Politecnico di Torino

Mathematics in Machine Learning

Fetal Cardiotocograms dataset analysis

Candidate:

Edoardo Fantolino

Professors:

Francesco Vaccarino, Mauro Gasparini

December 15, 2021

Contents

Introduction

Cardiotocography (CTG) is a medical technology that allow to establish the condition of the fetus. More precisely, CTG is a continuous recording of the fetal heart rate obtained via an ultradound transducer placed on the mother's abdomen. It is an extremely usefull technique especially for pregnancies with increased risk of complications.

This project consist in analysing fetal cardiotocograms using powerful techniques of Machine Learning (ML).

The main steps of the assignment will be:

- Explore the dataset focusing on the distribution of the features and the possible unbalance issues related to the target variable
- Clean and prepare the data to create the most suitable conditions for further analysis
- Apply and analyze different ML algorithms to the dataset and keep track of the performances using valuable and significant measures

You will probably find the code at: https://github.com/edoardofantolino/

1 Dataset Overview and Description

The dataset consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians-

There are 2126 total records. This exams were automatically processed and the respective diagnostic features measured. Moreover, the CTG's were classified by three expert obstetricians and they assigned a label to each of them. We have at our disposal morphologic pattern but also fetal state (N, S, P) and so we could performa a multi-class analysis or a 3-class analysis respectively. We will reduce the 3-class case in a binary situation.

In the following, there are the descriptions of some of the main attributes taken into consideration for the analysis:

LB - FHR baseline (beats per minute)

AC - Number of accelerations per second

FM - Number of fetal movements per second

UC - Number of uterine contractions per second

ASTV - percentage of time with abnormal short term variability

MSTV - mean value of short term variability

Min - minimum of FHR histogram

Max - Maximum of FHR histogram

Median - histogram median

Variance - histogram variance

NSP - fetal state class code (N=normal; S=suspect; P=pathologic)

The dataset and a more detailed description are available at:

http://archive.ics.uci.edu/ml/datasets/Cardiotocography

2 Data Exploration

The Data Exploration step is essential in order to get an overall idea of what we are going to manage. For example we can compute some simple statistics as the total number of records that is 2126 and the total number of features that is equal to 25.

Then we can compute useful information as the number of NaN (Not a Number) values present in our dataset. This values correspond to missing data. Fortunately, in this case there are not such values and we don't have to apply techniques to avoid the possible issues.

2.1 Balance

One of the main aspect that we have to check is if the dataset is balanced or unbalanced. A balanced dataset is a set of records that contains an equal number of samples for each class. Instead, we will have an unbalanced dataset if the frequency of a given class overwhelm the others. To create a binary case, we merge the class 'suspect' and 'pathologic' is necessary.

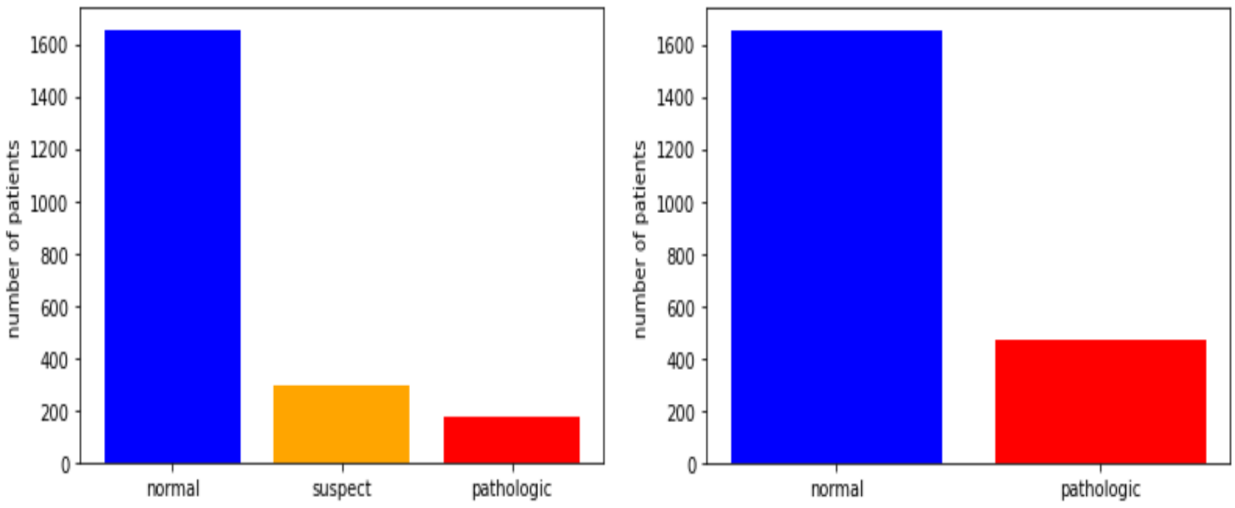

Figure 1: The distribution of the different classes. On the right, there is the binary case.

2.2 Feature distribution

In the following you can see some violin plots that represent the distribution of some of the attribute of the dataset. We divided the case of normal label and pathologic label, It is evident that some of the distribution are quite similar while others have a different shape,

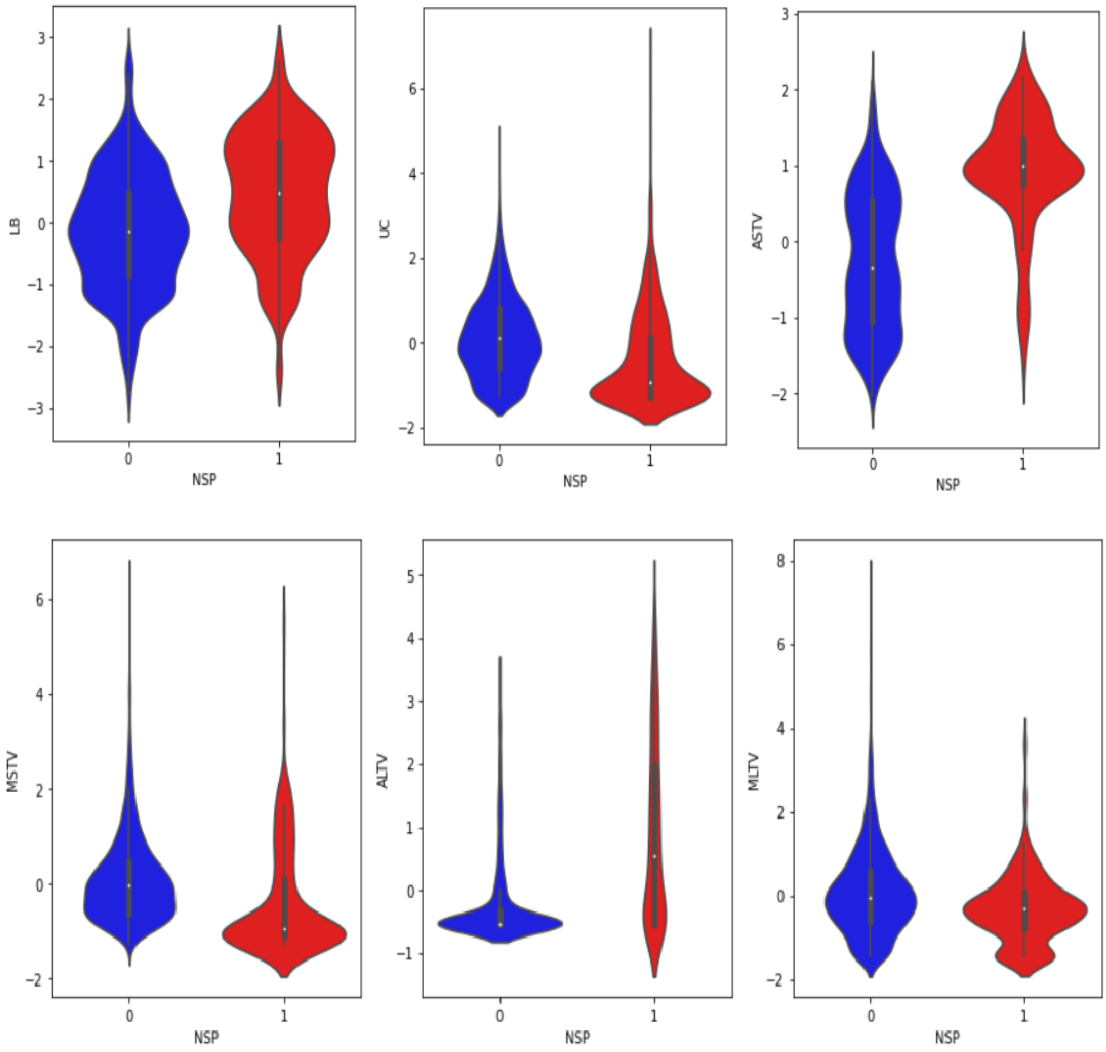

Figure 2: In this figure you can see the distribution divide by class for different features.

2.3 Feature correlation

Now we will analyze the correlation between features using the Pearson's coefficient.

Given two random variables X and Y, we can compute the Correlation coefficient as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E(X)^2} \cdot \sqrt{E(Y^2) - E(Y)^2}}$$

We can construct a matrix where the entries are the values of the correlation coefficient of two given variables. In our case, if the coefficient between 2 attributes has a values higher than .80 we delete one of them because they basically bring the same information.

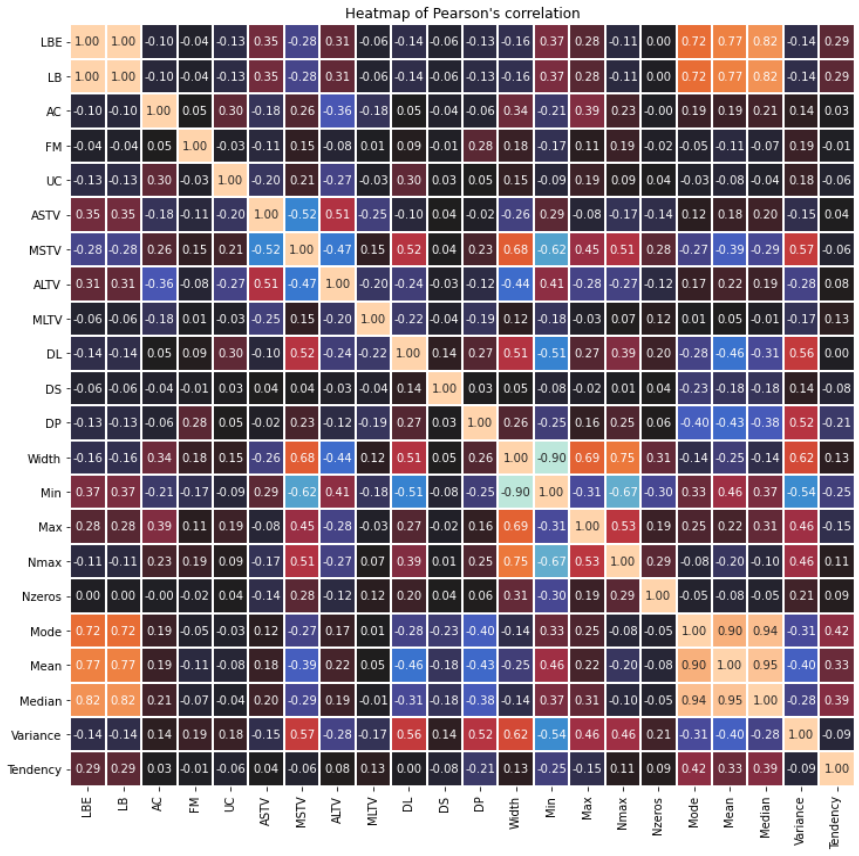

Figure 3: From the matrix you can see that some features have a strong correlation.

3 Data Preparation

After a careful analysis of the characteristic of the dataset, we need to correctly manage our data to train and test the algorithms. The first step is to divide the whole dataset in:

- **training set**: 90% of the original dataset is used to perform hyper-parameters tuning
- **test set**: 10% of the original dataset will be used only for the final evaluation of the performances of our models.

In order to enhance the performance of the algorithms, we will apply some transformations to the training set like standardization (section 3.1) and PCA (section 3.3.1). In the pipelines where these transformation are applied, we need to perform the same manipulations to the final test set. For example, if we want to standardize the training set, we will compute $\mu_{training}$ and $\sigma_{training}$ and next standardize the training data. We will then train and tune our model on this transformed training set. Next, in order to correctly compute the metric on the test set, we need to apply the same standardization to the test set. In case of normalization, we need to subtract $\mu_{training}$ and divide by $\sigma_{training}$ to the datapoint of the test set. The same reasoning apply to the case of PCA.

In order to faithfully estimate the error that our model is doing during training we will use the Cross Validation technique explained in section 4.2 and in order to preserve the proportion of our classes it is suggested to apply the stratified version of K-Fold Cross Validation.

Different pipelines will be exploited. To have a general idea: the standardization will be applied in all the pipelines, while the PCA and SMOTE techniques will be used one at a time or both together in order to see when they are useful. We do this in order to prove theoretical results.

3.1 Standardization

As we observed before, the distribution of many features of the dataset are far from being normal. This could be a problem because if we are comparing measures that have different scales it could happend that one of them will overwhelm the others. Moreover, having a standardized set of features is required for the optimal performance of many machine learning techniques such as: KNN, SVM and PCA.

In order to standardize the data, we need to compute the mean, subtract by the obtained value and then divide by the standard deviation. So, the following formula is applied:

$$z = \frac{x - \mu}{\sigma}$$

In this way, the distribution probability function is centered in zero and have a standard deviation equal to one.

3.2 Outliers detection

It could happened that inside our data there are some records that are completely or partially different from all the others. Sometimes it is useful to detect this anomalous values in order to remove them while in other cases they provide important information or they can actually be the objective of our research. When managing outliers we need to be extremely careful in particular in the medical field we need to take a lot of precaution with respect to the outliers management and in the following there will be an explanation.

3.2.1 Isolation Forest

Isolation Forest (IF) is an unsupervised anomaly detection technique. The method is based on two main fact: anomalous points are data that are easier to separate from the rest of the observations and are few with respect to the normal records. Isolation Forest are basically ensemble of binary trees. Each tree in the isolation forest is usually called isolation tree. The main idea to generate a isolation tree is to recursively divide the samples by randomly selecting an attribute, then on the given attribute we apply a random split in the range of the maximum and minimum values of that attribute. We can observe that the average number of partition required to isolate an anomalous point is lower with respect to the number of partition requested for identifying a normal data point.

Usually the outliers analysis is performed in order to delete the detected data but in our case, removing the outliers would further exacerbates the unbalance of our dataset as you can see from the picture below.

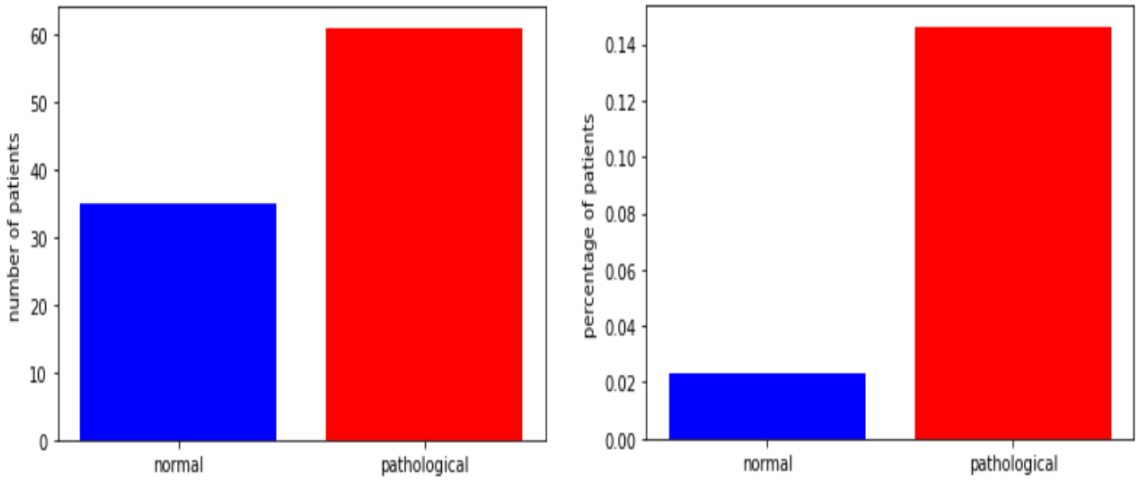

Figure 4: On the left of the figure you can see the number of data points that are labeled as outliers from our IF algorithm (35 normal and 61 pathological). On the right you can observe the percentage of outliers per diveded by class (2% normal and 15% pathological).

3.3 Dimensionality reduction

Dimensionality reduction is applied when we have data in a high dimensional space and we want to map it in a lower dimensional space. There are different reasons in order to apply such techniques. First of all, having high dimensional data require a huge amount of memory, so if we are able to reduce it we will save space and also speed up the process of our algorithm during training time. Second, in some cases high dimensionality could lead to poor performances in terms of generalization and finally, applying this technique allow us to create interpretable visualization.

3.3.1 Principal Component Analysis

Principal Component Analysis is a techniques that reduces the dimensionality of our data by applying a linear transformation to them. Let's suppose that our original data is in $\mathbb{R}^d$ and we want to reduce it in $\mathbb{R}^n$, where $n < d$. Our objective is to find a matrix $W \in \mathbb{R}^{n \times d}$ that induce the mapping $x \mapsto Wx$. Now we need to define a way that allow us to create a good W matrix and a natural idea is that we want to be able to recover x from the linear transformation discussed before. The linear recovery is defined as $\tilde{x} = Uy = UWx$.

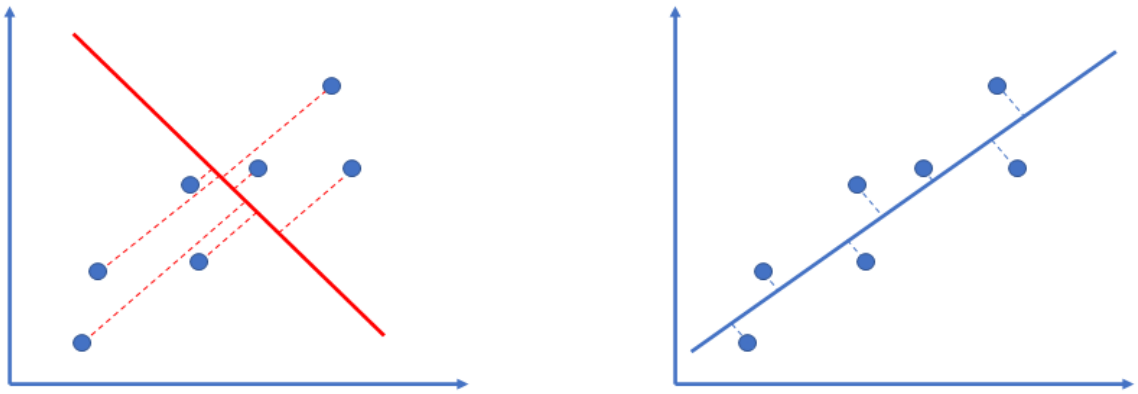

Figure 5: On the left: large projection error. On the right: good line to project to.

We use as measure that approximate the recovery the averaged square norm and for example if we have $x_1, x_2, ..., x_m$ we solve the following problem:

$$\underset{W \in \mathbb{R}^{n \times d}, U \in \mathbb{R}^{d \times n}}{\operatorname{argmin}} \sum_{i=1}^{m} ||x_i - UWx_i||_2^2$$

Thanks to a lemma we know that if (U,W) is a solution then the columns of U are orthonormal and $W = U^T$ and so we can write the previous optimization problem as:

$$\underset{U \in \mathbb{R}^{\cdot, \kappa} : \mathbb{U}^{\mathbb{T}}\mathbb{U} = \mathbb{I}}{\operatorname{argmin}} \sum_{i=1}^{m} ||x_i - UU^T x_i||_2^2$$

A simplification of the problem can be done by applying some elementary algebraic manipulations. For every $x \in \mathbb{R}^d$ and a matrix $U \in \mathbb{R}^{n,d}$ such that $U^T U = I$ we have:

$$\begin{aligned} ||x - UU^T x||^2 &= ||x||^2 - 2x^T UU^T x + x^T UU^T UU^T x \\ &= ||x||^2 - x^T UU^T x \\ &= ||x||^2 - trace(U^T xx^T U) \end{aligned}$$

where the trace of a matrix is the sum of its diagonal entries. Since the trace is a linear operator we can rewrite the problem as:

$$\underset{U \in \mathbb{R}^{d,n} : U^T U = I}{\operatorname{argmax}} trace\left(U^T \sum_{i=1}^{m} x_i x_i^T U \right)$$

The solution to this problem is the matrix U where the columns represent the n eigenvectors of $A = \sum_{i=1}^{m} x_i x_i^T$ corresponding to the largest n eigenvalues.

Before applying PCA, it is a common practice to center the data and so we could also proceed as follow:

- calculate the sample mean μ
- create the scatter matrix $S = \sum_{i=1}^{m}(x_i - \mu)(x_i - \mu)^T$
- compute the eigen(vectors,values) of S

The first Principal Component will be the eigenvector corresponding to the largest eigenvalues and the second PC will be the eigevector corresponding to the second largest eigenvalue and so on. In order to choose the number of PCs, we can use different approach such as to reach a percentage of the total explained variance or the average explained variance technique. We will proceed with the first option setting a threshold to 0.9 as total percentage of explained variance. To reach this common threshold we need 11 PCs.

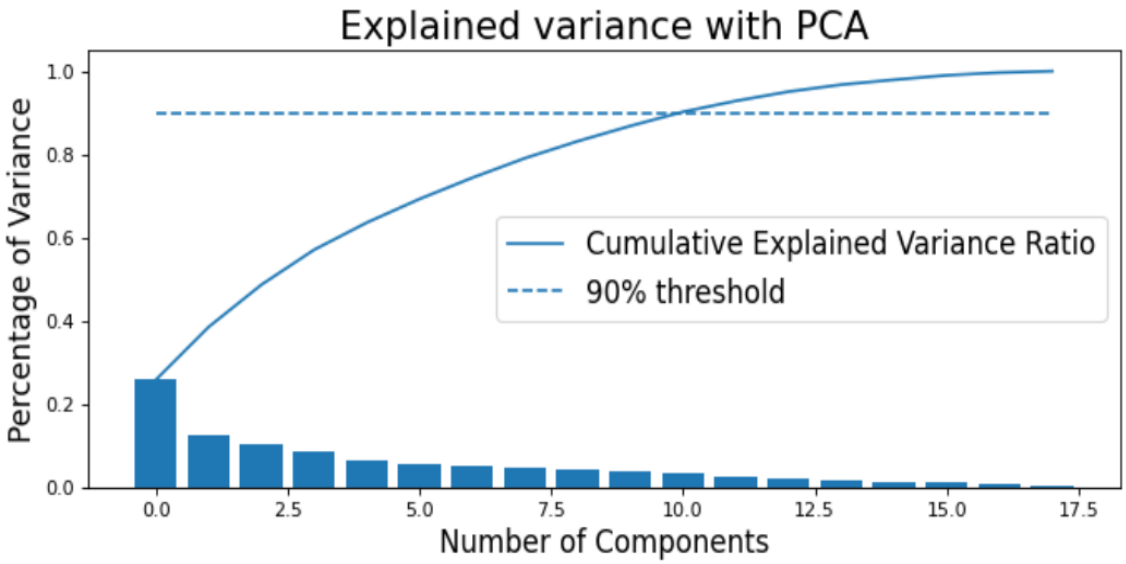

Figure 6: In this figure you can see the explained variance ratio and the cumulative explained variance ratio.

3.4 Synthetic Minority Oversampling TEchnique

Given that our data are highly unbalanced in terms of distribution of classes, we need and approach to fix the situation. SMOTE is a possible solution for this problem, in fact, this is an oversampling approach where the class with fewer record is oversampled creating new and 'synthetic' data. To perform this technique, we take a set of minority class sample exploiting the KNN algorithm and then we introduce in the lines between to different samples a new record of the minority class.

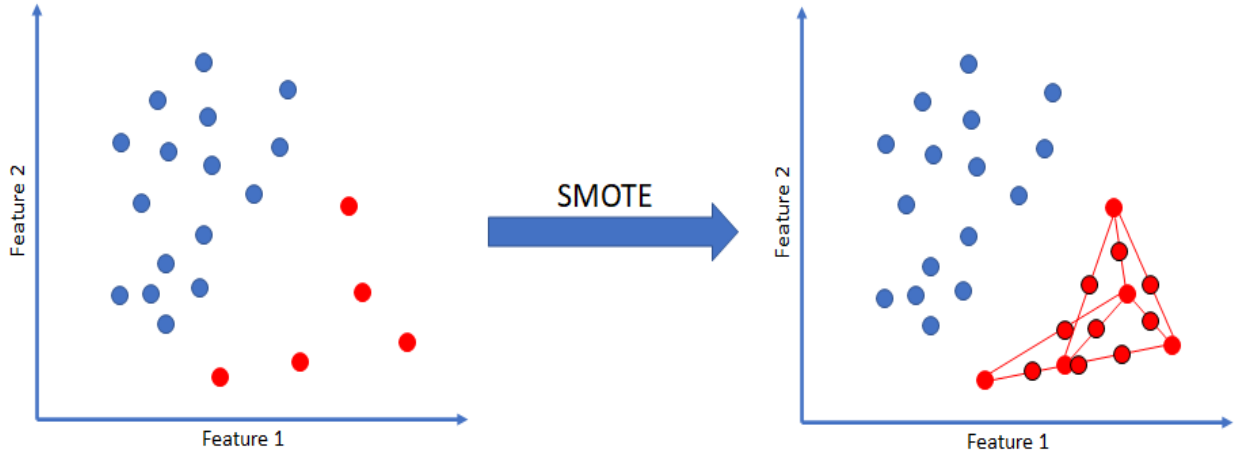

Figure 7: In this figure there is the representation of the concept of SMOTE with K=3. In the line that links two data points we create a new instance.

3.5 Random Oversampling Technique

In order to provide an alternative to the SMOTE technique and have more baselines, the application of another Oversampling techniques was made.

Random Oversampling consists in replicating examples from the minority class. The algorithm selects some record of the class with fewer instances and replicate the example putting new data into the training set.

4 Methodology

4.1 Metrics

In order to keep track of the performances of the algorithms that we are going to test we need to choose the most suitable evaluation metric. To correctly apply this concept we need to have clear in mind the meaning of the confusion matrix. The confusion matrix is a table where are collected the result of the algorithm. In the binary case, we will have 4 categories:

- True Positive: The elements that the algorithm has correctly assigned as positive
- True Negative: The elements that the algorithm has correctly assigned as negative
- False Positive: The elements where the algorithm has made a mistake and has assigned the positive class while the correct is the negative one
- False Negative: The elements where the algorithm has made a mistake and has assigned the negative class while the correct one is positive.

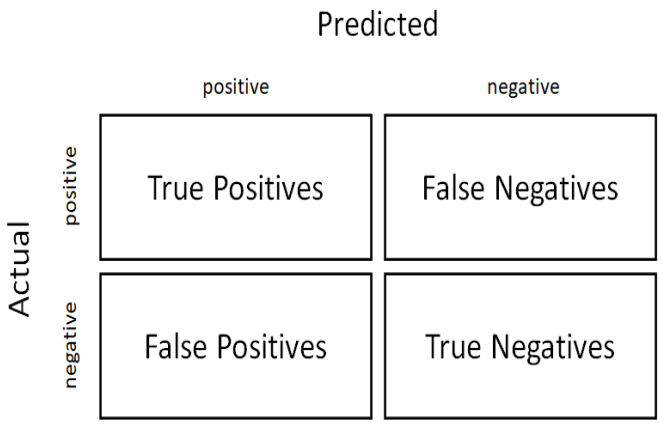

Figure 8: This is the representation of the confusion matrix.

Now that we have defined the confusion metrix, in the following you will find the definition of some metrics:

The accuracy measure is the most widely used in ML field and it is defined as the ration between the number of correct prediction over the total number of records used:

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

In some fields and in some cases the Accuracy is not a trustable measures and so we need to define other parameters as the recall and precision:

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

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

The recall tell us how many actual positive are correctly identified, while the precision is the measures that tell as the proportion of correct positive prediction.

A measure that combine this concepts all together is called the F1 measure and is defined as follow:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

4.2 Cross Validation

The Cross Validation technique is used in order to produce a faithful estimation of the error that our algorithm is doing. The most used approach is called the "K-Fold Cross Validation". Basically, we divide the training set into k smaller sets and then the following procedure is used for each of the K-Folds: A model is trained using $k-1$ of the folds as training data, next the performances of the resulting trained algorithm are computed using the remaining data. To obtain the final score of the model, we will average the result obtained for each K-Fold step. It is a computationally expensive approach, but it gives us a faithful result. It is usually preferred because if we use a naive training-eval split, the evaluation will be highly dependent on which data point we select for the trainging phase and for the evaluation phase. The optimal would be to have the same distribution of classes in the training and evaluation step.

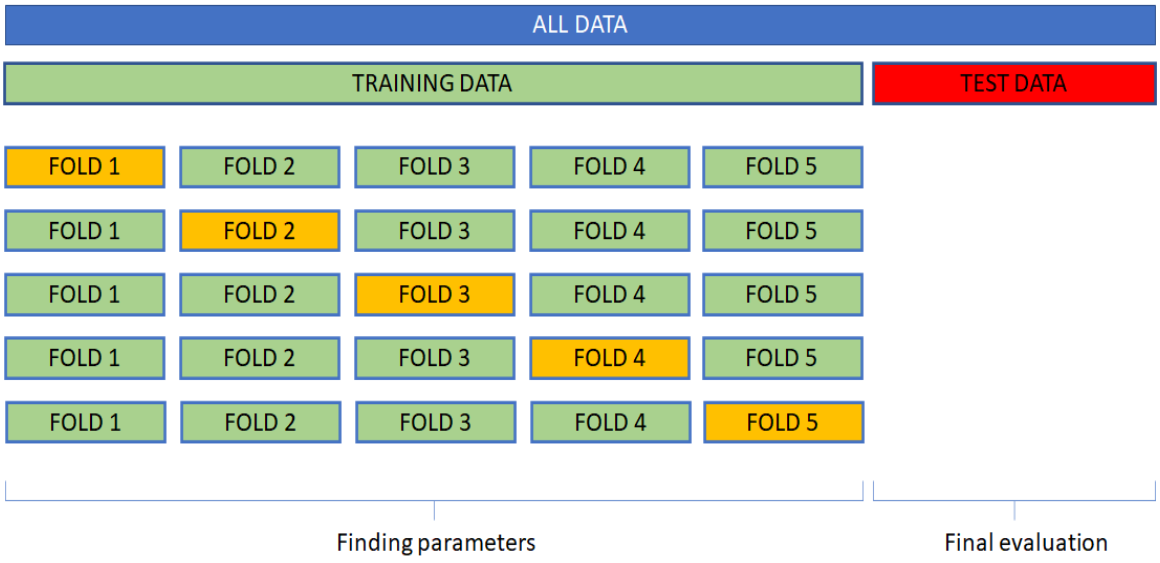

Figure 9: The idea of K-Fold Cross Validation. In this example we have 5 Folds and so K=5. In green you can see the fold used for training the model, in orange instead are highlighted the folds use for evaluating the model. In red there is the final test set.

5 Model Selection

5.1 Decision Trees

Decision trees are supervised algorithms that aim to stratify and segment the predictor space into a number of simple regions. For every observation that falls into a given region we predict that the point belong to the most commonly occurring class of training observations in the region to which it belongs.

Considering every single partition in the predictor space is infeasible. So, instead of finding the global optimal solution, we follow a top-down greedy approach that at each split find the local optimal solution. This approach is called recursive binary splitting.

We build a tree as follow: we select the attribute X_j and the cutpoint s such that splitting the feature space into the regions $\{X|X_j < s\}$ and $\{X|X_j > s\}$ leads to the greatest reduction of impurity of the given node.

The most common criteria are:

- Gini Index defined as $\sum_{k=1}^{K} p_{mk}(1 - p_{mk})$
- Cross Entropy defined as $-\sum_{p_m k}^{K} log(p_{mk})$

Next, we repeat the process till some criteria is satisfied.

Usually, trees produce very good predictions on the training set but if we feed a new data point to the algorithm we discover poor performances and this is due to overfitting. In order to tackle the problem we can use an early stopping strategy or build a big tree and then prune it. The main advantage of the trees is that they are quite simple and are interpretable, the drawback is that the accuracy obtained with this technique is not comparable with other supervised learning algorithms.

5.2 Random Forest

In order to solve the problem of low accuracy of the tree, we try to find more accurate predictions exploiting the "potential of the crowd". In fact, we can create ensembles of trees that are able to drastically increase the performances in terms of accuracy in the test set. We build the Random Forest applying bootstrap aggregation to the training set and also feature bagging. Bootstrap aggregation is a general-purpose procedure for reducing the variance of a statistical learning method and it consist in creating set of sample from the original training set and creating different trees from each different set. We want also to decorrelate the trees and so, during the building process, each time that we need to split a node a random subset of predictors is considered, When we need to make a prediction, we record the class predicted bu each of the trees in the random forest and we choose the class by majority vote.

5.2.1 Random Forest results

Normalized	PCA 90%	SMOTE	Criterion	Max Depth	#Estimators	F1
V	X	X	entropy	10	100	89.22
V	V	X	entropy	50	100	79.76
V	X	V	gini	50	100	90.11
V	V	V	entropy	50	100	79.77

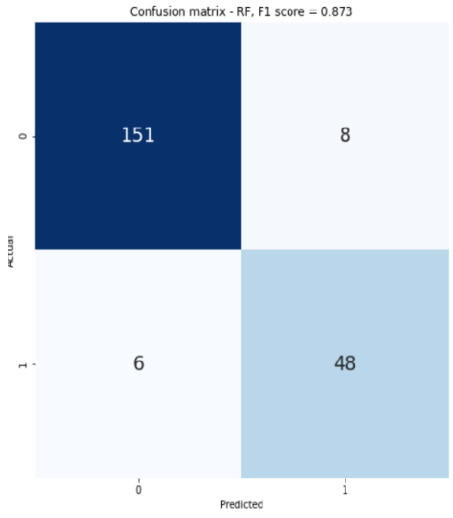

Figure 10: Final results of Random Forest applying pipeline: Normalization+SMOTE.

5.3 Support Vector Machine

Support Vector Machines are linear predictors. In SVM we use concepts like: margin and support vectors. The margin is defined as the minimum distance between a point and the linear separator while the support vectors are data points that are closer to the hyperplane and influence the position and orientation of the separator. The goal is to find a hyperplane that separates the data, more specifically we want the hyperplane that maximize the margin between classes. Roughly speaking, a halfspace separates a training set with a large margin if all the examples are not only on the correct side of the separating hyperplane but also far away from it. The advantage of having large margin is that even if we slightly perturb the values of our data, we will still separate them correctly,

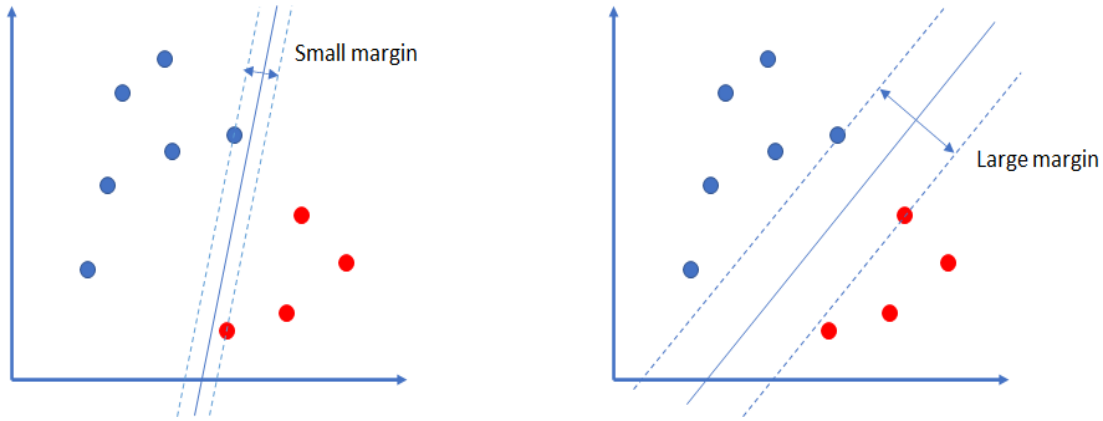

Figure 11: Linear Support Vector Machine. On the left you can see an example of small margine, while on the right we have a larger margin. In the algorithm we want to miximize the value of the margin.

We can define the primal problem as:

$$\underset{(w,b):||w||=1}{\operatorname{argmax}} \min_{i\in[m]} |\langle w, x_i\rangle + b| \quad s.t. \quad \forall i, (\langle w, x_i\rangle + b) > 0$$

In the case in which data are not linearly separable, we relax the problem with the introduction of a non-negative slack variable ε and we develop the so called: soft-SVM. Basically, we don't make the strong assumption of linear separability as before and we relax the hard-SVM problem. In fact, we allow the system to make some classification errors. In order to keep track of the mistake we will introduce a form of cost: the Hinge Loss.

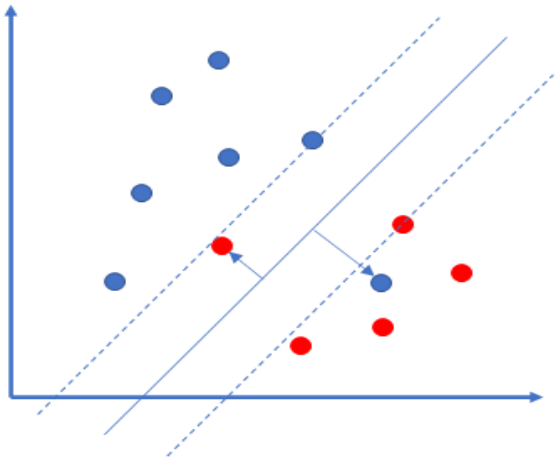

Figure 12: Linear Support Vector Machine. Soft Case.

The problem that we want to optimize is the following:

$$\min_{w}(\lambda||w||^2 + L_S^{hinge}(w))$$

where

$$L_S^{hinge}(w) = \frac{1}{m}\sum_{i=1}^{m} max\{0, 1 - y\langle w, x_i\rangle\}$$

The non-negative slack variable is incorporated in the meaning of the hinge loss. The λ parameter measure the tradeoff between the minimization of the norm of w and the average violation of the classification constraint.

If data are not linearly separable and allowing mistakes does not provide satisfactory result, we can exploit Kernel-SVM. Firstly we embed our data in a higher dimensional feature space and then we try to find a linear decision boundary in the new augmented space. Then we come back to the original feature space and here we can find "separating surfaces" with highly non-linear behaviors.

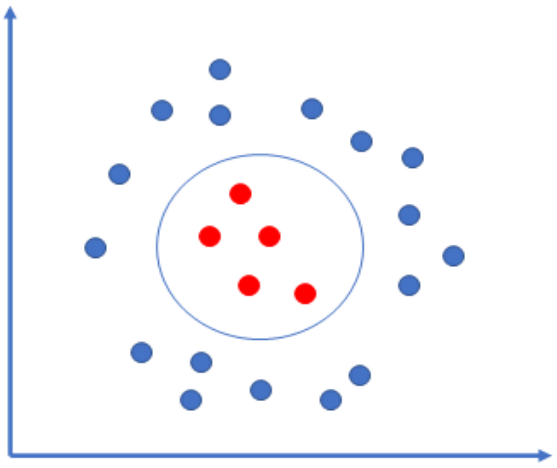

Figure 13: Kernel Support Vector Machine.

We can notice that the SVM optimization problem has the following general form:

$$\min_{w}(f(\langle w, \psi(x_1)\rangle, ..., \langle w, \psi(x_m)\rangle) + R(||w||))$$

Thanks to the representer theorem we can optimize the previous equation with respect to the coefficients α instead of the coefficients w as follows. Writing $w = \sum_{j=1}^{m} \alpha_j \psi(x_j)$ we have that for all i

$$\langle w, \psi(x_i)\rangle = \left\langle \sum_j \alpha_j \psi(x_j), \psi(x_i) \right\rangle = \sum_{j=1}^{m} \alpha_j \langle \psi(x_j), \psi(x_i)\rangle$$

and similarly

$$||w||^2 = \left\langle \sum_j \alpha_j \psi(x_j), \sum_j \alpha_j \psi(x_j) \right\rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j \langle \psi(x_i), \psi(x_j) \rangle$$

Then, we define $K(x, x') = \langle \psi(x), \psi(x') \rangle$ as a function that implements the kernel function with respect to the embedding ψ. Substituting the previous result in the general formula we obtain as equivalent problem:

$$\min_{\alpha \in \mathbb{R}^m} f\left(\sum_{j=1}^{m} \alpha_j K(x_j, x_1), ..., \sum_{j=1}^{m} \alpha_j K(x_j, x_m) \right) + R\left(\sqrt{\sum_{i,j=1}^{m} \alpha_i \alpha_j K(x_i, x_j)} \right)$$

To solve this optimization problem we do not need direct access to the elements in the new feature space but we need to just compute the inner products in the feature space. In practice we need to know the value of the Gram matrix: $G_{i,j} = K(x_i, x_j)$

The soft-SVM problem can be rewritten as:

$$\min_{\alpha \in \mathbb{R}^m} \left(\lambda \alpha^T G \alpha + \frac{1}{m} \sum_{i=1}^{m} max\{0, 1 - y_i (G\alpha)_i\} \right)$$

Between the most used Kernel function we find polynomial kernels:

$$K(x_i, x_j) = (\langle x_i, x_j \rangle + 1)^d$$

and Gaussian kernels:

$$K(x_i, x_j) = e^{-\frac{||x_i - x_j||_2^2}{2\sigma^2}}$$

5.3.1 Support Vector Machine results

In the tuning process the following hyperparameter have been evaluated:

- C:[1,10]
- kernel: ['linear', 'rbf', 'polynomial']

Moreover, different pipeline have been used before applying the algorithm to the data. In a few words the pipeline differ with respect to a combination of dimensionality reduction and oversampling techniques. As expected the best results are obtained without the application of PCA because in the case of dimenstionality reduction the algorithm will have hard time to find a solution in a more "crowded" and lower dimensional space, in fact, the best solutions are found when the dimensionality is higher.

Normalized	PCA 90%	SMOTE	F1 HARD-SVM	F1 SOFT-SVM	F1 RBF-SVM
V	X	X	77.91	78.58	85.93
V	V	X	77.16	76.97	81.31
V	X	V	78.08	79.12	85.29
V	V	V	77.06	77.04	80.64

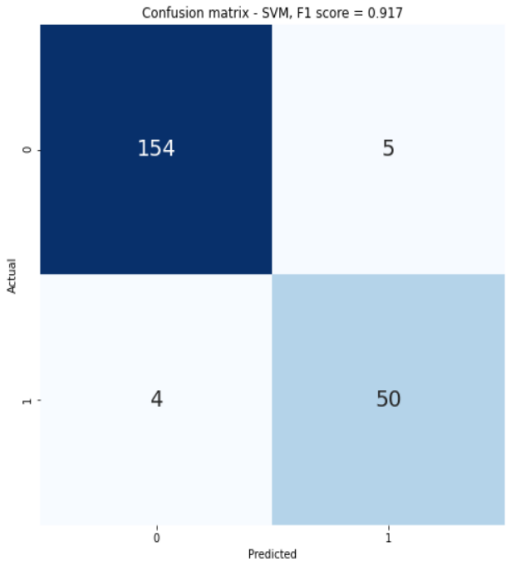

Figure 14: Final results of Support Vector Machine. On the left you can see the result of the tuning parameter, while on the right there is the result on the test set for the pipeline: Normalization.

5.4 K-Nearest Neighbors

Nearest Neighbors is the simplest family of algorithms in the Machine Learning field. Basically, we store all the information that are in the training set (features and labels). When we need to predict and classify a new record, we find the points in the training set that are closer to our new instance and then we find at which classes they belongs. We assign to the new instance the label apply a majority voting strategy.

5.4.1 K-Nearest Neighbors results

The hyperparameter tuning was made evaluating:

- number of neighbors : [3, 5, 7, 11]
- weights: 'uniform', 'distance'

Normalized	PCA 90%	SMOTE	#neighbors	weights	F1
V	X	X	5	distance	81.06
V	V	X	3	distance	81.49
V	X	V	3	distance	81.83
V	V	V	3	distance	82.16

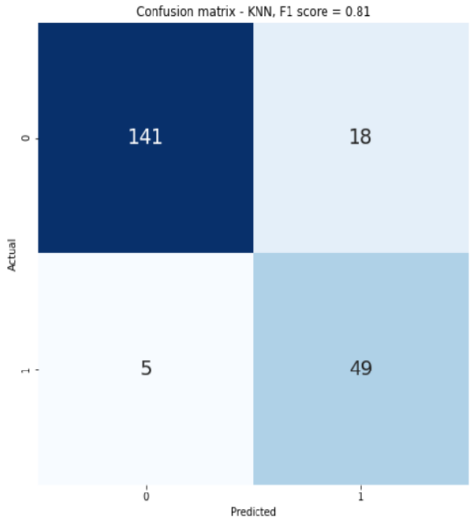

Figure 15: Result of KNN algorithm. The final confusion matrix refer to the pipeline: Normalize+PCA+SMOTE.

6 Curse of Dimensionality

Before conclusion, I would like to underline that the obtained results make arise an important issue in the Machine Learning world: the Curse of Dimensionality. We can clearly see this phenomenon comparing the result between SVM and KNN. It is important to highlights that this behavior is not caused by the algorithms but it is related to the intrinsic geometry of the euclidean space when you work in high dimensions.

The main concept to understand is that in high dimension the space is empty.

Assume that data lives in $[0,1]^p$, where p is the number of dimensions. To capture a neighborhood which represents a fraction s of the hypercube volume, we need the edge length to be $s^{\frac{1}{p}}$. For example:

- $p = 10, \quad s = 0.01, \quad s^{\frac{1}{p}} \approx 0.63$
- $p = 10, \quad s = 0.10, \quad s^{\frac{1}{p}} \approx 0.79$
- $p = 20, \quad s = 0.10, \quad s^{\frac{1}{p}} \approx 0.89$

We can read the previous point as: In p dimension, if you want to have $s\%$ of the volume, you need to take $s^{\frac{1}{p}}$ of the edge length.

Basically, this means that even to get a small fraction of volume we need to take a large portion of the edge length and we can translate it into: "if we want to collect a certain percentage of data we practically need to take the entire box". In this conditions the concept of local neighbors vanishes.

We can observe that the diagonal joining the origin $(0, 0, ..., 0)$ and a vertex $(1, 1, ..., 1)$ is equal to $\sqrt{p}$. So, $[0,1]^p$ has all size length 1 and the diagonal of length $\sqrt{p}$. If we increase the dimension we will always have edge length equal to one but the diagonal will become

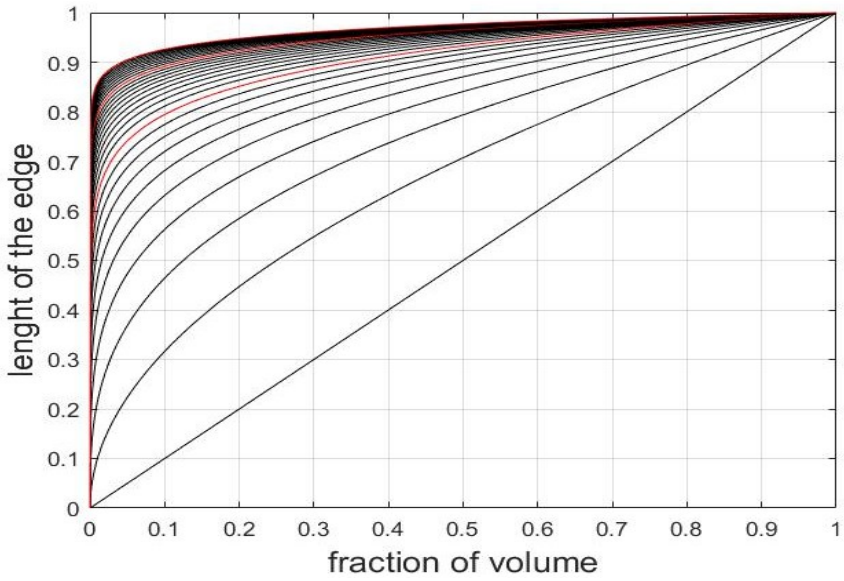

Figure 16: Relationship of the n° of dimensions, the hypercube volume and the edge length

larger and larger. A consequence is that the volume of an hypercube with edge length lower than 1 is basically 0 and to overcome this limitation we need a number of sample which grows exponentially with the number of dimensions.

Let's take two independent random variables X, Y uniformly distributed on $[0,1]^p$. We can demonstrate that the mean square distance $||X-Y||^2$ satisfies:

$$\alpha = \mathbb{E}[||X-Y||^2] = p/6 \quad \beta = std[||X-Y||^2] \approx 0.2\sqrt{p}$$

As we can clearly see, as the number of dimension p grows, the scaled standard deviation ($\frac{\beta}{\alpha} \approx p^{-\frac{1}{2}}$) shrink to 0. This means that point are basivally equally distant from one another and the concept of neighbors vanishes as the dimensions increase.

The high-dimensional spaces are almost empty and so it is easier to separate groups with classifiers such as SVM but we tend to overfit, while it will be really hard to obtain satisfactory result with KNN where the concept of neighbors play a key role.

7 Conclusion

In the following table you can see the best pipeline associated with each model and the corresponding F1 score.

model	**pipeline**	**F1 score**
RF	Normalization + SMOTE	89.30
RBF-SVM	Normalization	91.74
KNN	Normalization + PCA + SMOTE	82.35

Table 1: Result on the test set of the different algorithms associated with the best combination of pipeline and hyperparameters.

We can observe that the obtained results respect the theory. That means that usually it is better to perform SVM without applying the dimensionality reduction because even intuitively, it should be easier to separate data that are more sparse. Instead, with an algorithm like KNN we gain an advantage by applying the Principal Component Analysis.

Overall the best performing algorithm is the RBF Kernel SVM.

In the medical field it is preferable to adopt a precautionary approach, so that if we make a mistake it is better to be unbalance toward the False Positive instead of False Negative. From the confusion matrix shown in the previous pages we can observe that also in this perspective we obtain satisfactory results. Anyway, given the particular field in which we apply this "cold" algorithm I will strongly encourage to keep the result as suggestion to the doctors and not as final decision. The output obtained by the algorithm should be integrated by the knowledge of the experts. Moreover, I will probably encourage the integration of other documentation and other parameters in order to better understand the situation of the patients and give the most suitable and precise support to them.