POLITECNICO
DI TORINO

Prediction of death events for heart failure clinical records

Data Spaces, A.Y. 2020/2021
Politecnico di Torino

Luca Barco s276072
luca.barco@studenti.polito.it

Abstract

The cardiovascular diseases are one of the most frequent causes of death. They mainly causes heart failures: the hearth cannot pump enough blood to satisfy the needs of the body. Nowadays, the electronic medical records can help in finding correlations between symptoms and body values, and applying machine learning algorithms can help to predict the patient's survival and to highlight the most important features to take in consideration.
This work aims to train classifiers in order to predict the death of a patient with an heart failure. It is based on the dataset used in the work of Chicco and Jurman [1] "Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone".

Contents

1. Introduction

The aim of this work is to build and compare several classification models in order to predict the death of the patients starting from a dataset containing clinical, body and lifestyle data data about heart failures .

We will analyze the dataset, looking at the features and their distribution, we will evaluate if a dimensionality reduction can be useful and then we will train different models considering different scenarios.

The code was developed on Google Colab plaform, and it is available here:
`https://github.com/LucaBarco/TesinaDataSpaces/blob/main/Tesina_s276072.ipynb`

2. The Dataset

The dataset contains the medical records of 299 patients with an hearth failure gathered in Pakistan during 2015.
In this section we will analyze the features and their distribution.

2.1. Features analysis

The dataset contains 13 features which report clinical, body and lifestyle data, described by Table 1.

Feature	Explanation	Measurement	Range
Age	Age of the patient	Years	[40,...,95]
Anaemia	Decrease of red blood cells or hemoglobin	Boolean	0, 1
High blood pressure	If a patient has hypertension	Boolean	0, 1
Creatinine phosphokinase (CPK)	Level of the CPK enzyme in the blood	mcg/L	[23, ..., 7861]
Diabetes	If the patient has diabetes	Boolean	0, 1
Ejection fraction	Percentage of blood leaving the hearth at each contraction	Percentage	[14, ..., 80]
Sex	Female or Male	Boolean	0, 1
Platelets	Platelets in the blood	kiloplatelets/mL	[25.01, ..., 850.00]
Serum creatinine	Level of creatinine in the blood	mg/dL	[0.50, ..., 9.40]
Serum sodium	Level of sodium in the blood	mEq/L	[114, ..., 148]
Smoking	If the patient smokes	Boolean	0, 1
Time	Follow-up period	Days	[4, ..., 285]
Death event	If the patient died during the follow-up period	Boolean	0, 1

Table 1: **Features description**. [mcg/L: micrograms per liter. mL: microliter. mEq/L: milliequivalents per litre]

- *CPK*: when a muscle tissue gets damaged, it flows into the blood. High levels of CPK might indicate a hearth failure or injury.
- *Serum creatinine*: is a waste product generated by creatine when a muscle breaks down. It's generally used to check kidney functions (an high value might indicate a renal dysfunction).
- *Serum sodium*: sodium is a mineral used for the correct functioning of muscles. An abnormal value of sodium in the blood might be caused by an hearth failure.
- The target feature is **Death event**: we will use the terms *death* and *no death* for the two classes.
- The **time** feature is the follow-up period after a hearth failure. It is a particular feature, as observed by Chicco and Jurman [1] in their work. Even if it is not a clinical feature, we will consider it in our models in order to understand how it affects the predictions.

In Figure 1 there is the plot of the number of death and no death events.

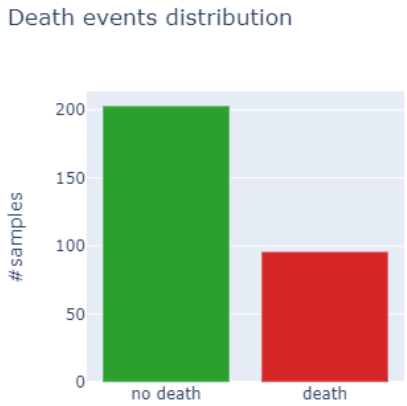


Figure 1: Amount of Death and No death events

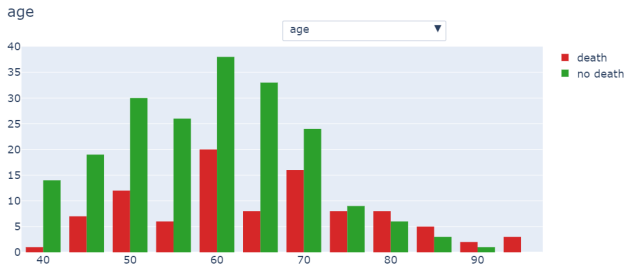
We can highlight an unbalancing between the two classes. This is a problem because there are models (like KNN) in which an unbalancing can cause problems (because the predictions are affected by an huge presence of samples belonging to a class with respect to another one).

In this work we compare two solutions for this problem: an under-sampling on the no-death distribution and a **Synthetic Minority Oversampling Technique (SMOTE)**. SMOTE works by selecting samples that are close in the feature space, drawing a line between them, and picking a new sample along that line.

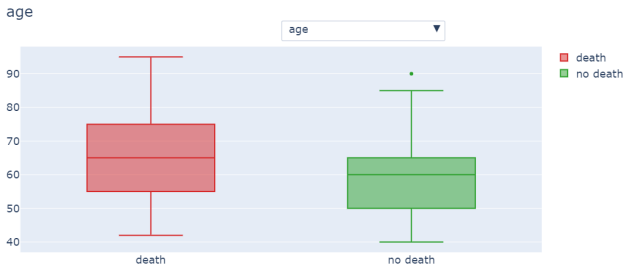
We can also plot, for each feature, the distribution of death and no death events, and the corresponding box plots only for the non-boolean features.
A box plot helps in visualizing the data distributions. It shows a box with some information:

- the median of the data: the line that divides the box in two parts.
- the lower quartile (Q1, the 25% of the data fall below this value) and the upper quartile (Q3, the 75% of the data fall below this value), and they are used to compute the interquartile range IRQ.
- the lower fence: Q1-1.5*IQR
- the upper fence: Q3+1.5*IQR
- the outliers: the values that appear out of the lower and upper fences

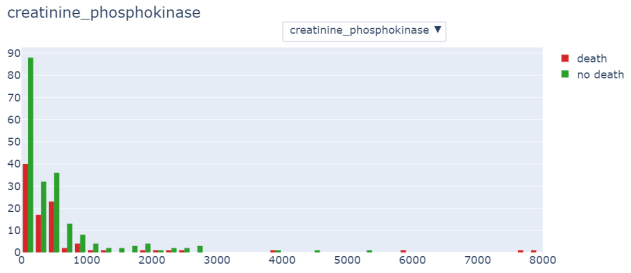
This plot can help us in understanding how the data are distributed, if they are dispersed and if the distribution is symmetrical.
In Figures 2 and 3, there are the distributions and the box plots for each feature.



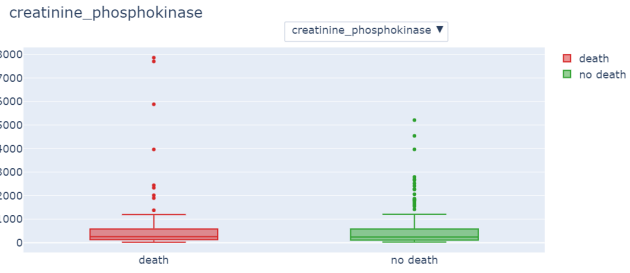
(a) Distribution of *Age* feature [years]



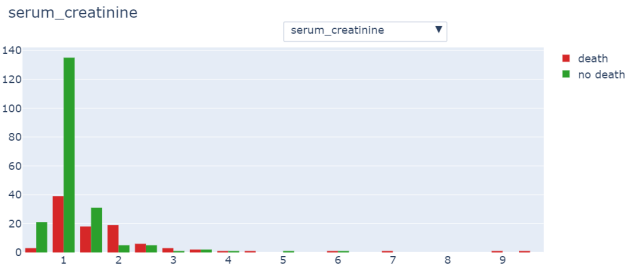
(b) Box plot of *Age* feature [years]



(c) Distribution of *Creatinine phosphokinase* feature [mcg/L]

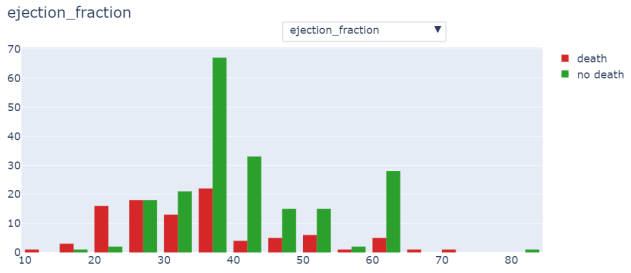


(d) Box plot of *Creatinine phosphokinase* feature [mcg/L]

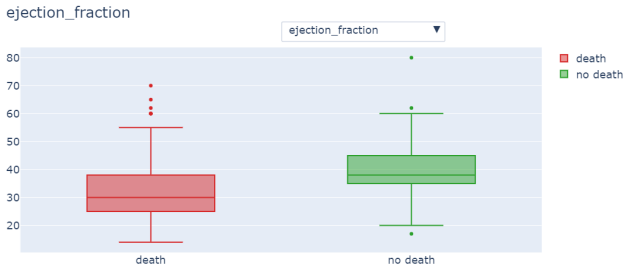


(e) Distribution of *Serum creatinine* feature [mEq/L]

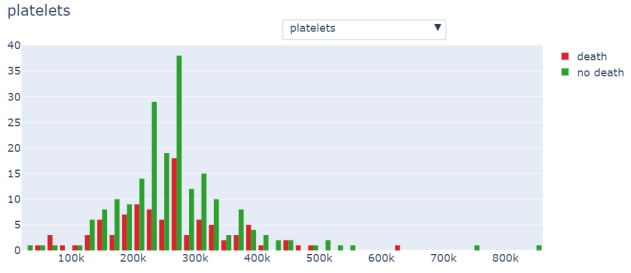
(f) Box plot of *Serum creatinine* feature [mEq/L]



(g) Distribution of *Ejection fraction* feature [%]



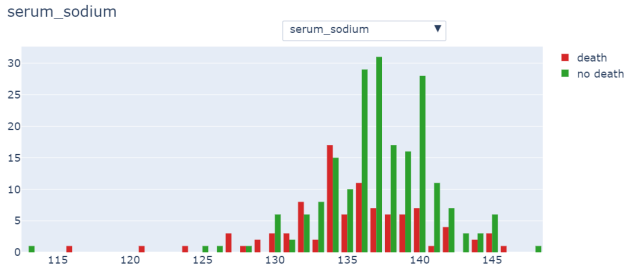
(h) Box plot of *Ejection fraction* feature [%]



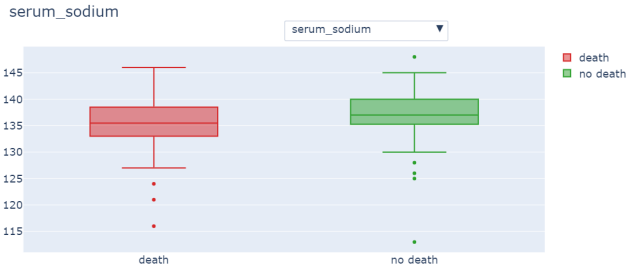
(i) Distribution of *Platelets* feature [kiloplatelets/mL]

(j) Box plot of *Platelets* feature [kiloplatelets/mL]

Figure 2: Plots of death and no death trends for each feature

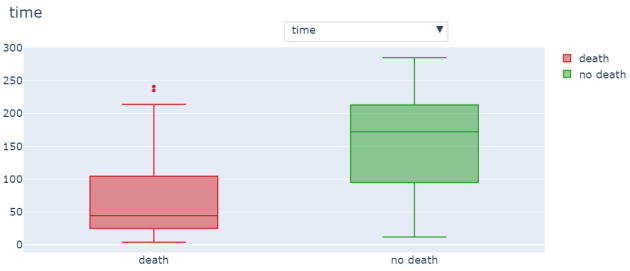


(a) Distribution of *Serum Sodium* feature [mEq/L]

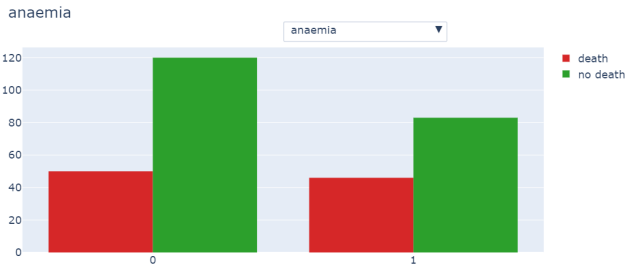


(b) Box plot of *Serum Sodium* feature [mEq/L]

(c) Distribution of *Time* feature [Days]

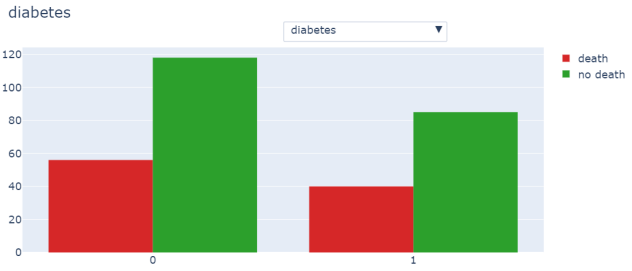


(d) Box plot of *Time* feature [Days]



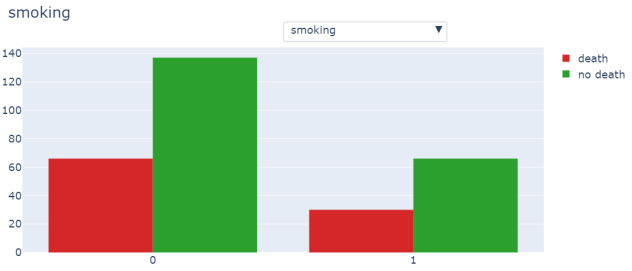
(e) Distribution of *Anaemia* feature [Boolean]

(f) Distribution of *High blood pressure* feature [Boolean]



(g) Distribution of *Diabetes* feature [Boolean]

(h) Distribution of *Sex* feature [Boolean]



(i) Distribution of *Smoking* feature [Boolean]

Figure 3: Plots of death and no death trends for each feature

- **Age**: we observe similar trends for death and no death events. The interesting fact is that the older the patients, the higher the probability to have a death event. In particular, the proportion between death events and no death events is reversing after an age of 70. Looking at the box plot, we can observe that the distributions are quite similar, there is more dispersion in the death distribution because there are death events for each range of age values. Furthermore, the median of the death box corresponds to the upper quartile of the no-death one.
- **Creatinine phosphokinase**: The death and no death trends have particular behaviours. They look similar to a branch of hyperbola. The most of the deaths happen in the range 0-600. In the box plot we can observe a very huge number of outliers but the boxes have similar median values.
- **Serum creatinine**: death and no death have similar trends. The box plot shows different boxes: the death distribution has more variance in the data with respect to the no death one. There are also a lot of outliers.
- **Ejection fraction**: the death and no death are similar to gaussians. The box plot contains different boxes, centered at different values and with different boxes length. There are outliers in the death distribution and the median of the no death box corresponds to the upper quartile of the death box.
- **Platelets**: death and no death have both gaussian trend. The box plots have similar median values and the boxes have different lengths. Furthermore, the no death distribution has more outliers.
- **Serum sodium**: the death and no death are similar to two gaussians centered at different mean values. The box plots have a slight difference in terms of box length.
- **Time**: most of the people die in the first follow up period, and this leads the two distributions of death and no death events to be really different among them. In particular, the higher numbers of deaths appear in the first days of follow up period, then the trend decreases. This is confirmed also by the box plots, in which we can observe two very different boxes, with very different medians and lengths. So, this feature can potentially create a bias in the models: the number of death events is lower than the no death one, and they are concentrated in the first follow-up days. Furthermore, this is not a clinical feature, so if we want to build models based only on clinical features we have to not consider it, otherwise we can consider it to understand how this "external" feature can affect the predictions, and this is the case of our work.
- **Anaemia**: The proportion of death/no death with respect to the presence or absence of anaemia is almost the same. (No: 30% death vs 70% no death, Yes: 35% death vs 65% no death)
- **High blood pressure**: the proportion is almost the same between diabetes and not diabetes. There are more deaths for patients without diabetes. (No: 30% death vs 70% no death, Yes: 36% death vs 64% no death)
- **Diabetes**: same proportions. (No: 32% death vs 68% no death, Yes: 32% death vs 68% no death).
- **Sex**: men have more heart failure than women. Considering the sex alone, the proportions are the same between death and no death for men and women. (Female: 32% death vs 68% no death, Male: 32% death vs 68% no death).
- **Smoking**: there are more death events for no smoking patients. (No: 33% death vs 67% no death, Yes: 31% death vs 69% no death).

2.2. Correlation Matrix

We can also look at the correlation matrix in Figure 4. It is represented using an heatmap. The correlation matrix helps in visualizing the positive and negative correlations between pairs of features.

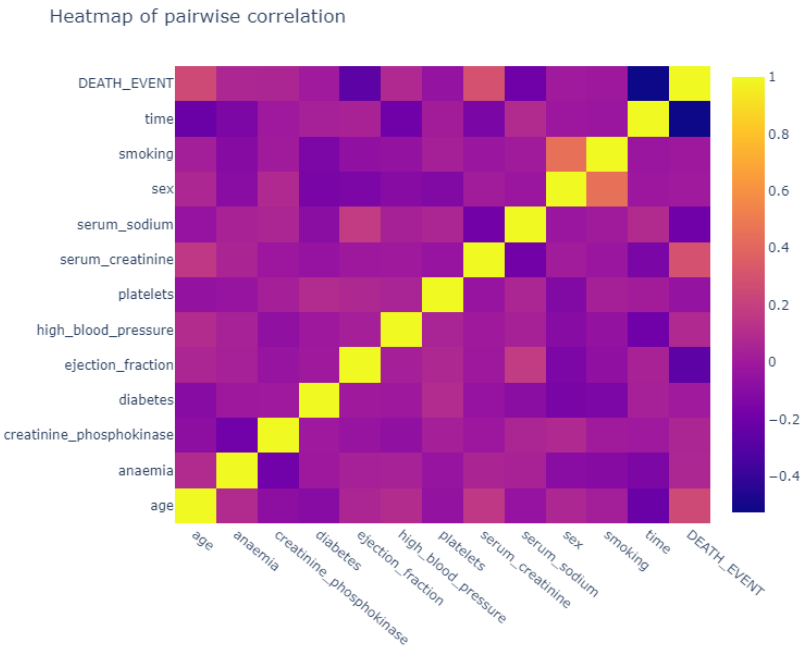


Figure 4: Features' correlation matrix

We can observe that almost all the features are uncorrelated. There is a negative correlation between the *death* event and *time*, as expected, and a slight positive correlation between *sex* and *smoking*. There are also slight negative correlations between *serum_creatinine* and *serum_sodium* and between *anaemia* and *creatinine_phosphakinase*.

2.3. Data preprocessing

In order to train our models, we have to first perform some operations on the data. We have first to do a normalization. We can do:

- a **min-max normalization**: squash the values of the feature in the range [0, 1]

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- a **z-score normalization**: normalize the values to be similar to a gaussian with mean equal to 0 and variance 1

$$x' = x - \mu\sigma$$

In general, for features belonging to different scales, is better to do the Z-score normalization, so we decide to use it.

2.4. Split in Training and Test set

We randomly split the dataset in 2 parts: Validation and Test set, containing the 70% and the 30% of the data, respectively. Starting from 299 samples, we have a training set containing 239 samples and a test set containing 60 samples.

3. Dimensionality Reduction through PCA

In our dataset we have 12 features (without the target feature DEATH_EVENT). We can try to reduce the dimensionality of our problem using the Principal Component Analysis (PCA). It is a tool to reduce the dimensionality of the data in order to avoid the curse of dimensionality. It is an unsupervised method based on the correlation between features. We have our data

$$x = [x_1, ..., x_d], x \in \mathbb{R}^d$$

We want to define a d x k-dimensional transformation matrix

$$W \in \mathbb{R}^{dxk}$$

that allows us to map x onto a new k-dimensional feature subspace, with k < d.

$$xW = z$$

obtaining the output vector

$$z = [z_1, ..., z_k], z \in \mathbb{R}^k$$

The algorithm is such that the first principal component is the one with the largest possible variance, and also the next one but given the constraint that all the principal components must be uncorrelated among them. In particular, we have to:

- Standardize the d-dimensional dataset
- Construct the Covariance matrix Σ, where each entry σ_{ij} is

$$\sigma_{ij} = 1n - 1\sum_{i=1}^{n}\left(x_j^{(i)} - \mu_i\right)\left(x_k^{(i)} - \mu_k\right)$$

- Decompose the covariance matrix into its eigenvectors and eigenvalues: $\Sigma v = \lambda v$
- Sort the eigenvalues by decreasing order basing on the corresponding eigenvectors
- Select k eigenvectors, which correspond to the k largest eigenvalues, where k is the dimensionality of the new feature subspace (k$\leq d)Constructaprojectionmatrix, W, fromthetopeigenvectors : eachrowcontainaneigenvector.$

- Transform the d-dimensional input data x using the projection matrix W in order to obtain the new k-dimensional feature subspace: $x' = xW$.

Applying this procedure to our data, we can plot the explained variance of the PCs.

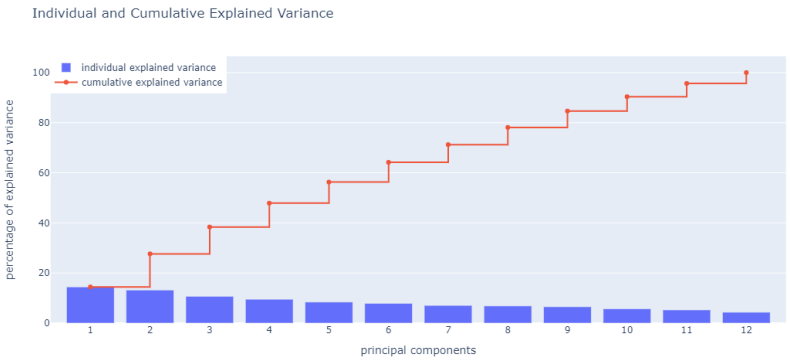


Figure 5: PCA applied on all 12 features: Explained variance

There is a sort of balance among Principal Components, as we can see from Figure 5. Thus, in our case the PCA is not so useful: to cover the 80% of the variance we have to take 9 over 12 features. Besides this, we will consider a reduced dataset using this procedure to see if it can improve the performance of our models.

For sake of completeness, we can also plot a biplot of the first two Principal Components in order to know which features affect the PCs, in Figure 6.

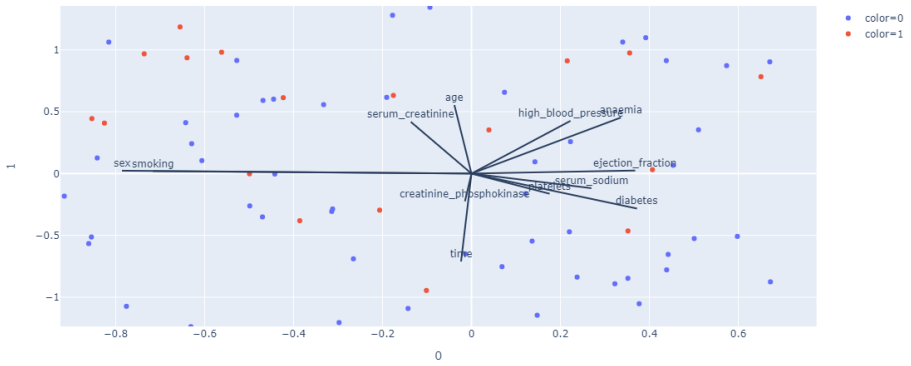


Figure 6: Biplot of the first 2 PCs

From this chart we can see that PC1 is mainly affected by *smoking* and *sex*, that are positive correlated, and by *ejection_fraction* that is negatively correlated with respect to them. PC2 is mainly affected by *time*, and it has also a negative correlation with respect to *age* and *serum_creatinine*.

4. Classification models

In this section we train and analyze different classification models. In particular, we will train Support Vector Machines, Linear regression, K Nearest Neighbors, Decision Trees and Random forests. For each model we perform a grid search on a predefined set of hyper parameters and we evaluate them using a metric.

4.1. Configurations

We train the models considering several configurations:

- **Configuration 1**: The entire Dataset
- **Configuration 2**: Reduced Dataset with PCA
- **Configuration 3**: Reduced Dataset with undersampling
- **Configuration 4**: Balancing with SMOTE

For each model we use the **K-fold Cross Validation Technique**: the training set is splitted into k subsets and we train k times the model considering k-1 subsets as the training dataset and the remaining one as the validation dataset.

4.2. Evaluation metrics

The definition of a metric is really important in order to evaluate the classifiers. First of all, given a classification task, we can define:

- **TP** True Positive: number of correct positive class predictions
- **FP** False Positive: number of uncorrect positive class predictions
- **TN** True Negative: number of correct negative class predictions
- **FN** False Negative: number of uncorrect negative class predictions
- **PRECISION**$= \frac{TP}{TP+FP}$
- **RECALL**$= \frac{TP}{TP+FN}$

Then, we can define the following metrics:

- **Accuracy** $= \frac{TP+TN}{TP+TN+FP+FN}$
- **F1-SCORE** $= 2\frac{PRECISION*RECALL}{PRECISION+RECALL}$

The Accuracy is a good metric if the classes are not too unbalanced, so in our case the F1-Score is more appropriate.

4.3. Support Vector Machine

The Support Vector Machine is a model used for linearly separable problems. The algorithm aims to find an hyperplane that well separates the two data distributions with a constraint: maximize the *margin*, that is, the distance between the plane and the nearest points of the data distributions. In the hard SVM formulation, the optimization problem is:

$$minimize_{w,b}\ \frac{1}{2}||w|\,|^2\ subject\ to\ y_i[< x_i, w > +b] \geq 1$$

and this can be solved using the quadratic programming approach, obtaining as solutions $\hat{w} = \frac{w_0}{||w_0||}$ and $\hat{b} = \frac{b_0}{||b_0||}$. Since the pure linear separability is not common, there is a soft variation of this problem: the Soft margin SVM, in which we allow some errors in classifications in the margin region. For this purpose *slack variables* are introduced in the optimization problem. The optimization problem becomes, for m points:

$$minimize_{w,b}\ \lambda\,||w|\,|^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i\ subject\ to\ y_i[< x_i, w > +b] \geq 1-\xi_i,\ \xi_i \geq 0$$

For each sample there is a slack variable that express a measure of how much errors are allowed. The λ hyperparameter is a regularization term. We call C the inverse of λ. Furthermore, there are cases in which we want to change the feature space in order to make the problem linearly separable. In such case, we can use the **kernel SVM** in which we project the features onto a new feature space but just using a kernel function that easily express the dot product in this new space without worries about the explicit spaces mapping.

$$K(x_1, x_2) = \phi(x_1)^T \phi(x_2)$$

. In particular, we consider two kernel functions:

- **linear kernel**

$$K(x_1, x_2) = x_1 \dot{x}_2$$

- **radial basis (RBF) kernel**

$$K(x_1, x_2) = exp(-\gamma(||x_1 \dot{x}_2||^2)$$

The $gamma$ in the RBF is an hyperparameter that affect the notion of "nearness" between points (it determines when two points should be considered near or not). For our grid search we choose these sets of parameters:

- Linear kernel
  - C: 0.1, 1, 10
- RBF Kernel
  - C: 0.01, 0.1, 1, 10, 100
  - gamma: 0.01, 0.1, 1, 10, 100

In the Table 2 we can see the best parameters for each configuration of the dataset.

#	Configuration	Best parameters	F1-Score
1	Full dataset	linear kernel, C=10	0.752
2	Reduced Dataset with PCA	linear kernel, C=10	0.752
3	Reduced Dataset with undersampling	RBF Kernel, C=10, gamma=0.01	0.828
4	SMOTE	RBF Kernel, C=100, gamma=0.01	0.753

Table 2: Best results of grid searches on SVM

We can easily see that there are no much differences between the full dataset and the one reduced using PCA. The 3rd configuration is the one that has the best performance.

We can also take a look at the confusion matrices obtained using these hyperparameters, in Table 3:

Configuration 1: full dataset		
	Predicted No Death	**Predicted Death**
True No Death	33 (82.5%)	7 (17.5%)
True Death	7 (35%)	13 (65%)

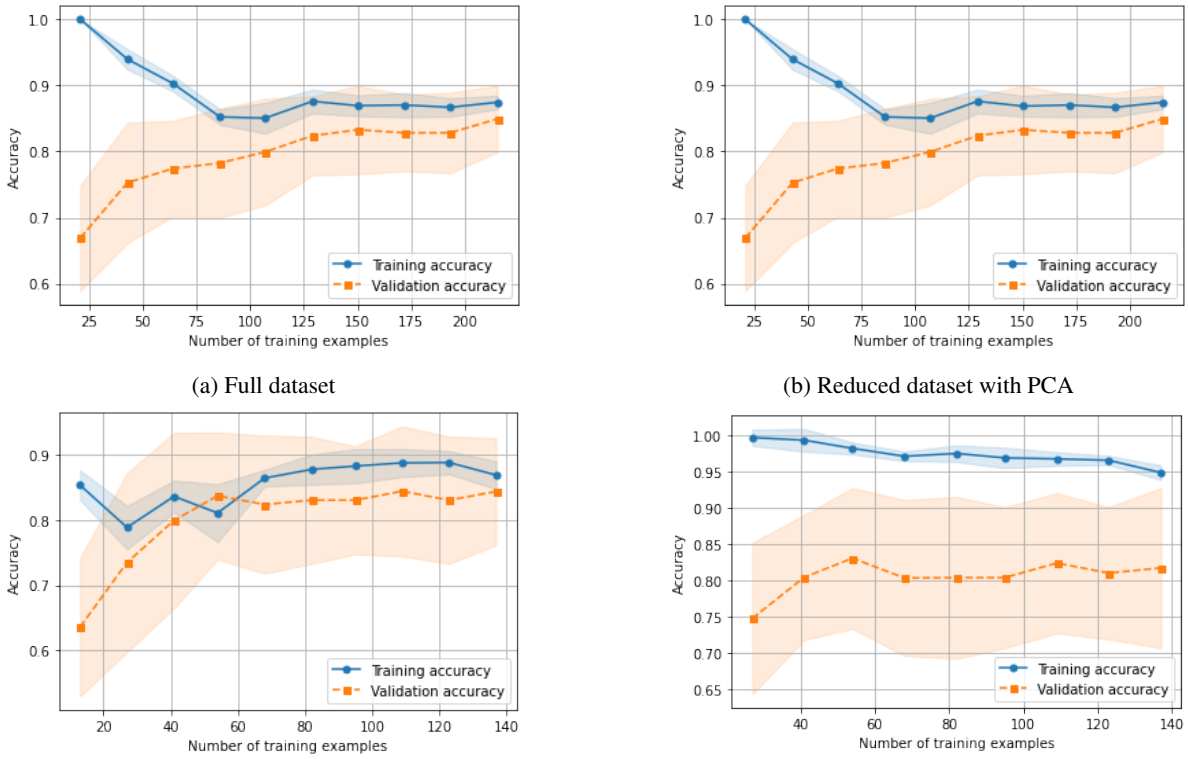
Configuration 2: reduced dataset with PCA		
	Predicted No Death	**Predicted Death**
True No Death	28 (70%)	12 (30%)
True Death	14 (70%)	6 (30%)

Configuration 3: reduced dataset with undersampling		
	Predicted No Death	**Predicted Death**
True No Death	16 (84.2%)	3 (15.8%)
True Death	5 (25%)	15 (75%)

Configuration 4: SMOTE		
	Predicted No Death	**Predicted Death**
True No Death	27 (67.5%)	13 (32.5%)
True Death	7 (35%)	13 (65%)

Table 3: Confusion Matrices for SVM

Looking at the confusion matrices, we can see that Configuration 1 and 2 have different behaviours even if the F1-Score is the same, in particular we have better results using the full dataset. According to our expectations, the Configuration 3 has the best scores. We can observe the learning curves of the models in Figure 7



(a) Full dataset

(b) Reduced dataset with PCA

(c) Reduced dataset with undersampling

(d) SMOTE

Figure 7: Learning curves for SVM

The training and validation curves are almost good for the first three configurations. Instead, for SMOTE configuration (the 4th), there is a big gap between them: it means that there is an high variance in the data and that there is overfitting (good performance in training set, poor performance in validation set).

4.4. Logistic regression

The logistic regression model is based on the logistic function to evaluate the probability for a sample to belong to a class. This model is based on the linear regression (find a function that good interpolates the data) and then use it to obtain a probability using a S-shaped function (like sigmoid). Given an input vector $x \in \mathbb{R}^n$ and a set of parameters $= (\beta_1, ..., \beta_n)^T$, the logistic regression model is defined as

$$P(Y = 1|x_i) = \frac{e^{\beta_0 + x_i^T \beta}}{1 + e^{\beta_0 + x_i^T \beta}} = \frac{1}{1 + e^{\beta_0 + x_i^T \beta}}$$

The hyperparameters that we choose for our grid search are:

- penalty (the norm used for penalization): l1 or l2
- C (the inverse of the regularization strength): 0,1, 1, 10

In the Table 4 we can see the best parameters for each configuration of the dataset.

#	Configuration	Best parameters	F1-Score
1	Full dataset	l2, C=0.1	0.738
2	Reduced Dataset with PCA	l2, C=0.1	0.738
3	Reduced Dataset with undersampling	l2, C=0.1	0.838
4	SMOTE	l2, C=10	0.716

Table 4: Best results of grid searches on Logistic Regression

The first two configurations have the same F1-Score and the best one is the 3rd.
We can also take a look at the confusion matrices obtained using these hyperparameters, in Table 5:

Configuration 1: full dataset		
	Predicted No Death	**Predicted Death**
True No Death	29 (72.5%)	11 (27.5%)
True Death	6 (30%)	14 (70%)

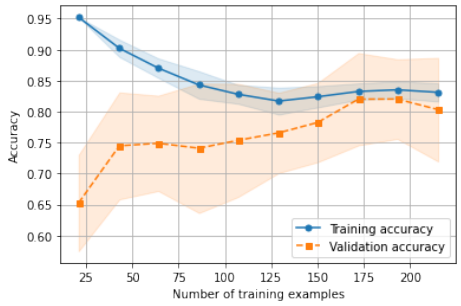
Configuration 2: reduced dataset with PCA		
	Predicted No Death	**Predicted Death**
True No Death	28 (70%)	12 (30%)
True Death	6 (30%)	14 (70%)

Configuration 3: reduced dataset with undersampling		
	Predicted No Death	**Predicted Death**
True No Death	14 (73.7%)	5 (26.3%)
True Death	5 (25%)	15 (75%)

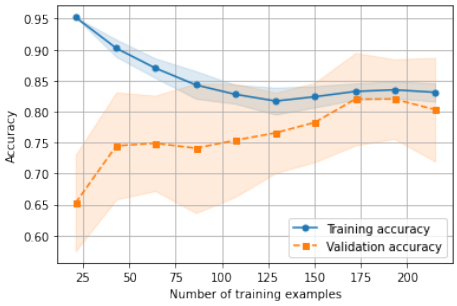
Configuration 4: SMOTE		
	Predicted No Death	**Predicted Death**
True No Death	28 (70%)	12 (30%)
True Death	6 (30%)	14 (70%)

Table 5: Confusion matrices for Logistic Regression

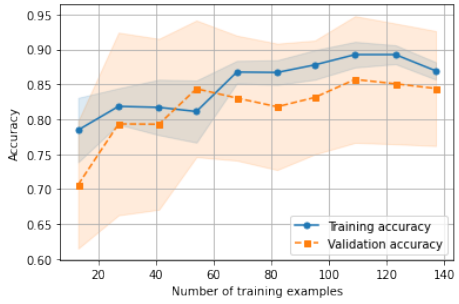
In this case, the confusion matrices of configuration 1, 2 and 4 are similar (the 4th is exactly the same of the 1st).
We can observe the learning curves of the models in Figure 8. The training and validation curves seem quite good, there is not a big gap among the training and validation curves.



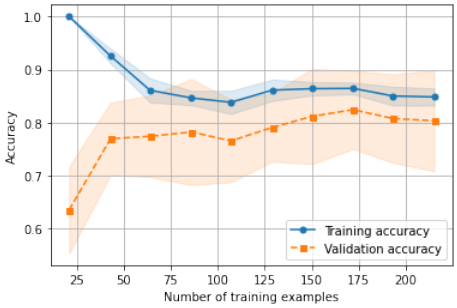
(a) Full dataset



(b) Reduced dataset with PCA



(c) Reduced dataset with undersampling



(d) SMOTE

Figure 8: Learning curves for Logistic Regression

4.5. K Nearest Neighbour

The K-Nearest Neighbour is a classification model that works considering only distance metrics. It assigns the class label basing only on a neighbourhood of K samples by majority voting. This model works well for multi-class classification tasks but suffers of the curse of dimensionality, because the higher the dimension the higher the density of the data in order to keep the same number of points at a given distance. The distances can be computed using the Monkowski metric:

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

where n is the number of dimensions. The common values of p are

- p=1: Manhattan distance
- p=2: Euclidean distance

The hyperparameters that we choose for our grid search for this model are:

- K (the number of neighbours: if it is too high, we can overfit; if it is too low, we can bee to generic): 5, 7, 9, 13
- Weights of the votes:
  - uniform: the votes of all neighbours have the same importance
  - distance: the votes are multiplied by the inverse of the distance
- p (the power of the Minkowski metric): 1, 2, 10.

In the Table 6 we can see the best parameters for each configuration of the dataset.

#	Configuration	Best parameters	F1-Score
1	Full dataset	uniform, K=5, p=1	0.474
2	Reduced Dataset with PCA	distance, K=9, p=10	0.477
3	Reduced Dataset with undersampling	uniform, K=13, p=2	0.737
4	SMOTE	uniform, K=9, p=2	0.628

Table 6: Best results of grid searches on KNN

The first two configurations have the worst performance, instead the 3rd is the best one, followed by the 4th. We can also take a look at the confusion matrices obtained using these hyperparameters, in Table 7:

Configuration 1: full dataset		
	Predicted No Death	**Predicted Death**
True No Death	38 (95%)	2 (5%)
True Death	14 (70%)	6 (30%)

Configuration 2: reduced dataset with PCA		
	Predicted No Death	**Predicted Death**
True No Death	32 (80%)	8 (20%)
True Death	15 (75%)	5 (25%)

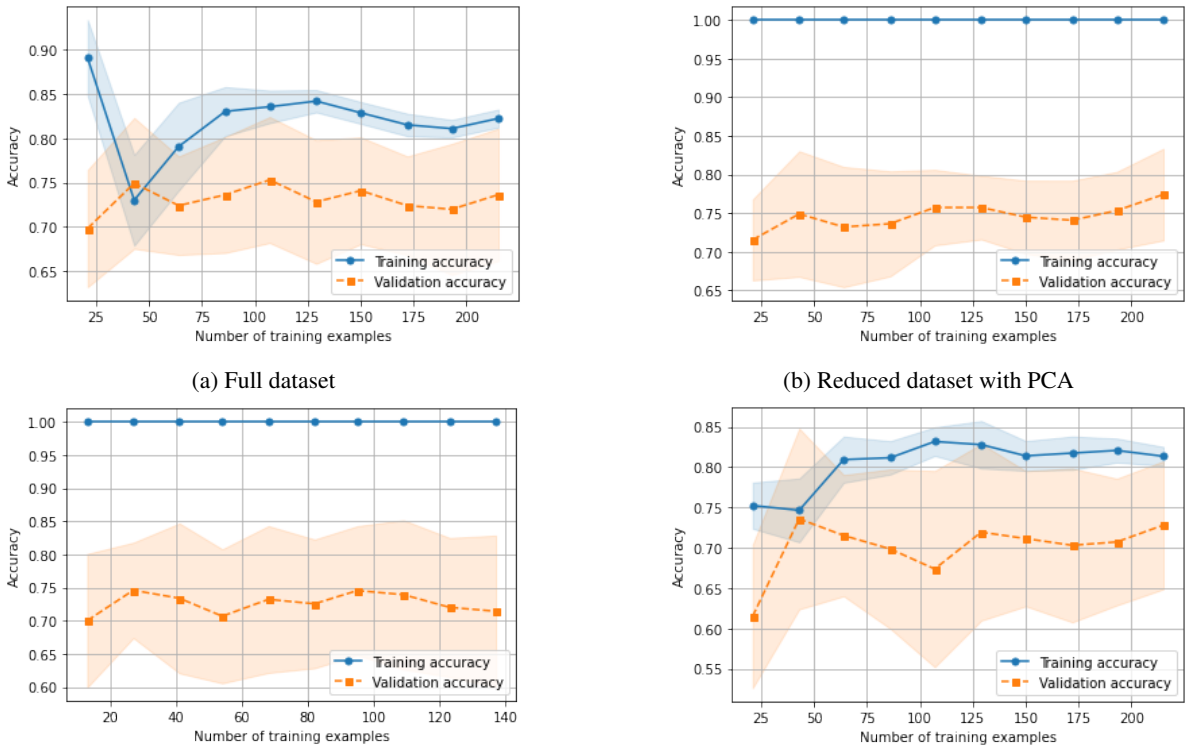
Configuration 3: reduced dataset with undersampling		
	Predicted No Death	**Predicted Death**
True No Death	13 (68.4%)	6 (31.6%)
True Death	9 (45%)	11 (55%)

Configuration 4: SMOTE		
	Predicted No Death	**Predicted Death**
True No Death	24 (60%)	16 (40%)
True Death	7 (35%)	13 (65%)

Table 7: Confusion matrices for Logistic Regression

The confusion matrices highlight the limits of the KNN method: the first two configurations have a very huge number of false prediction of "no death". This because of the unbalancing of the two classes. Indeed, this phenomenon doesn't happen in the 3rd and 4th configurations in which we did something to balance the two classes. This problem is highlighted in KNN since it works basing only on distances and on majority voting: if there is an high unbalancing we surely have more neighbours that votes for a label against the other one.

This issue is also present in the learning curves (Figure 9).



(a) Full dataset

(b) Reduced dataset with PCA

(c) Reduced dataset with undersampling

(d) SMOTE

Figure 9: Learning curves for KNN

There is an huge gap between training and validation curves, especially for configuration 2 and 3. This is due to the unbalancing for configuration 2 and to the limited amount of samples for configuration 3, in which we have a reduced dataset. So, at the end, the best model seems to be the 4th that uses SMOTE.

4.6. Decision Tree

The decision tree model is based on splitting the space of the data into boxes in order to minimize a criterion about the purity of the split of the data. This minimization problem is NP-hard, so we use a top-down greedy approach in order to do a recursive split considering the best solution in a specific step (and there are no guarantees that the obtained solution is the best one even if we did the best split at each step). In order to evaluate the splits, we can use two metrics:

- **Information Gain**: it measures the reduction of the entropy $H(t) = -\sum_{i=1}^{j} p_i log_2 p_i$. If we split n samples in a parent node p into k partitions (each one containing n_i samples), the information gain is defined as $IG = H(p) - \sum_{i=1}^{k} \frac{n_i}{n} H(i)$
- **Gini index**: it measures the impurity of the node. Given j as the number of classes, p_i as the fraction of items of class i in a subset p, for $i \in 1, 2, ..., j$, then the Gini index is defined as $I_G(p) = 1 - \sum_{i=1}^{j} p_i^2$

The hyperparameters that we choose for our grid search for this model are:

- criterion: gini or entropy
- max_depth (the max depth for each tree): 25, 50, 75
- min_samples_split (the minimum number of samples in a node to be considered for another splitting): 6, 10, 14

In Table 8 we can see the best parameters for each configuration of the dataset.

#	Configuration	Best parameters	F1-Score
1	Full dataset	entropy, max_depth: 75, min_samples_split:14	0.674
2	Reduced Dataset with PCA	entropy, max_depth: 75, min_samples_split:14	0.596
3	Reduced Dataset with undersampling	gini, max_depth: 75, min_samples_split:14	0.827
4	SMOTE	gini, max_depth: 25, min_samples_split:6	0.616

Table 8: Best results of grid searches on SVM

The model with the best score is the 3rd, the worst is the 2nd. The 1st and the 4th have similar values.

We can also take a look at the confusion matrices obtained using these hyperparameters, in Table 9:

Configuration 1: full dataset		
	Predicted No Death	**Predicted Death**
True No Death	33 (82.5%)	7 (17.5%)
True Death	5 (25%)	15 (75%)

Configuration 2: reduced dataset with PCA		
	Predicted No Death	**Predicted Death**
True No Death	25 (62.5%)	15 (37.5%)
True Death	5 (25%)	15 (75%)

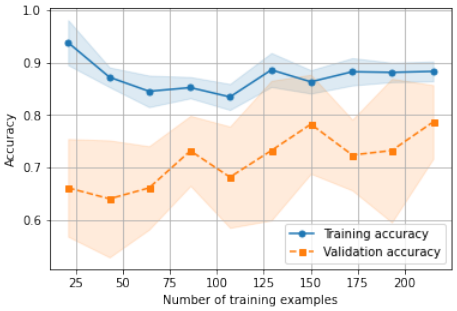
Configuration 3: reduced dataset with undersampling		
	Predicted No Death	**Predicted Death**
True No Death	13 (68.4%)	6 (31.6%)
True Death	10 (50%)	10 (50%)

Configuration 4: SMOTE		
	Predicted No Death	**Predicted Death**
True No Death	33 (82.5%)	7 (17.5%)
True Death	5 (25%)	15 (75%)

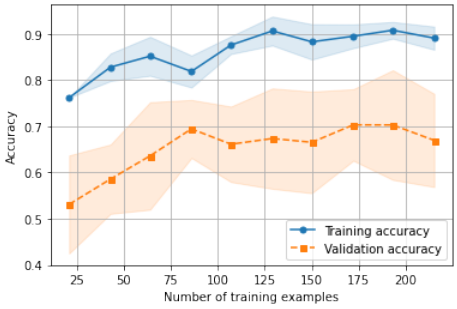
Table 9: Confusion matrices for Decision Tree

The confusion matrices of the 1st and 4th configurations are the same, instead the confusion matrix of the 2nd configuration has more miss-classification for Death class. The 3rd has very good performance but on a subset of the data.

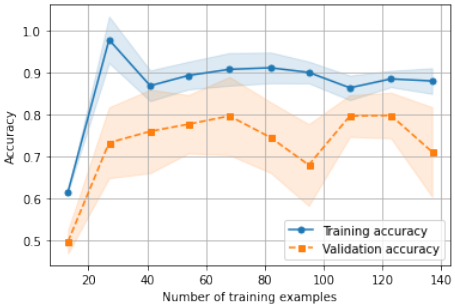
We can observe the learning curves of the models in Figure 10.



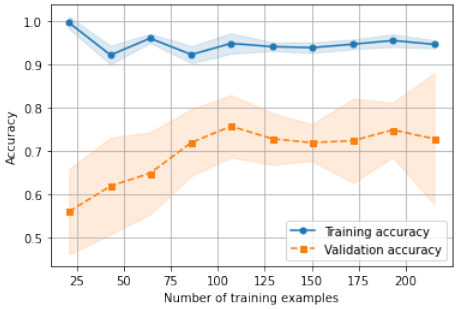
(a) Full dataset



(b) Reduced dataset with PCA



(c) Reduced dataset with undersampling

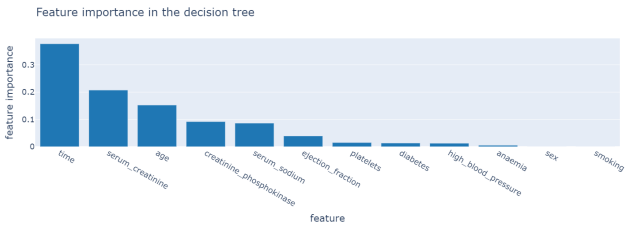


(d) SMOTE

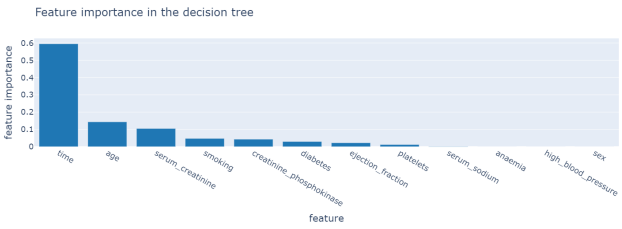
Figure 10: Learning curves for Decision Tree

The worst curves are the 2nd and the 4th, according to our expectations. Furthermore, this model potentially tends to overfit.

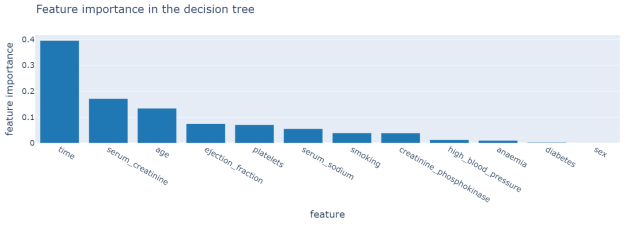
For decision trees we can also have an idea of the most important features involved in the decision process, looking at the charts in Figure 11. In this figure there is not the plot for the dataset reduced with PCA, since in this case the features are the Principal components.



(a) Full dataset



(b) Reduced dataset with undersampling



(c) SMOTE

Figure 11: Features importance for Decision Trees

We can see that the *time* feature is the most important one in all the configurations (and this is due to the observations done in the first section: there is a negative correlation between the target feature and *time*). The other important features are *serum_creatinine* and *age*.

4.7. Random Forest

The random Forest is an ensemble model that considers more uncorrelated trees in order to do a prediction. In particular, while a decision tree is built, each time a split is considered, a random selection m of the p predictors is chosen (usually $m = \sqrt{p}$) in order to do the split.

The hyperparameters that we choose for grid search for this model are:

- n_estimators (number of trees): 100, 300, 500, 1000
- criterion to split: gini or entropy
- max_depth (the max depth for each tree): 25, 50, 75

In the Table 10 we can see the best parameters for each configuration of the dataset.

#	Configuration	Best parameters	F1-Score
1	Full dataset	100 estimators, gini, max_depth: 25	0.741
2	Reduced Dataset with PCA	100 estimators, entropy, max_depth: 75	0.560
3	Reduced Dataset with undersampling	500 estimators, gini, max_depth: 5	0.821
4	SMOTE	1000 estimators, gini, max_depth: 75	0.779

Table 10: Best results of grid searches on Random Forest

The best model is the 3rd (and it's the only one that uses the entropy), instead the worst is the 2nd, The 1st and the 4th are similar.

In the Table 5 we can see the best parameters for each configuration of the dataset.

Configuration 1: full dataset		
	Predicted No Death	**Predicted Death**
True No Death	38 (95%)	2 (5%)
True Death	7 (35%)	13 (65%)

Configuration 2: reduced dataset with PCA		
	Predicted No Death	**Predicted Death**
True No Death	34 (85%)	6 (15%)
True Death	8 (40%)	12 (60%)

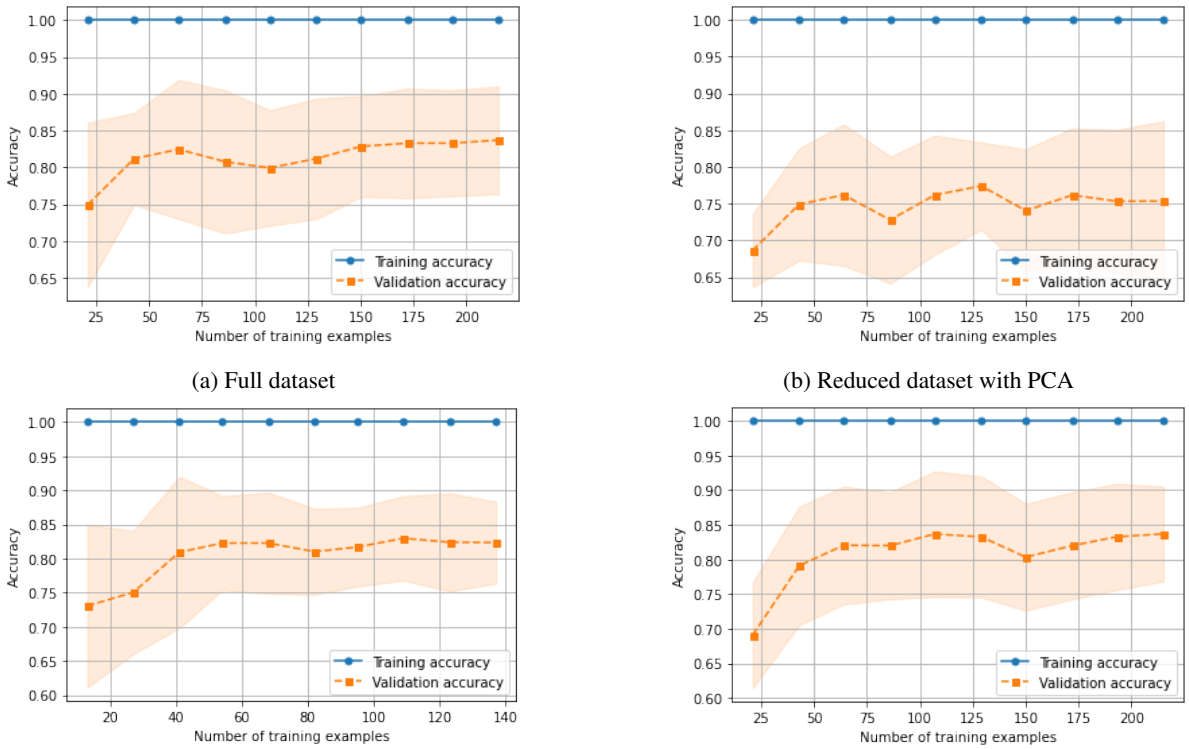
Configuration 3: reduced dataset with undersampling		
	Predicted No Death	**Predicted Death**
True No Death	12 (63.2%)	7 (36.8%)
True Death	5 (25%)	15 (75%)

Configuration 4: SMOTE		
	Predicted No Death	**Predicted Death**
True No Death	36 (90%)	4 (10%)
True Death	7 (35%)	13 (65%)

Table 11: Confusion matrices for Random Forest

The confusion matrices show good performance in all the cases. The 1st model is the one that has the highest percentages of True Positive and True Negative.

We can observe the learning curves of the models in Figure 12. The model tend to overfit a lot, with respect to the previous ones. This is probably due to the max_depth of the trees.



(a) Full dataset

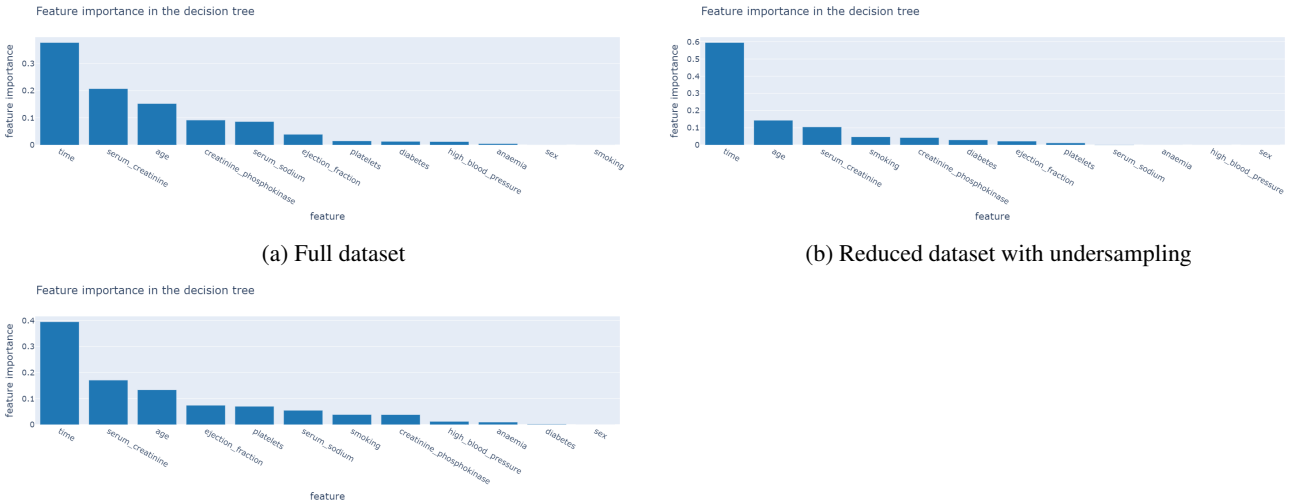
(b) Reduced dataset with PCA

(c) Reduced dataset with undersampling

(d) SMOTE

Figure 12: Learning curves for Random Forest

Also in this case we can looking at the charts in Figure 13 to understand the importance of each feature.



(a) Full dataset

(b) Reduced dataset with undersampling

(c) SMOTE

Figure 13: Features importance for Random Forest

We can see a similar trend with respect to decision trees: the *time* feature is the most important in all the configurations. The other important features are *serum_creatinine* and *ejection_fraction*.

5. Comparisons

In order to compare our models, we can plot the ROC curves and compare several metrics.

5.1. ROC curve

The **Receiver Operating Characteristic** Cuvers illustrate the performance of a binary classifier when its discrimination threshold is varied. The curve is plotted considering, at various thresholds

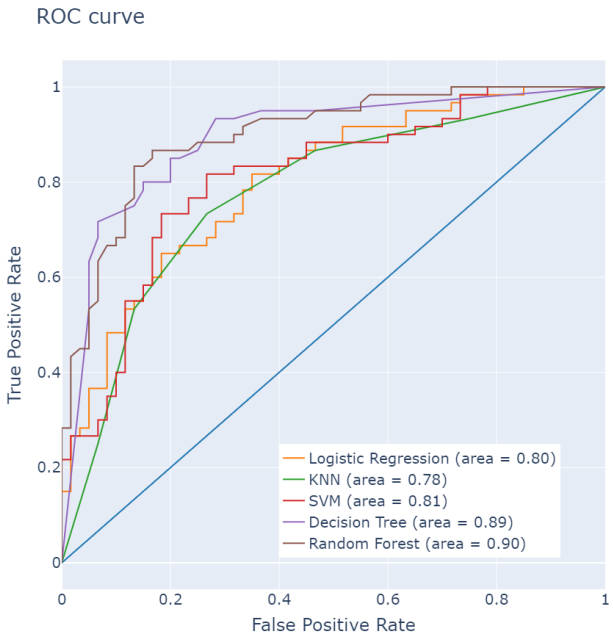
- The **True Positive Rate** $TPR = \frac{TP}{TP+FN}$
- The **False Positive Rate** $FPR = \frac{FP}{FP+TN}$

We can combine these two metrics into the **Area Under the ROC Curve (AUC)** metric: it measures the entire area under the ROC curve.
It considers all the thresholds expressing the probability that a model ranks a random positive sample more highly than a random negative sample. It is scale invariant (it is in the range [0,1]) and classification-threshold-invariant (the threshold is not important), but sometimes these properties are not good if we have a problem in which the scale is important or the false positive and false negative are not equally important (i.e. spam detection: you have to optimize the false positive even if this leads to have a lot of false negative).

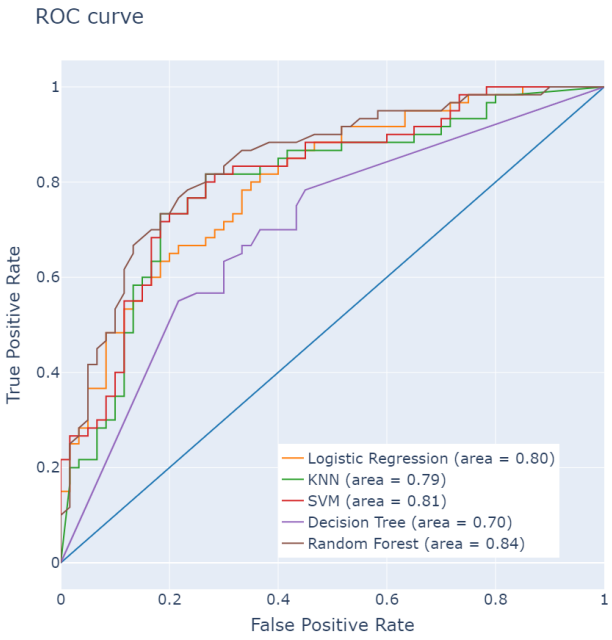
The ROC curve lives in a square of side 1, in which the diagonal represent a random guessing and the curves should be above this diagonal. The higher the area covered vy the model, the higher the performance.

In Figure 14 we can see the ROC curves for each configuration. No curves are below the diagonal, so there is no random guessing and all the models have learnt something.

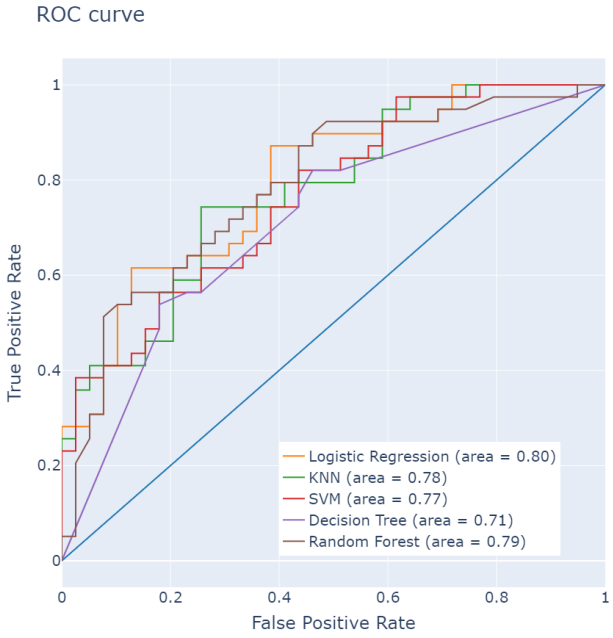
- Configuration 1 (entire dataset): The best models are the Random Forest and Decision Tree.
- Configuration 2 (reduced dataset with PCA): The best models are the Random Forest, SVM and KNN.
- Configuration 3 (reduced dataset with undersampling): The models are very similar except for Decision Tree. This chart seems to be the worst with respect to the others.
- Configuration 4 (SMOTE): The best models are the Random Forest and KNN. Furthermore, this seems to be the best chart with respect to others.



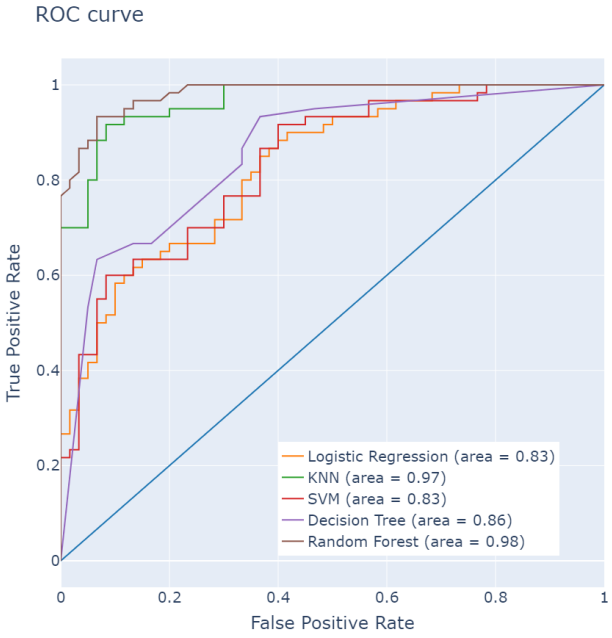
(a) Full dataset



(b) Reduced dataset with PCA



(c) Reduced dataset with undersampling



(d) SMOTE

Figure 14: ROC curves

5.2. Metrics comparison

In the Table 12 we compare the accuracy, precision, recall, f1-score and AUC for each model. For each metric the best values are in bold.

Configuration 1: full dataset					
	Accuracy	**Precision**	**Recall**	**F1-score**	**AUC**
Logistic Regression	0.717	0.560	0.700	0.622	0.802
KNN	0.733	0.750	0.300	0.429	0.776
SVM	0.767	0.650	0.650	0.650	0.809
Decision Tree	0.800	0.682	**0.750**	0.714	0.892
Random Forest	**0.850**	**0.867**	0.650	**0.743**	**0.899**

Configuration 2: Reduced dataset with PCA					
	Accuracy	**Precision**	**Recall**	**F1-score**	**AUC**
Logistic Regression	0.700	0.538	0.700	0.609	0.801
KNN	**0.767**	**0.800**	0.400	0.533	0.793
SVM	**0.767**	0.650	0.650	**0.650**	0.809
Decision Tree	0.667	0.500	**0.750**	0.600	0.700
Random Forest	**0.767**	0.667	0.600	0.632	**0.838**

Configuration 3: Reduced dataset with undersampling					
	Accuracy	**Precision**	**Recall**	**F1-score**	**AUC**
Logistic Regression	0.667	0.684	0.650	0.667	**0.797**
KNN	**0.744**	**0.812**	0.650	**0.722**	0.776
SVM	0.641	0.688	0.550	0.611	0.766
Decision Tree	0.590	0.625	0.500	0.556	0.710
Random Forest	0.692	0.682	**0.750**	0.714	0.785

Configuration 4: SMOTE					
	Accuracy	**Precision**	**Recall**	**F1-score**	**AUC**
Logistic Regression	0.700	0.538	0.700	0.609	0.829
KNN	0.617	0.448	0.650	0.531	0.966
SVM	0.667	0.500	0.650	0.565	0.832
Decision Tree	0.800	0.682	**0.750**	**0.714**	0.858
Random Forest	**0.817**	**0.765**	0.650	0.703	**0.982**

Table 12: Metrics comparison

- Configuration 1 (entire dataset): The best model is the Random Forest, that has the best scores in almost all the metrics.
- Configuration 2 (reduced dataset with PCA): The best models are the Random Forest, SVM and KNN, as we have already seen from the ROC curves. In particular, Random Forest and SVM have the best scores in the appropriate metrics for our problem: F1-score and AUC.
- Configuration 3 (reduced dataset with undersampling): The metrics replicate the ROC curves trends. In particular, we can see that KNN works very good in this configuration because of the balancing of the two classes.
- Configuration 4 (SMOTE): The best models are the Random Forest and Decision Tree. The AUC scores are, in general, very high for every model.

Looking purely at metrics, it seems that Random Forest is one of the best model for every configuration, even if we have seen that there is overfitting. Furthermore, for the dataset reduced with undersampling, the KNN works very good: this shows that the KNN is sensible to unbalancing.

6. Conclusion

Looking at all our results, we can say that, for the prediction of death of patients

- Even if the follow-up *time* feature is not a clinical feature, it has an huge importance in classification. In the features' correlation matrix we have observed a negative correlation between time and death and in both Decision Tree and Random Fores it is the most important feature used to classify samples.
- The PCA is not useful in this case: we have no improvements using it
- The unbalancing of the two classes is an issue, especially for KNN model. Indeed, the KNN works better in the 3-rd configuration in which the balancing is reached by doing an under-sampling on the majority class.
- The best results are reached by Random Forest model, even if it overfits on the data.In particular, the best AUC score is in the 4th configuration in which we balance the two classes using the SMOTE approach.
- Considering all the configurations, the best one is the 4th, that uses SMOTE. It leads to very good performance (with Random Forest we have the highest AUC) and all the models, also KNN, works well using it. This because the SMOTE technique solves the unbalancing adding samples in the dataset, instead of reducing them. Indeed, a cause of overfitting is the low amount of training data, so in general is not good to reduce samples as we did in configuration 3. So, this configuration surely helps the models to not overfit and to be more able to generalize. At the end, it seems to be the configuration more stable for all the models.

References

[1] D. Chicco and G. Jurman. Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. *BMC Medical Informatics and Decision Making*, 20(1):16, Feb 2020.