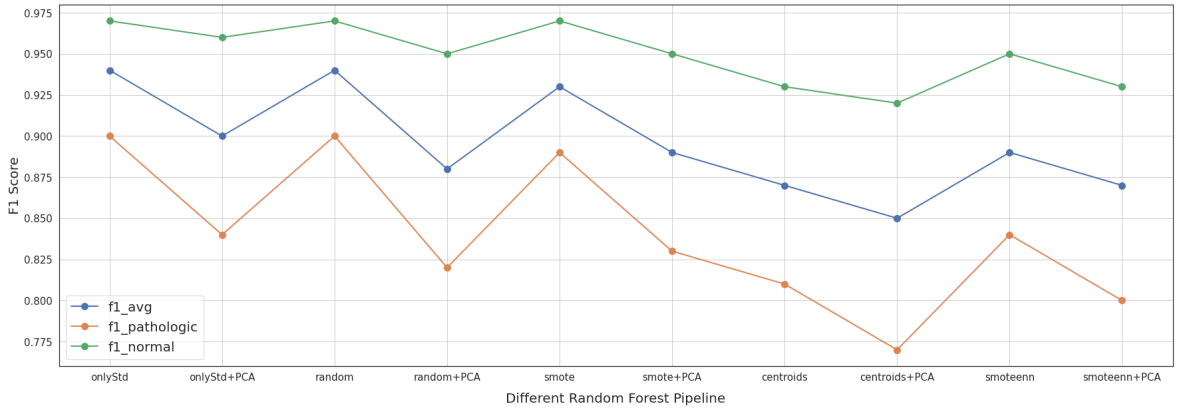


Figure 24: Different Random Forest Pipeline

It is possible to notice how the results are worse and worse using PCA. This is easily explained by the fact that PCA aims to reduce the feature space and this can be counterproductive for classification using decision trees that seek to achieve maximum class separation. Below, in tabular form, is a summary of other metrics and the best hyperparamenters chosen by the *Grid Search* for each pipeline.

	pipeline	f1_avg	f1_pathologic	f1_normal	accuracy	criterion	max_depth	max_features	n_estimators
0	onlyStd	0.94	0.90	0.97	0.96	entropy	None	sqrt	100
1	onlyStd+PCA	0.90	0.84	0.96	0.93	gini	None	log2	50
2	random	0.94	0.90	0.97	0.96	entropy	10	sqrt	50
3	random+PCA	0.88	0.82	0.95	0.92	entropy	10	log2	100
4	smote	0.93	0.89	0.97	0.95	gini	25	auto	50
5	smote+PCA	0.89	0.83	0.95	0.92	entropy	20	auto	100
6	centroids	0.87	0.81	0.93	0.90	entropy	25	sqrt	10
7	centroids+PCA	0.85	0.77	0.92	0.88	entropy	20	log2	100
8	smoteenn	0.89	0.84	0.95	0.92	entropy	10	auto	100
9	smoteenn+PCA	0.87	0.80	0.93	0.90	entropy	25	log2	200

Figure 25: Metrics and tuning results for the Random Forest Classifier

6.2 KNN (K-Nearest Neighbours)

The second classification method used is the KNN or *K-Nearest Neighbours* Classifier. The basic idea around KNN is that data points that are closer in the feature space should also share similar properties and so similar labels. This method classifies the data point x according to the most frequently occurring class labels of his K-Neighbours.
There are several parameters, with different implication, that need to be tuned.

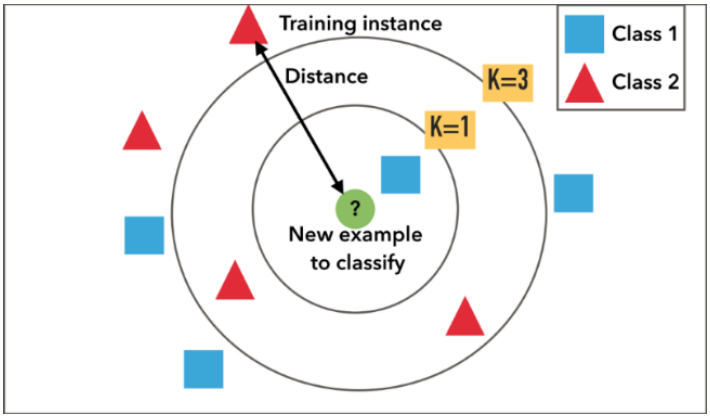


Figure 26: KNN graphical representation

- The first parameter is K the number of neighbors. A very low number of neighbors (such as 1,2,3) could be very sensitive to outliers and not capture enough information. Vice versa, choosing a too high K risks to be counterproductive and to create a too general model that doesn't divide well the two classes.

- The second relevant parameter is the formula with which the **distance** of the possible neighbors (from the selected data point) is computed.
  In the specific case it has been chosen to use the algorithm *KNeighborsClassifier* from the library *sklearn*, which provides the possibility to choose between 3 different distances:

  1. Euclidean Distance: $d(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$
  2. Manhattan Distance: $d(a,b) = \sum_{i=1}^{n}|a_i - b_i|$
  3. Chebyshev Distance: $d(a,b) = max_{i=1}^{n}|a_i - b_i|$

  where a is the possible neighbour, b our data point and n the number of different dimensions (features). All of these distances can be generalized varying p within the *Minkowski* distance:

$$d_p(a,b) = (\sum_{i=1}^{n}|a_i - b_i|^p)^{1/p} \tag{15}$$

- The third parameter is about adding a **weight** in the final majority voting calculation, creating a *Weighted KNN*. The basic nearest neighbors classification uses uniform weights: that is, the value assigned to a query point is computed from a simple majority vote of the K nearest neighbors. Under some circumstances, it is better to weight the neighbors such that nearer neighbors contribute more to the fit. The closer the neighbor is to the data point to be classified, the more weight this neighbor will have in determining the outcome of the vote.

Specifically, in this analysis, hyperparameters have been tuned as follows:

```
"n_neighbors":[200, 100, 50, 30, 20, 10, 5, 3],
"weights":['uniform','distance'],
"p":[1,2,inf]
```

It was analyzed how the *f1-score* changes as the number of neighbors varies and the differences between the canonical and weighted algorithm. It can be seen as KNN performs better with lower value of K and that in the weighted form the increasing of K hurts less then for the uniform variant. That is because in the weighted one further points are given less importance for the classification. Below are the results.

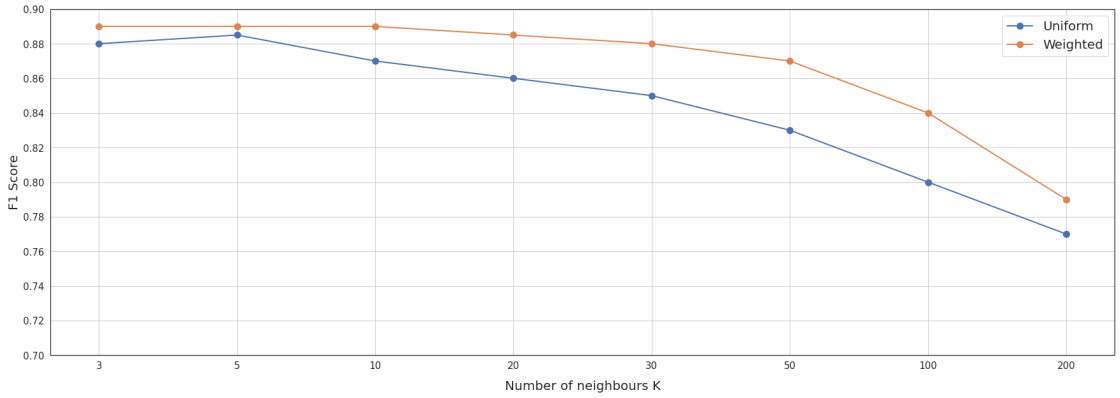


Figure 27: Comparison between different K-values and Weighted or Standard KNN

Finally, here is a summary graph of the *f1-score* performance, compared to all possible pipelines chosen. More detail can be found in the appendix A.2 where the confusion matrix is located.

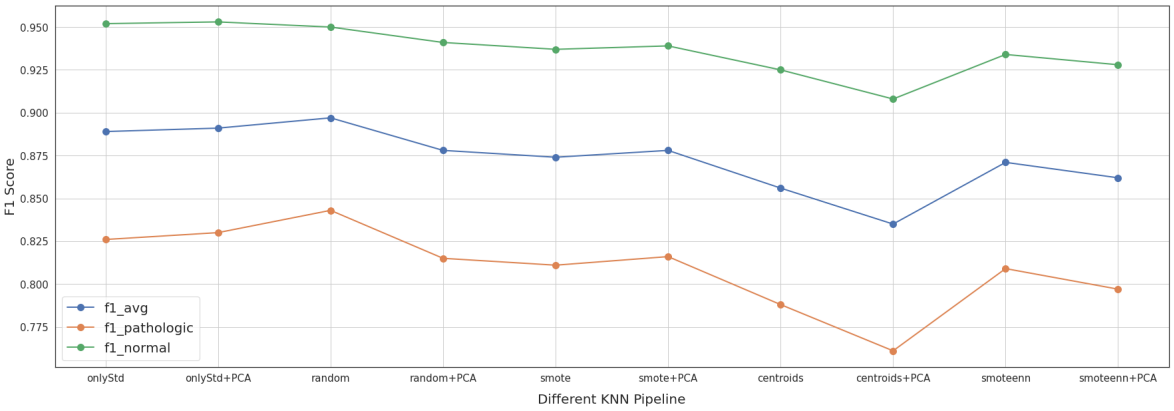


Figure 28: Performances of different KNN Pipeline

The trend of the KNN is much more constant than the one of the Random Forest and, as expected, the application of PCA is more convenient since, if we increase the dimensionality, the concept of KNN vanishes and it suffers of curse of dimensionality, with PCA we are able to reduce this effect and reduce the dimension. We obtain good results in increasing the recall using SMOTE and SMOTEENN, which increase the point-density, particularly convenient for the KNN application.

Below, in tabular form, is a summary of other metrics and the best hyperparamenters chosen by the *Grid Search* for each pipeline.

	pipeline	f1_avg	f1_pathologic	f1_normal	accuracy	n_neighbours	distance_type	weights
0	onlyStd	0.889	0.826	0.952	0.925	5	1	distance
1	onlyStd+PCA	0.891	0.830	0.953	0.926	3	2	distance
2	random	0.897	0.843	0.950	0.925	3	1	distance
3	random+PCA	0.878	0.815	0.941	0.910	3	2	distance
4	smote	0.874	0.811	0.937	0.906	10	1	uniform
5	smote+PCA	0.878	0.816	0.939	0.909	5	2	distance
6	centroids	0.856	0.788	0.925	0.889	3	1	distance
7	centroids+PCA	0.835	0.761	0.908	0.868	5	1	distance
8	smoteenn	0.871	0.809	0.934	0.902	3	1	distance
9	smoteenn+PCA	0.862	0.797	0.928	0.893	3	1	distance

Figure 29: Metrics and tuning results for the KNN Classifier

6.3 SVM (Support Vector Machine)

SVM is a parametric linear classification algorithm that aims at separating two classes through an **hyperplane**. The goal is to find this particular hyperplane that maximizes the margin (the distance) between the **support vectors**, i.e. the points, of each class, closest to the hyperplane. Once the hyperplane is outlined, the prediction rule is simply to assign the point to the side where it is located.

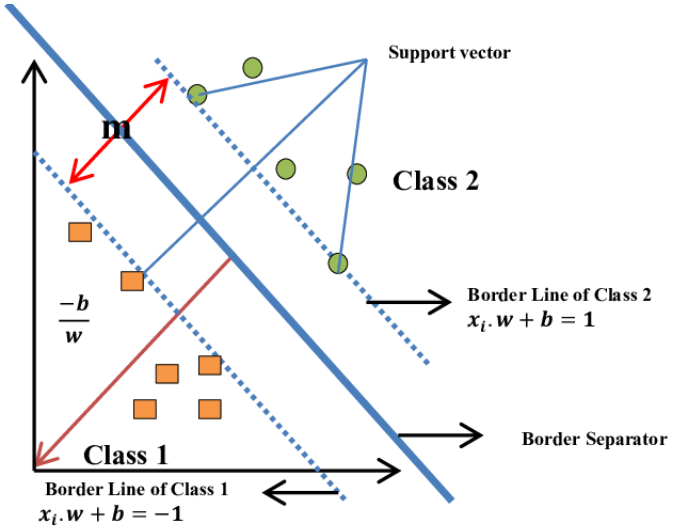


Figure 30: Graphical representation of the hyperplane of a Support Vector Machine

More in details, we are trying to define the hyperplane $w^T x + b = 0$ such that the distance between that hyperplane and the support vectors x_i is maximized.
So, supposing we have a negative class and a positive one, we would like to have that $w^T x_+ + b >= 1$ and $w^T x_- + b <= -1$. Those equations could be generalized to:

$$y_i(w^T x_i + b) \geq 1, \forall i \tag{16}$$

The margin, the distance, between the two border line, is equal to $width = \frac{2}{||w||}$. The goal is to **maximize** that distance, that's equal to say:

$$min \frac{1}{2}||w||^2 \tag{17}$$

The decision function on a binary label is then applied as $y_{pred} = sign(w^T x + b)$.
The given problem can be also solved through the **dual formulation** problem, where at the optimum w is a linear combination of the points called support vectors :

$$\begin{aligned} max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ s.t. \sum_i \alpha_i y_i = 0, \forall i \\ \alpha_i \geq 0, \forall i \end{aligned} \tag{18}$$

6.3.1 Soft Margin SVM

The previous model, called *Hard Margin SVM*, works well if the space is linearly separable, otherwise no feasible solution would be found.
For this reason we can introduce the so-called Soft Margin SVM that relax the constraint ad allow some sort of missclassification. To do so we introduce a slack variable ξ_i and an hyperparameter C.

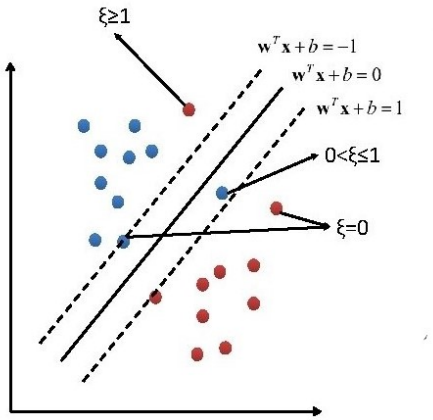


Figure 31: Graphical representation of the Soft Margin SVM

This last parameter controls the amount of missclassification since an higher C will force training points to be more correctly classified, while a lower C leads to a larger margin and more missclassifications. The previous conditions then becomes:

$$\begin{aligned} min_{w,\xi} \frac{1}{2} ||w||^2 + C \sum_i^n \xi_i \\ s.t : y_i(w^T x_i + b) \geq 1 - \xi_i, \forall_i \\ \xi_i \geq 0, \forall_i \end{aligned} \tag{19}$$

Again, this can be solved by solving the dual Lagrangian problem:

$$\begin{aligned} max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ s.t. \sum_i \alpha_i y_i = 0, \forall i \\ 0 \leq \alpha_i \leq C, \forall i \end{aligned} \tag{20}$$

It was analyzed how the *f1-score* changes as the value of C varies. Since a lower C encourage a larger margin, with very small values we can see how the classifier is not able to distinguish one class from another. Using the two values 0.0001 and 0.001 the F1 score is very low and, even if the average is around the 44%, all the correct predictions are about the normal class and none of the pathological patient are recognized. Things get better if C increase and, over the value of 10, we get really good performances. Below are the results.

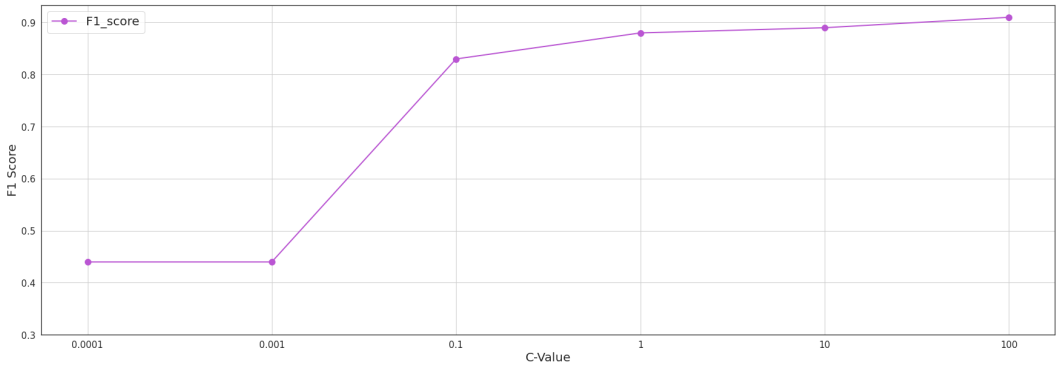


Figure 32: Performance tuning C-parameter

6.3.2 Kernel trick

When we have points that are not linearly separable we can try to see them in another dimension, by a non linear function. Solving the dual problem we can find $w = \sum_i^n \alpha_i y_i x_i$ as a linear combination of the training data. Starting from that is possible to generalize SVM to a non-linear separable classes. The training point are transformed as $x_i \in \mathbb{R}^d \rightarrow \varphi(x_i) \in \mathbb{R}^{d'}, d' > d$. In this way the algorithm can be kernelized and the scalar product inside the dual problem is replaced with a kernel function that represents the scalar product in another space. No matter what the higher-dimensional space is, all that it's needed is a function able to compute the scalar products in the new transformed space.

This *Kernel function* k is defined as $k : X \times X \rightarrow \mathbb{R}$ and it is:

- symmetric $(k(u, v) = k(v, u) \forall u, v \in X)$
- the Gram Matrix K is positive semidefinite $(K \in R^{x \times x} where K_{ij} = k(x_i, x_j) \forall x \in X$

If a function satisfies these conditions, for the **Mercer's theorem**, there exist an Hilbert Space S and a transformation φ that maps our data into a space in which the kernel represent the scalar product on the dual formulation:

$$k(x, x') := \varphi(x)^T \varphi(x') \tag{21}$$

In this way we can work in the our space, without mapping data in a new one, replacing the scalar product with the kernel function (18). We can choose between different type of Kernels. The more common and the one used here are:

- the **Linear kernel**: $K(x, x') = x^T x$'
- the **Gaussian RBF kernel**: $K(x, x') = e^{-\gamma ||x - x'||^2}, \gamma > 0$ where γ is a parameter of how the decision boundaries are affected by the points closer to it. If *gamma* is high the decision boundary are more wiggled, while if it is low it means that many different points, also far from that, influence the decision boundary.

- the **Polynomial kernel**: $K(x, x') = (x^T x' + c)^d$ where d is the polynomial degree and c a constant usually fixed to 1.

Specifically, in this analysis, hyperparameters have been tuned as follows:

```
"C": [100, 0.1, 1, 10, 0.001, 0.0001],
"kernel": ['rbf', 'poly','linear']
```

Finally, here is a summary graph of the *f1-score* performance, compared to all possible pipelines chosen. More detail can be found in the appendix A.3 where the confusion matrix is located. Also in this case the trend is much more stable than in the Random Forest, even if PCA does not give us good improvements. This could be due to the contrast between a desirable higher dimensional space by SVM and the dimensionality reduction given by PCA.

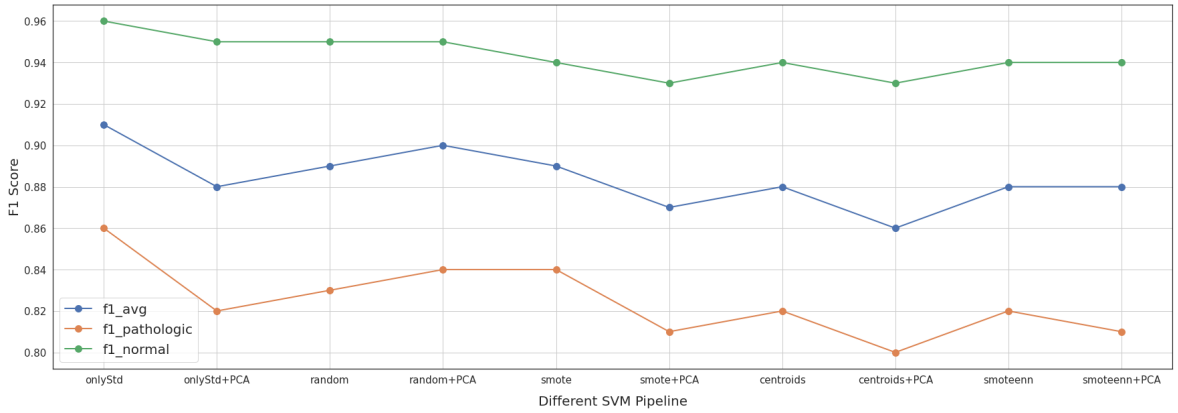


Figure 33: Performance of different SVM pipelines

Below, in tabular form, is a summary of other metrics and the best hyperparamenters chosen by the *Grid Search* for each pipeline.

	pipeline	f1_avg	f1_pathologic	f1_normal	accuracy	C	gamma	kernel
0	onlyStd	0.91	0.86	0.96	0.94	100	0.01	rbf
1	onlyStd+PCA	0.88	0.82	0.95	0.92	100	0.01	rbf
2	random	0.89	0.83	0.95	0.92	100	0.01	rbf
3	random+PCA	0.90	0.84	0.95	0.92	100	0.01	poly
4	smote	0.89	0.84	0.94	0.92	100	0.01	rbf
5	smote+PCA	0.87	0.81	0.93	0.90	100	0.01	rbf
6	centroids	0.88	0.82	0.94	0.90	100	0.01	rbf
7	centroids+PCA	0.86	0.80	0.93	0.89	100	0.01	rbf
8	smoteenn	0.88	0.82	0.94	0.91	100	0.01	rbf
9	smoteenn+PCA	0.88	0.81	0.94	0.91	100	0.01	poly

Figure 34: Metrics and tuning results for the SVM Classifier

7 Conclusions

In conclusion, it can be seen that the best performing classifier was the just standardized Random Forest with an average *f1 score* of 0.94 and that the PCA algorithm is not always improving the classifier, exept for the KNN algorithm. This may be also due to the size of the database, with many

values, many entries, compared to the number of features, which therefore did not require a particular reduction of the feature space. Moreover, the accuracy and f1 score performances are overall quite high, thanks to the standardization and the feature selection and so they are difficult to be improved. The application of resampling strategies were not always beneficial, probably because of the distribution of features in the space and especially for algorithms that are more prone to overfitting, such as Random Forest.

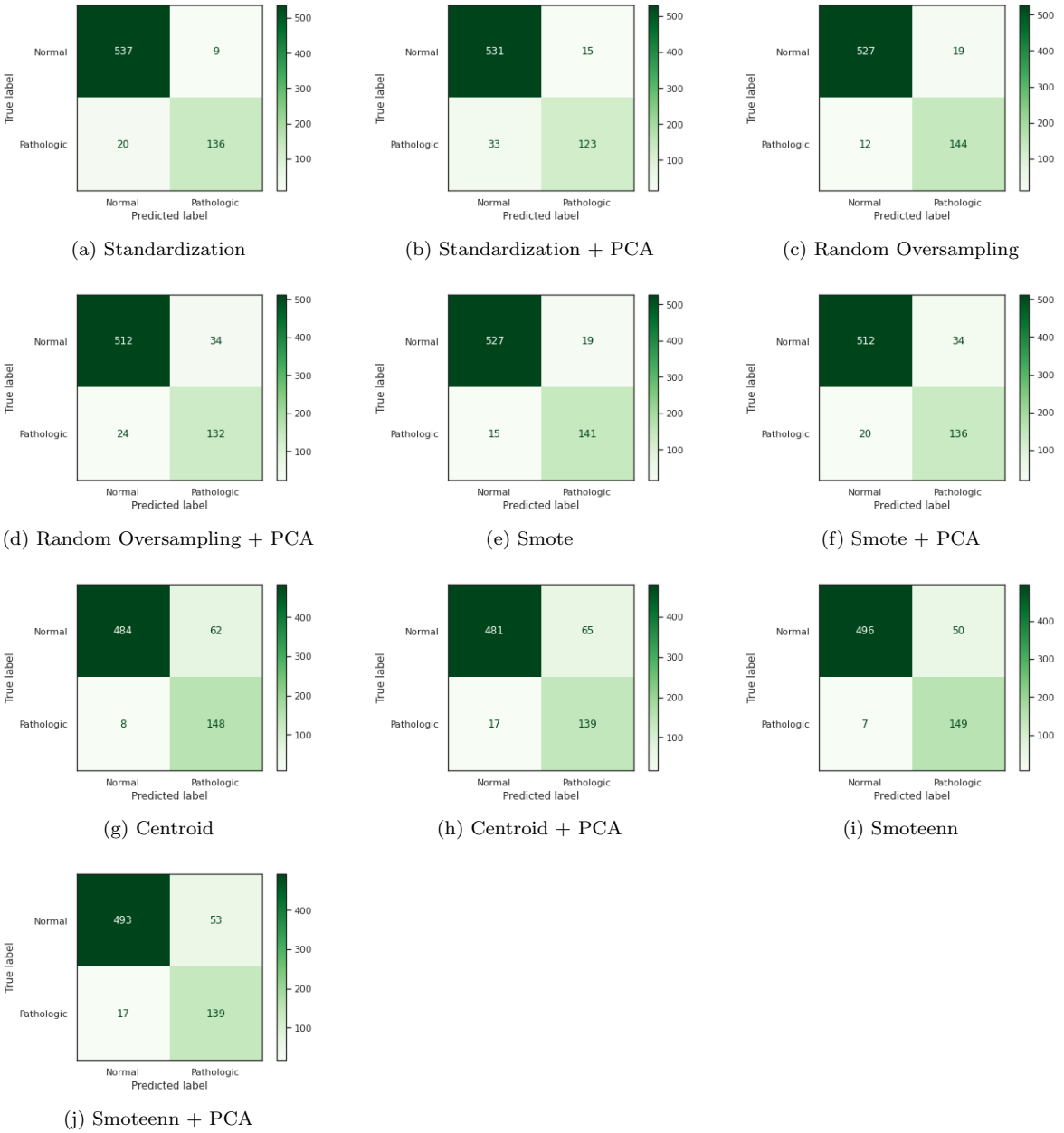
8 Bibliography and sitography

1. https://archive.ics.uci.edu/ml/datasets/cardiotocography → dataset
2. https://thewomens.r.worldssl.net/images/uploads/downloadable-records/clinical-guidelines/ctg-interpretation-and-response_280720.pdf → useful source for feature interpretation
3. *A Study of the Behavior of Several Methods for BalancingMachine Learning Training.* Data-Gustavo E. A. P. A. BatistaRonaldo C. PratiMaria Carolina Monard
4. *Understanding machine learning: From theory to algorithms.* Shalev-Shwartz, Shai, and Shai Ben-David

Appendices

A Confusion matrix

A.1 Random Forest (24) Confusion Matrix



(a) Standardization (b) Standardization + PCA (c) Random Oversampling

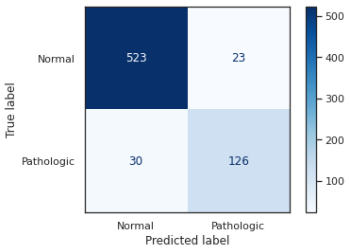
(d) Random Oversampling + PCA (e) Smote (f) Smote + PCA

(g) Centroid (h) Centroid + PCA (i) Smoteenn

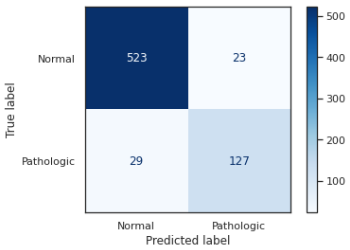
(j) Smoteenn + PCA

Figure 35: Confusion Matrix of Random Forest

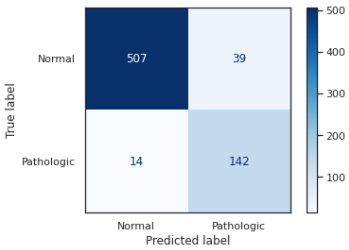
A.2 KNN (28) Confusion Matrix



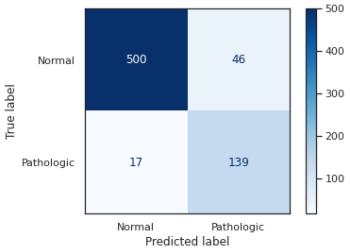
(a) Standardization



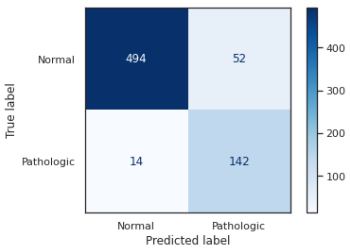
(b) Standardization + PCA



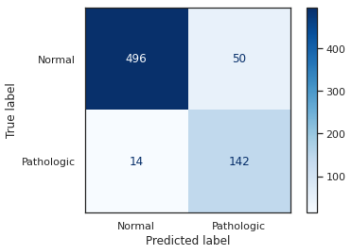
(c) Random Oversampling



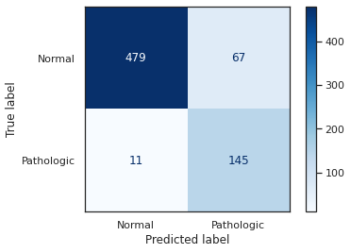
(d) Random Oversampling + PCA



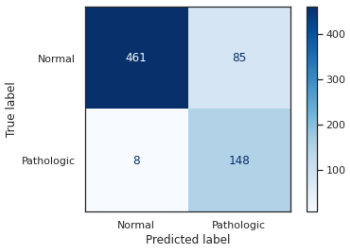
(e) Smote



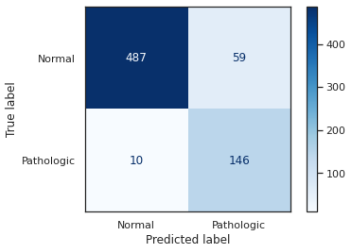
(f) Smote + PCA



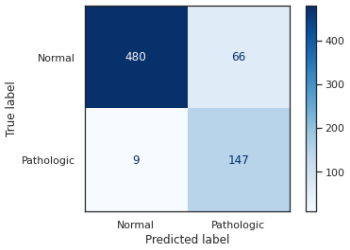
(g) Centroid



(h) Centroid + PCA



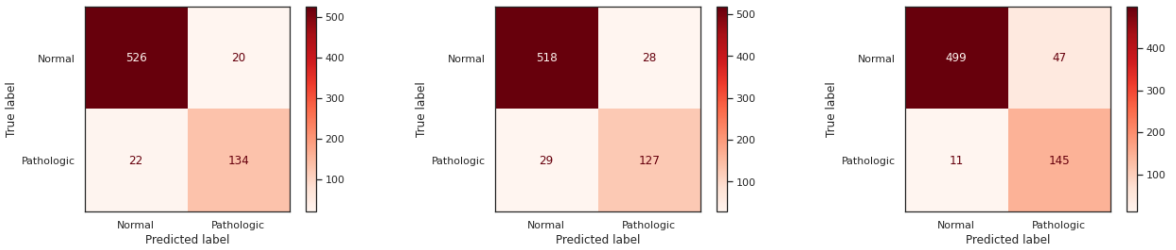
(i) Smoteenn



(j) Smoteenn + PCA

Figure 36: Confusion Matrix of KNN

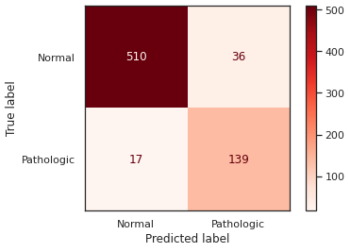
A.3 SVM (33) Confusion Matrix



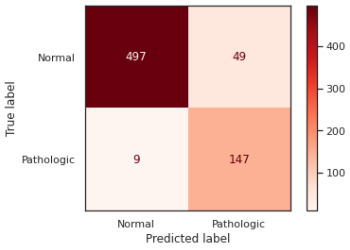
(a) Standardization

(b) Standardization + PCA

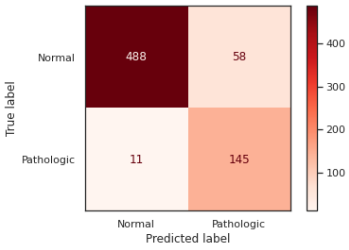
(c) Random Oversampling



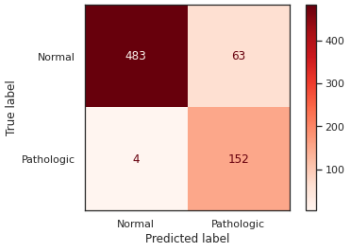
(d) Random Oversampling + PCA



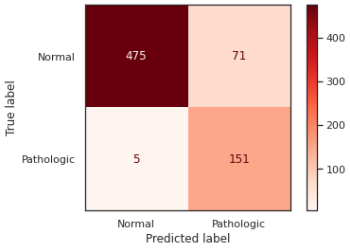
(e) Smote



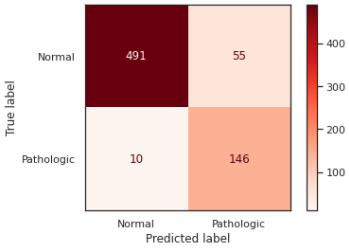
(f) Smote + PCA



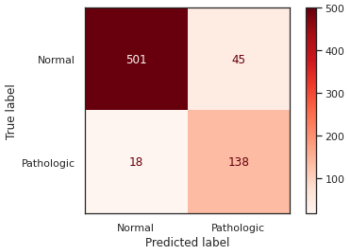
(g) Centroid



(h) Centroid + PCA



(i) Smoteenn



(j) Smoteenn + PCA

Figure 37: Confusion Matrix of SVM