Thoracic Surgery

Politecnico di Torino

Christian Vincenzo Traina
s269575

Giuseppe Moscarelli
s270240

Introduction

The data is dedicated to classification problem related to the post-operative life expectancy in the lung cancer patients. In particular this dataset presents data of patients, attributes and whether they survive within one year of the thoracic operation. The goal is to understand if there is a way to determine the 1-year postoperative survival of lung cancer patients using patient attributes in the dataset. This could help patients and doctors assess the risks of the surgery and, therefore, decide whether to proceed or evaluate other alternatives.

Dataset description

The dataset is available at http://archive.ics.uci.edu/ml/datasets/Thoracic+Surgery+Data. According to the main repository site the data was collected retrospectively at Wroclaw Thoracic Surgery Centre for patients who underwent major lung resections for primary lung cancer in the years 2007-2011. The Centre is associated with the Department of Thoracic Surgery of the Medical University of Wroclaw and Lower-Silesian Centre for Pulmonary Diseases, Poland, while the research database constitutes a part of the National Lung Cancer Registry, administered by the Institute of Tuberculosis and Pulmonary Diseases in Warsaw, Poland.

The original dataset was in the form of a Weka ARFF file, and due to its incompatibility with the majority of tools and data science instruments, we decided to convert it into the CSV format.

Attributes description

The majority of the attributes is composed of binary values: there are 11 binary attributes that, in the original dataset, are expressed as two literal strings: *"T"* if the characteristic is present and *"F"* if it is absent.

Furthermore, there are 3 categorical attributes. All of them are ordinal attributes and the number contained in the suffix indicates the ordinal significance of the value. As an example, referring to the attribute *"Tumor_Size"*, the categorical values OC11, OC12, and OC14 can be sorted in ascending order.

Finally, to complete the dataset, there are 3 continuous attributes. An initial attributes description taken from the UCI machine learning repository site is shown below:

- **DGN**: Diagnosis - specific combination of ICD-10 codes for primary and secondary as well multiple tumours if any (DGN3, DGN2, DGN4, DGN6, DGN5, DGN8, DGN1)
- **PRE4**: Forced vital capacity - FVC (numeric). Amount of air which can be forcibly exhaled from the lungs after taking the deepest breath possible
- **PRE5**: Volume that has been exhaled at the end of the first second of forced expiration - FEV1 (numeric)
- **PRE6**: Performance status - Zubrod scale (PRZ2, PRZ1, PRZ0)
- **PRE7**: Pain before surgery (T, F)
- **PRE8**: Haemoptysis before surgery (T, F)

- **PRE9**: Dyspnoea before surgery (T, F)
- **PRE10**: Cough before surgery (T, F)
- **PRE11**: Weakness before surgery (T, F)
- **PRE14**: T in clinical TNM - size of the original tumour, from OC11 (smallest) to OC14 (largest) (OC11, OC14, OC12, OC13)
- **PRE17**: Type 2 DM - diabetes mellitus (T, F)
- **PRE19**: MI - Myocardial infarction (Heart Attack) up to 6 months before surgery (T,F)
- **PRE25**: PAD - peripheral arterial diseases (T, F)
- **PRE30**: Smoking (T, F)
- **PRE32**: Asthma (T, F)
- **AGE**: Age at surgery (numeric)
- **Risk1Y**: 1 year survival period - (T) value if died (T, F)

For the continuous features we also show a table containing some useful information about the distribution of the values:

	FVC	FEV1	Age
count	470.000000	470.000000	470.000000
mean	3.281638	4.568702	62.534043
std	0.871395	11.767857	8.706902
min	1.440000	0.960000	21.000000
25%	2.600000	1.960000	57.000000
50%	3.160000	2.400000	62.000000
75%	3.807500	3.080000	69.000000
max	6.300000	86.300000	87.000000

Dataset cleaning

We decided to make some changes in order to make the dataset eligible for the following methods for data augmentation and classification.
In particular, we encoded binary attributes containing the “T” and “F” literal string into the “1” and “0” numerical values respectively. We removed the id column, because it was useless to the aim of our analysis and could have impacted the performance of classifiers. Finally, we decided to rename the attributes of the original dataset in order to make them more understandable and easier to handle. The mapping is shown below:

DGN	→	Diagnosis
PRE4	→	FVC
PRE5	→	FEV1

PRE6 → Performance

PRE7 → Pain

PRE8 → Haemoptysis

PRE9 → Dyspnoea

PRE10 → Cough

PRE11 → Weakness

PRE14 → Tumor_Size

PRE17 → Diabetes_Mellitus

PRE19 → MI_6mo

PRE25 → PAD

PRE30 → Smoking

PRE32 → Asthma

AGE → Age

Risk1Yr → Death_1yr

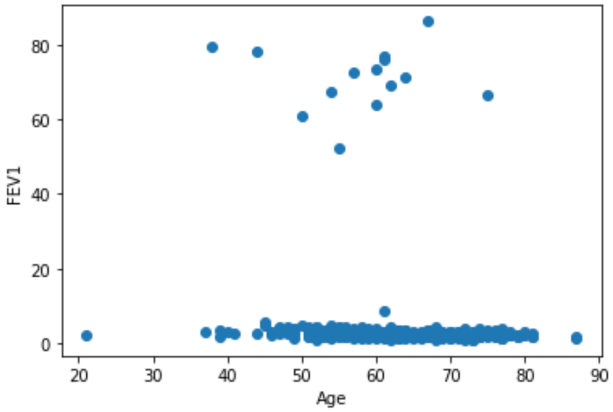
Missing Values

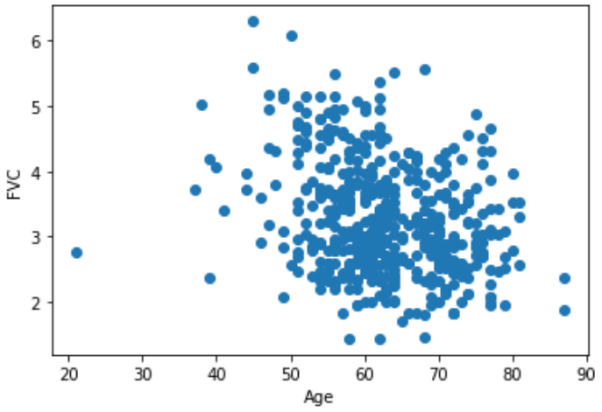
We verified the presence of missing values, and in our analysis we found that there are none.

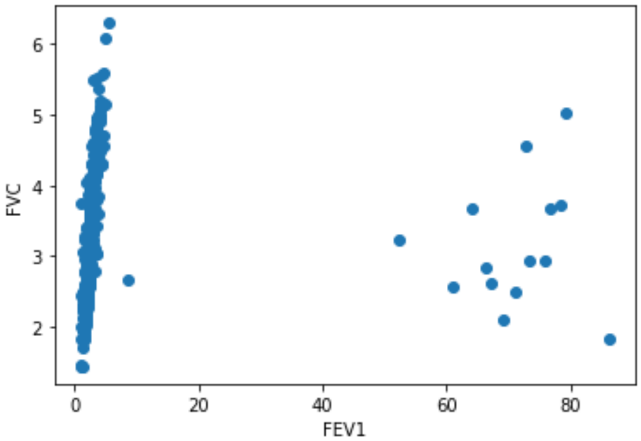
```
The number of missing values for each column is:
Diagnosis            0
FVC                  0
FEV1                 0
Performance          0
Pain                 0
Haemoptysis          0
Dyspnoea             0
Cough                0
Weakness             0
Tumor_Size           0
Diabetes_Mellitus    0
MI_6mo               0
PAD                  0
Smoking              0
Asthma               0
Age                  0
Death_1yr            0
dtype: int64
```

Outliers detection

For the outlier detection, we focused only on the three continuous attributes: FVC, FEV1 and Age. The scatter plots of these attributes show the presence of some outliers:





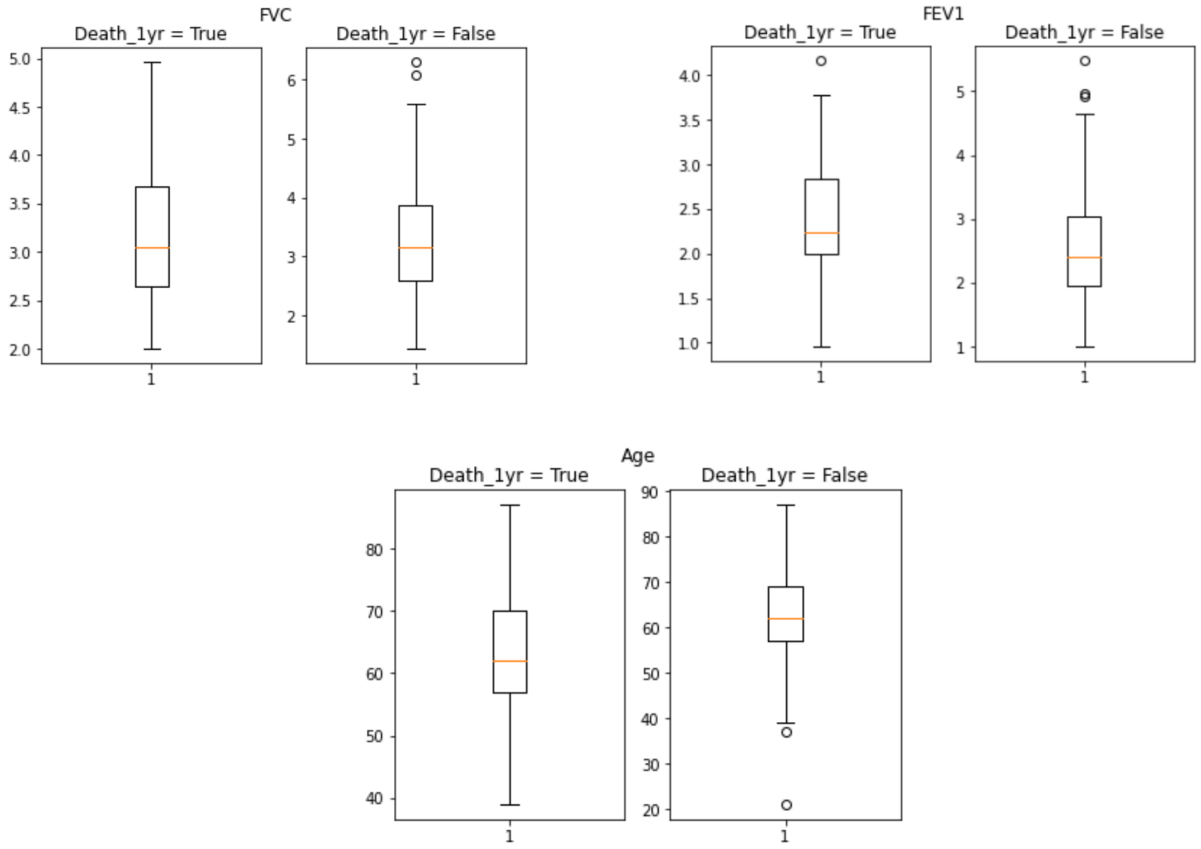


So we decided to remove all data with a FEV1 value greater than 7. In this way we removed 15 outliers. Observing the Age attribute it is possible to notice the presence of a possible outlier with a value of about 20. However, we decided to keep it since it is not much far from the other ages and we think it may contain important information for the purposes of classification.
After the outliers removal, the dataset now contains 455 instances against the starting 470.

The code for the above work can be found at:
https://github.com/GiuseppeMoscarelli/Thoracic-Surgery/blob/main/src/0_clean_dataset.ipynb

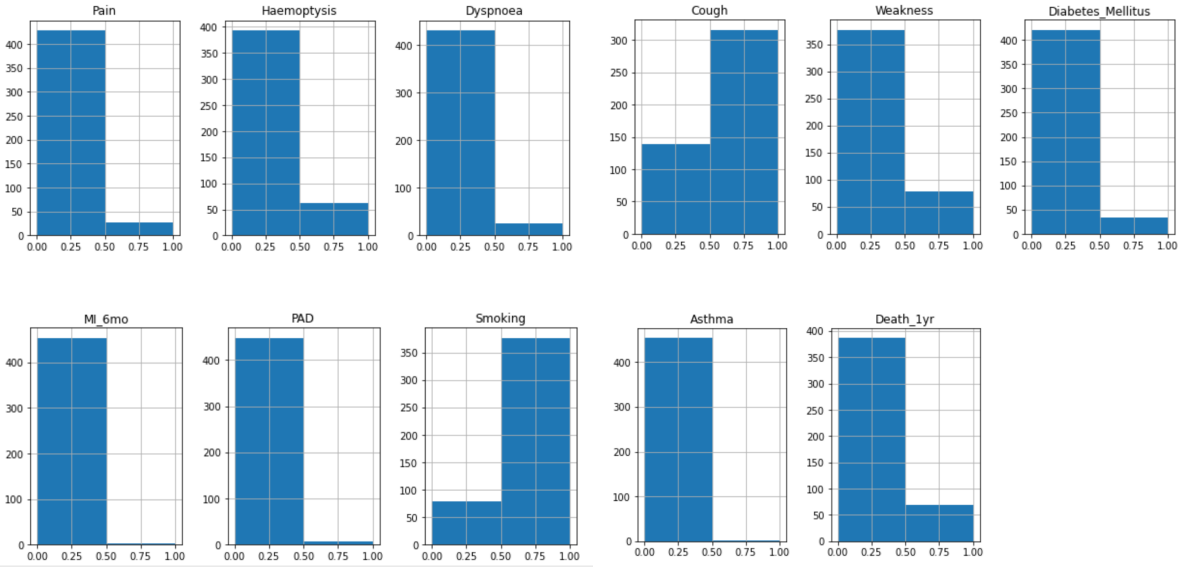
Data analysis

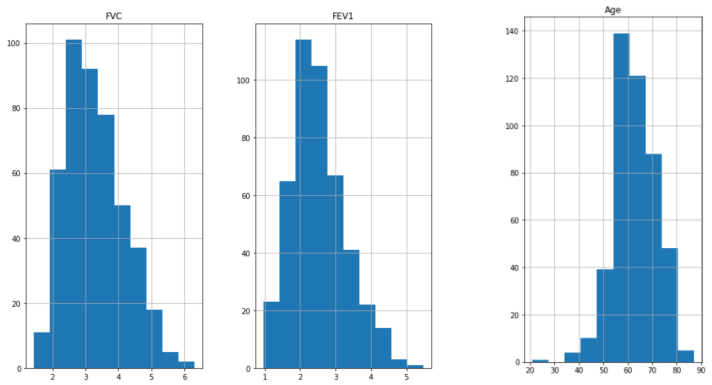
In order to retrieve some information on how the attributes influence the decision, we plotted the whisker boxplots of the continuous variables considering separately the cases in which the class label *"Death_1yr"* was positive and the cases in which it was negative. The figures are shown below:

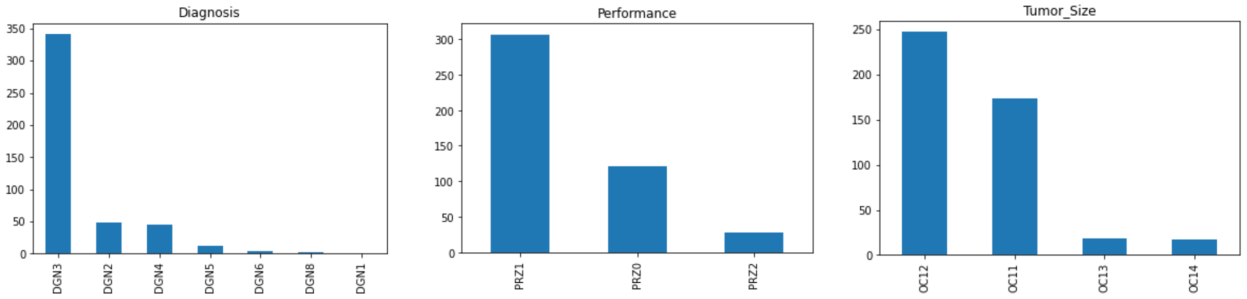


The result of the plotting shows that the “Age” attribute plays a more decisive role in the decision with respect to the other two continuous attributes, as there is a more marked inter-class difference between the assumed values. This denotes the fact that there is a correlation between *“Age”* attribute and the class label.

After that, in order to better visualize the distribution of all attributes, we plotted various histogram and bar graph:







Focusing on the label class *"Death_1yr"* we can note that the dataset is very unbalanced, since there are 69 positive samples and 368 negative samples. An analysis on unbalanced problems and the rebalancing solutions were explored in the next section.

In order to better understand the correlation between the different attributes, we used the correlation matrix (shown below). A **correlation matrix** is simply a table which displays the correlation coefficients for different variables. It is a powerful tool to summarize a large dataset and to identify and visualize patterns and relationships between the given data. The matrix has the features as rows and columns and the cells contain the correlation coefficients of the corresponding row and column.

	FVC	FEV1	Pain	Haemoptysis	Dyspnoea	Cough	Weakness	Diabetes_Mellitus	MI_6mo	PAD	Smoking	Asthma	Age	Death_1yr
FVC	1.000000	0.887629	0.002314	-0.103153	0.070447	-0.063979	-0.104872	-0.119992	-0.009675	-0.036955	-0.009885	-0.061944	-0.285144	-0.044533
FEV1	0.887629	1.000000	0.012448	-0.158192	0.024821	-0.132661	-0.127682	-0.110765	-0.034142	-0.032711	-0.051169	-0.078987	-0.295495	-0.072821
Pain	0.002314	0.012448	1.000000	0.225624	0.061907	-0.055571	-0.064880	0.034755	-0.016689	-0.033601	-0.081340	-0.016689	0.058561	0.075355
Haemoptysis	-0.103153	-0.158192	0.225624	1.000000	0.101023	0.068715	0.074306	0.008942	-0.026392	0.093096	-0.020890	-0.026392	0.085480	0.082120
Dyspnoea	0.070447	0.024821	0.061907	0.101023	1.000000	0.076163	-0.058494	-0.031843	-0.016021	0.114513	-0.042251	-0.016021	0.004298	0.113174
Cough	-0.063979	-0.132661	-0.055571	0.068715	0.076163	1.000000	0.200394	0.025165	0.044069	0.016118	0.199857	-0.028058	0.143508	0.094172
Weakness	-0.104872	-0.127682	-0.064880	0.074306	-0.058494	0.200394	1.000000	0.070334	0.057928	0.027890	0.116124	-0.030223	0.205173	0.084079
Diabetes_Mellitus	-0.119992	-0.110765	0.034755	0.008942	-0.031843	0.025165	0.070334	1.000000	-0.018883	0.025578	-0.046266	-0.018883	0.090474	0.112879
MI_6mo	-0.009675	-0.034142	-0.016689	-0.026392	-0.016021	0.044069	0.057928	-0.018883	1.000000	-0.008889	0.030457	-0.004415	-0.032134	-0.028093
PAD	-0.036955	-0.032711	-0.033601	0.093096	0.114513	0.016118	0.027890	0.025578	-0.008889	1.000000	0.061321	-0.008889	0.057025	0.036683
Smoking	-0.009885	-0.051169	-0.081340	-0.020890	-0.042251	0.199857	0.116124	-0.046266	0.030457	0.061321	1.000000	-0.057251	0.063131	0.080563
Asthma	-0.061944	-0.078987	-0.016689	-0.026392	-0.016021	-0.028058	-0.030223	-0.018883	-0.004415	-0.008889	-0.057251	1.000000	-0.020618	-0.028093
Age	-0.285144	-0.295495	0.058561	0.085480	0.004298	0.143508	0.205173	0.090474	-0.032134	0.057025	0.063131	-0.020618	1.000000	0.031636
Death_1yr	-0.044533	-0.072821	0.075355	0.082120	0.113174	0.094172	0.084079	0.112879	-0.028093	0.036683	0.080563	-0.028093	0.031636	1.000000

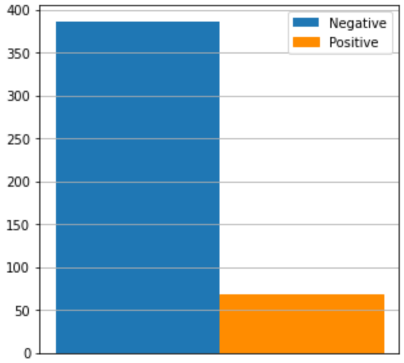
The obtained correlation matrix shows that there is a very strong correlation between FEV1 and FVC but also a mild negative correlation between Age-FVC, and Age-FEV1. The latter makes intuitive sense as it would be expected that as you get older, your lung capacity decreases.

The code for the above work can be found at:

https://github.com/GiuseppeMoscarelli/Thoracic-Surgery/blob/main/src/1_data_analysis.ipynb.

Dataset rebalancing

As previously mentioned, the dataset is very unbalanced. Indeed, there are 69 positive samples and 386 negative samples.



So, we needed to rebalance our dataset in order to improve the performance of the different algorithms and reduce the bias.

First of all, we split the entire dataset in training set and test set, assigning 70% of data points to the former and the remaining 30% to the latter, in a stratified way. This means that in the training and test sets, we have the same proportion of positive and negative samples as we had in the original dataset. Therefore, we trained the models using the training set and then validated the models on the test set.

After the splitting we obtain a training set composed of 318 samples and a test set of 137 samples.

As a second step, we rebalanced only the training set, leaving intact the test set. We decided to use two different techniques: **Random Oversampling** and **SMOTE**.

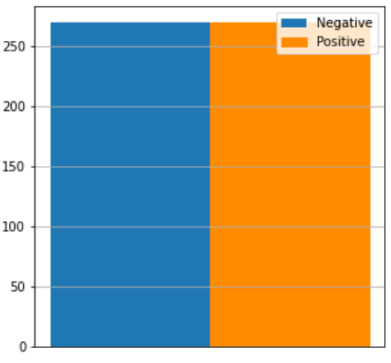
Random Oversampling

Random Oversampling is one of the main approaches to randomly resampling an imbalanced dataset. It consists in duplicating examples from the minority class, selecting them with replacement, 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. This approach is repeated until the desired class distribution is achieved in the training dataset, such as an equal split across the classes (in our case Negative and Positive).

So, starting with 270 negative samples and 48 positive samples in the training set, we obtained a balanced one containing exactly 270 negative and 270 positive samples.

```
Before Random Oversampling
Positive samples: 48
Negative samples: 270

After Random Oversampling
Positive samples: 270
Negative samples: 270
```



SMOTE

The Random Oversampling technique balances the class distribution but does not increase the variety of the dataset. An improvement over the duplication of the examples from the minority class is about synthesizing new examples from the minority class that were not present in the original dataset. This is a type of data augmentation for tabular data and can be very effective. The most widely used approach to synthesizing new examples is called **Synthetic Minority Oversampling Technique (SMOTE).**

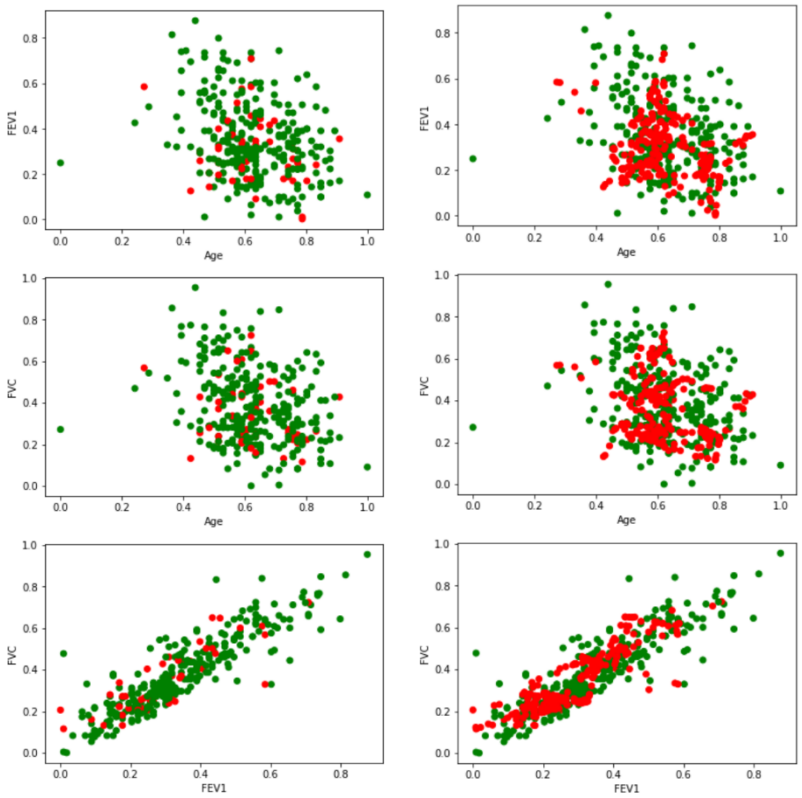
SMOTE works by utilizing a k-nearest neighbour algorithm to create synthetic data. First, it starts by choosing random data from the minority class, then the k-nearest neighbours from the data are selected, where **k** is an algorithm's parameter that was set to 5. Therefore, a randomly selected neighbour is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space. In other words, SMOTE first randomly selects a minority class instance **a** and finds its k nearest minority class neighbours. Among the k data points, one of them gets randomly selected and used to synthesize the new data point, we will refer to this point as **b**. The synthetic instance is thus created by connecting **a** and **b** to form a line segment in the feature space. In mathematical terms, the synthetic instances are generated as a convex combination of the two chosen instances **a** and **b**:

$$x_{new} = a + rand(0,1) * |a - b|$$

in which rand(0, 1) represents a real random number between 0 and 1.
The procedure is repeated enough times until the minority class has the same proportion as the majority class.

Because in our dataset there are categorical variables as well as continuous variables, we used a variation of SMOTE, called **SMOTE-NC.** This variant uses the continuous variables to determine the distribution of the categorical ones. We considered binary attributes as categorical since the application of SMOTE on these would have produced intermediate samples which did not make any sense at all.

Some examples of the dataset before (left) and after SMOTE-NC (right) application are shown below:

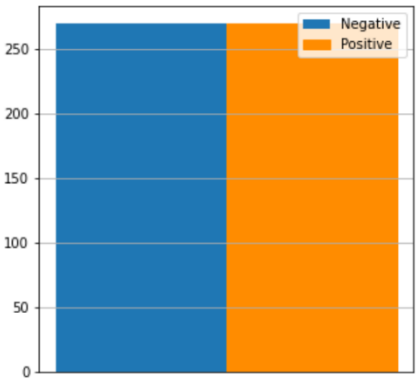


On the left there are three plots of the data before SMOTE-NC application, while on the right there are three plots of the data after SMOTE-NC application. Green points are those belonging to the negative class, while red points are those belonging to the positive class (minority class). As we can see in the figures representing the oversampled categories, there are many new generated samples belonging to the positive class which were not present before.

The number of positive and negative samples of the obtained new balanced training set, are the same as of balanced training set obtained from Random Oversampling:

```
Before SMOTE
Positive samples: 48
Negative samples: 270

After SMOTE
Positive samples: 270
Negative samples: 270
```



The code for the above work can be found at: https://github.com/GiuseppeMoscarelli/Thoracic-Surgery/blob/main/src/2_rebalancing_dataset.ipynb.

Principal Component Analysis

Principal Component Analysis (PCA) is a technique commonly used for dimensionality reduction by projecting each data point onto a lower-dimensional space, through a linear mapping, while preserving as much of the data's information as possible. This compressing procedure is called **data encoding**, while the reverse procedure is called **data decoding**. So PCA produces as a solution a linear mapping such that the distance between the original data and the reconstruction (encoding and then decoding) of the data is as small as possible. In mathematical terms, it is expressed in the following way:

$$\arg\min_{U,W} \sum_{i=1}^{m} \|\boldsymbol{x_i} - UW\boldsymbol{x}_i\|_2^2$$

where $W \in R^{n,d}$ and $U \in R^{d,n}$ are the matrices used for encoding and decoding respectively. Given *(U, W)* as a solution of the previous problem, it was demonstrated that the column of *U* are orthonormal and $W = U^T$. The PCA Theorem states that, given a data matrix *X*, which contains as lines the samples $x_1^T, \ldots, x_m^T \in R^d$, and given a matrix $A = X^T X$, called **scatter matrix**, the columns of *U* are the eigenvectors $u_1, \ldots, u_n$ associated to first *n* eigenvalues of *A*, ordered by magnitude ($\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq \lambda_{n+1} \geq \ldots \geq \lambda_p$).

The quantity $\lambda_{n+1} + \lambda_{n+2} + \ldots + \lambda_p$ is called **reconstruction error** and represents the distance between the original and the reconstructed point.

In terms of variance, the first principal component can be equivalently defined as a vector lying on the direction that maximizes the variance of the projected data. In other words, given a set of features $X_1, X_2, \ldots, X_p$, the first principal component is the normalized linear combination of the features:

$$PC_1 = \alpha_{11}X_1 + \alpha_{21}X_2 + \ldots + \alpha_{d1}X_d = \sum_{i=1}^{d} \alpha_{i1}X_i \qquad with \sum_i a_{i1}^2 = 1$$

and has the direction of largest variance on the feature space. After the first principal component PC_1 has been determined, we can find the second principal component PC_2 which is itself a normalized linear combination of the p features and has maximal variance among all linear combinations that are uncorrelated with PC_1.
It turns out that constraining PC_2 to be uncorrelated with PC_1 is equivalent to constraining the directions of the two principal components to be orthogonal. We can continue in this way until we find the *p*-th principal component.

In mathematical terms, the problem of PCA in term of variance can be formulated as:

$$PC_i = \max_{\|PC\|=1} Var(PC) \quad | \quad \langle PC, PC_j \rangle = 0 \ \ \forall\, j < i$$

So, in general, we can say that the *i*-th principal component can be taken as a direction orthogonal to the first *(i-1)*-th principal components that maximizes the variance of the projected data.

In order to estimate the variance explained by each component and to understand the importance of each component we have to compute the **PVE (Proportional Variance Explained)**.

Given a centered dataset *X* composed by *n* observations, the total variance is defined as:

$$\sum_{j=1}^{p} Var\left(X_j\right) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2$$

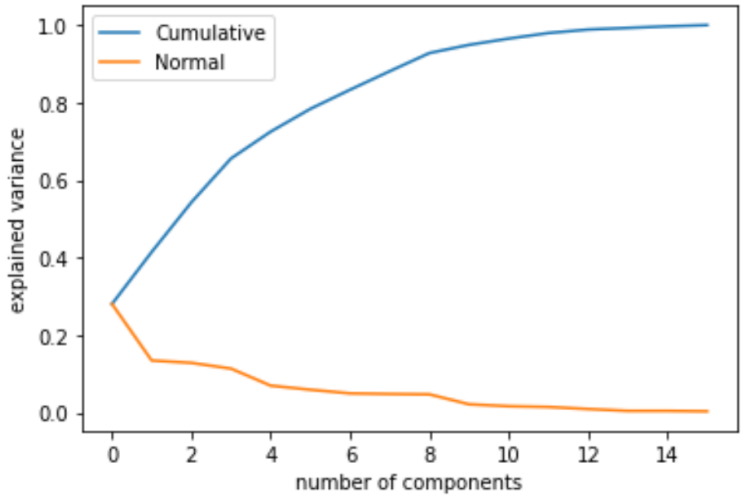
and the variance explained by the *m*-th principal component is:

$$Var(PC_m) = \frac{1}{n} \sum_{i=1}^{n} z_{im}^2$$

where z_{im} are the component of PC_m. Therefore, the **PVE** of the *m*-th principal component is given by the positive quantity between 0 and 1:

$$PVE_m = \frac{Var(PC_m)}{\sum_{j=1}^{p} Var\left(X_j\right)} = \frac{\lambda_m}{\sum_{i=1}^{p} \lambda_i}$$

The cumulative PVE is simply the incremental sum of PVE of different principal components.
To perform PCA on our dataset we removed the prefixes DGN, PRZ and OC from categorical variables and then we normalized them.
After plotting the values of cumulative and normal explained variance, we decided to take the first 9 principal components that explain a cumulative variance of almost 90%, as shown by following figure:



Finally, we applied SMOTE on the obtained PCA dataset. The choice to use SMOTE instead of SMOTE-NC was due to the fact that, this time, we had only continuous variables.

The code for the above work can be found at:
https://github.com/GiuseppeMoscarelli/Thoracic-Surgery/blob/main/src/3_PCA.ipynb.

Classification

Grid search cross validation

Grid search is an exhaustive search over a manually specified set of hyperparameter values for an estimator. In other words, it is a technique used to find a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process, thus it has to be tuned so that the model can optimally solve the machine learning problem.

In order to find the best hyperparameters values, in our work we used the **GridSearchCV** module of sklearn. It uses internally a 5-fold cross validation on the training set to measure the model performance with respect to two different alternative metrics:

$$recall = \frac{tp}{tp+fn}, \qquad accuracy = \frac{tn+tp}{tn+fp+fn+tp}$$

We performed grid search cross validation for all classifiers trained on five different datasets:

- ***with_outliers***: a normalized unbalanced dataset containing outliers
- ***no_oversampling***: a normalized unbalanced dataset without outliers
- ***random_oversampling***: a normalized balanced dataset using random oversampling technique, without outliers
- ***SMOTE***: a normalized balanced dataset using SMOTE-NC technique, without outliers
- ***PCA***: a normalized dataset on which we applied PCA and then SMOTE technique, without outliers

Decision tree

Our first step into classification was made using decision trees, since thanks to their interpretability they offer an insight on the classification task and let us dive into separability concerns. Despite their lower accuracy when compared to other classifiers, we considered it important since some hyperparameters tuned in this task were also used for the random forest algorithm.

As discussed in the previous section, we have already divided our dataset in train and test splits, for each of the different flavors of the dataset: retaining outliers, relieving outliers, using data augmentation (RO / SMOTE), and using PCA.

A **decision tree** is the composition of all the possible solutions generated splitting the datasets into two or more homogeneous sets. The operation of splitting into two or more homogenous sets is

repeated in a recursive fashion until all the leaves of the generated tree are totally homogenous or another stop condition is met (for example, if the maximum number of levels is reached).

We have different formulas to determine whether a split is homogenous or not, in our work we examined the two most used: **Gini Index** and **Entropy**.

The **Gini Index** is calculated by subtracting the sum of the squared probabilities of each class from one, and it can be expressed with the following formula:

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

Where $\hat{p}_{mk}$ indicates the proportion of training samples in the **m-th** region that are from the **k-th** class, while ***K*** is the number of classes.

On the other hand, **Entropy** is a concept taken from information theory, where it is defined as the average level of "information" in all possible outcomes of a variable.

The entropy can be expressed as:

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

Both Gini Index and Entropy are measures that indicate how heterogenous the set is, since we want our splits to be as homogenous as possible, we calculate the measures both before and after the split and we only select the split on the attributes and on the threshold that minimize the measure after the split (i.e. that maximize the gain, which is calculated as the difference between the information before the split and the information after the split).

We performed a **grid search cross validation** in order to find the best criterion between Gini Index and Entropy, altogether with other hyperparameters such as the maximum levels of the tree and the minimum number of samples to perform the split.

We used different grid search scoring metrics and different datasets, and in our experiments it turned out that the gini index performed better in almost all the cases. The best recall values were reached with the dataset with SMOTE oversampling, while it performed poorly on the other datasets, as shown in the following tables:

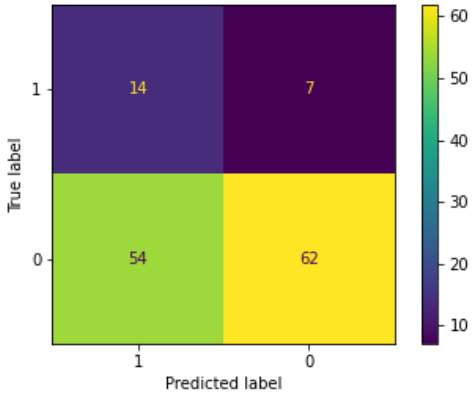
ACCURACY TABLE

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
DecisionTree	0.851064	0.846715	0.737226	0.678832	0.635036

RECALL TABLE

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
DecisionTree	0.428571	0.190476	0.142857	0.666667	0.428571

The confusion matrix with SMOTE is the following:



In this way we obtained a recall of:

$$recall = \frac{tp}{tp+fn} = \frac{14}{14+7} = 0.6667 = 66.67\%$$

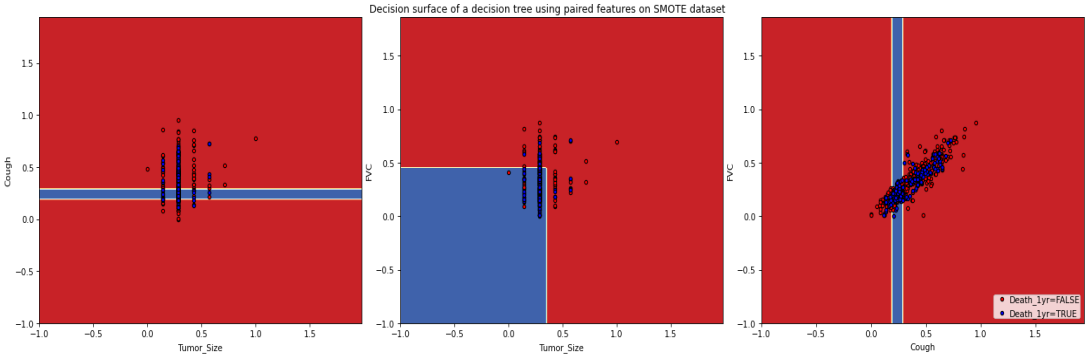
which is satisfying since it correctly classifies the majority of positive samples of the test dataset.

By contrast, the decision trees trained on the datasets with no oversampling or random oversampling performed poorly on the respective test datasets, reaching a recall of **19.04%** and **14.29%** respectively.

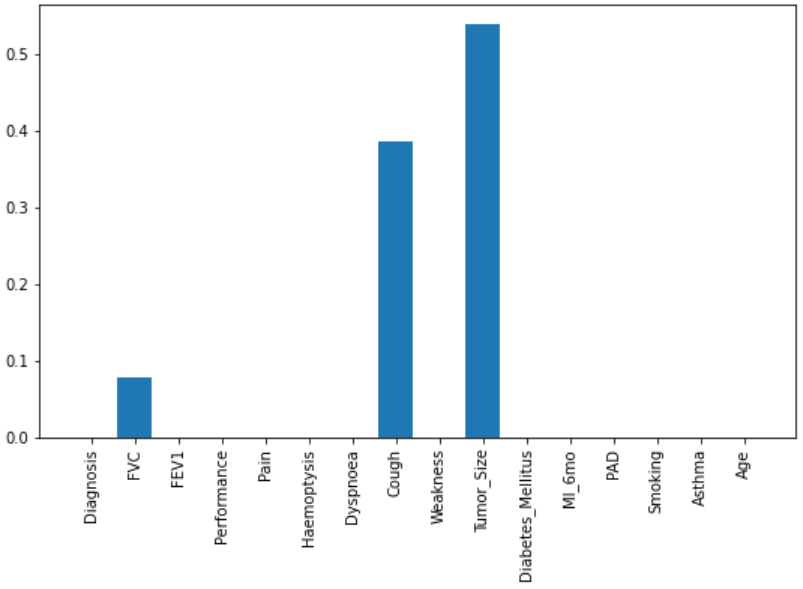
The accuracy score on the same datasets is, instead, high (**84.67%** and **73.72%** respectively). The accuracy reached by the decision tree with no oversampling and retaining outliers was the best, reaching a value of **85.11%**. However, since the test dataset is imbalanced as the oversampling was performed only on the train set, the accuracy value is misleading and could not bring valuable information in this context.

In the end, the decision tree trained on the SMOTE oversampled dataset performed well but not best. Indeed, decision trees are desirable for their explainability but they don't reach state-of-the-art performance. This is caused by the decision boundary which is constrained only by linear separators.

In order to provide a visual interpretation, in the following image is visible the decision boundary of the decision tree trained on the SMOTE dataset:

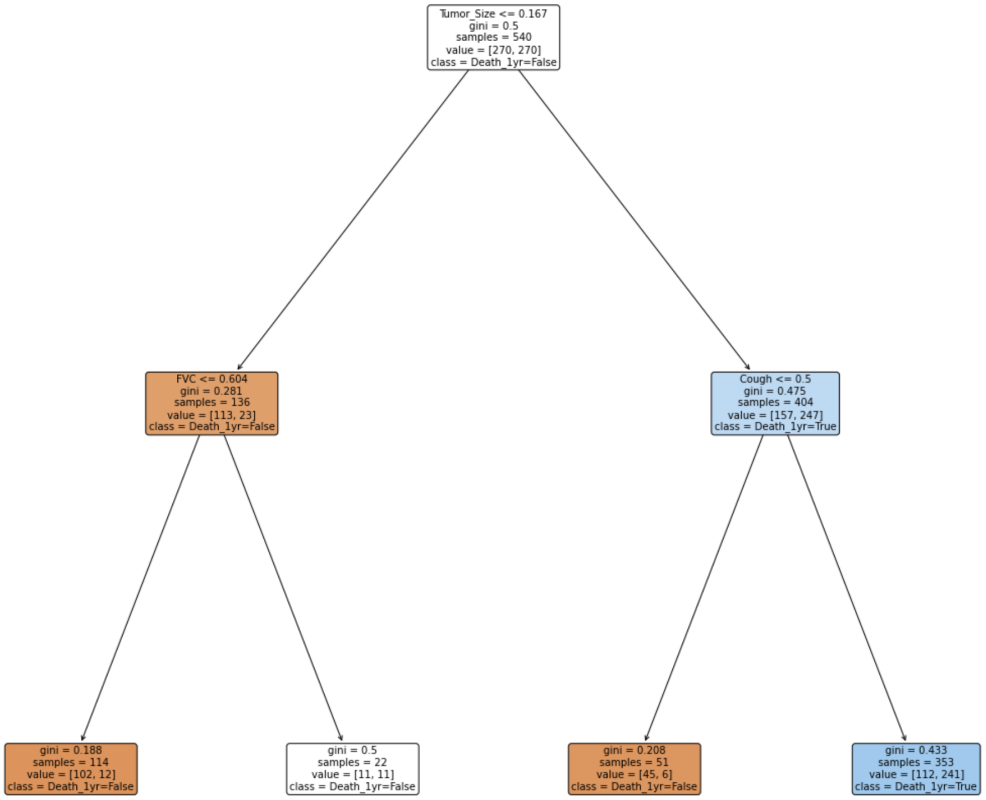


To plot the decision boundaries above, we previously extracted the feature importance for the tuned classifier, and took the pairs of the three most important features. The feature importance bar chart for the SMOTE dataset is visible in the following figure:



where it's clearly visible that the features *FVC*, *Cough* and *Tumor_Size* are not only the most relevant for this classifier, but the only three features the whole classifier is based on.

Another way to interpret the classifier and extrapolate concerns about the most important features is by reading directly the generated tree and inspecting the nodes:



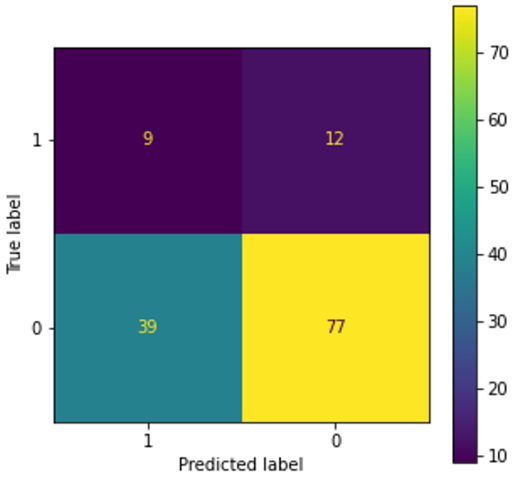
As it is visible exploring the nodes, there are only three levels and the leaves are not pure. Particularly, we have a leaf that is extremely impure, where 11 positive and 11 negative samples live together, thus the gini index value is **0.5**. Despite it does not perfectly fit the train set, in the grid search algorithm this tree performed best during cross validation.

As an example, we can take the following sample randomly taken from the test set:

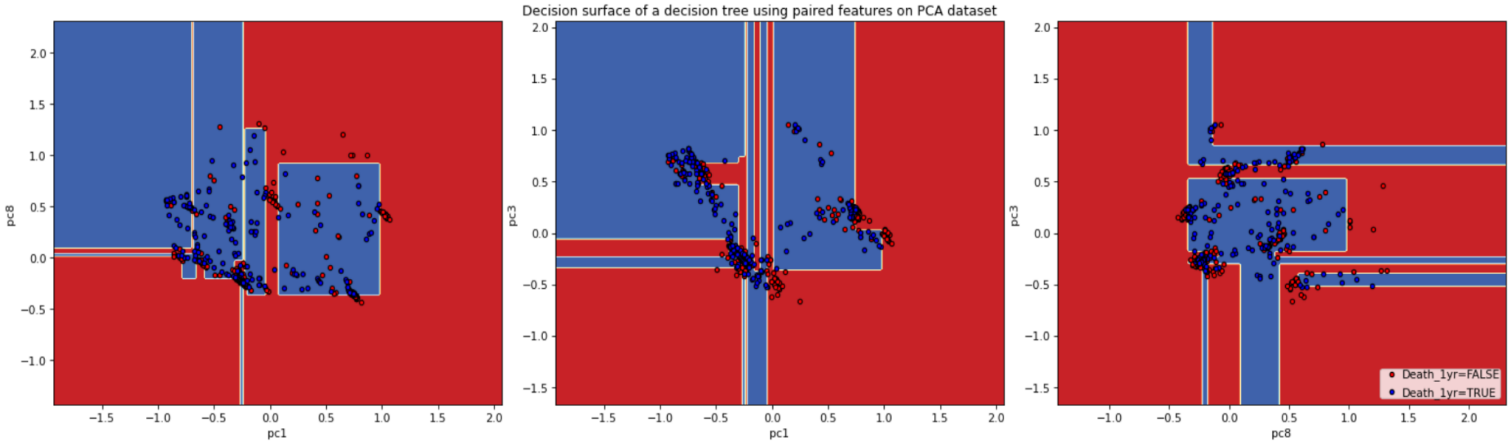
	Diagnosis	FVC	FEV1	Performance	Pain	Haemoptysis	Dyspnoea	Cough	Weakness	Tumor_Size	Diabetes_Mellitus	MI_6mo	PAD	Smoking	Asthma	Age	Death_1yr
135	DGN2	0.337449	0.336283	PRZ1	0	0	0	1	0	OC11	0	0	0	0	0	0.424242	0

As we can see it has a value of OC11 for the **Tumor_Size** feature, that was normalized to **0** and so **<0.167**, and a value of **0.34** for **FVC < 0.604.** Thus, finally, the sample reaches a leaf containing the label "*Death_1yr=False*", that matches to the value of *Death_1yr* of the ground truth sample.

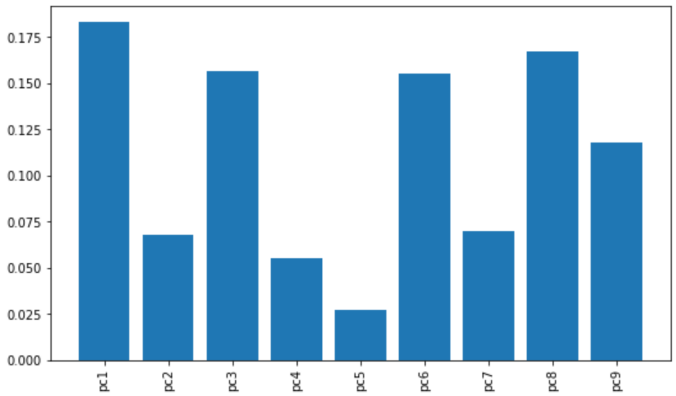
After the case of SMOTE dataset, the second best recall score was reached on the dataset with outliers and the one on which we applied PCA. Focusing on the latter, the obtained confusion matrix is shown below:



The decision boundaries of the decision tree trained on the PCA dataset is instead the following:

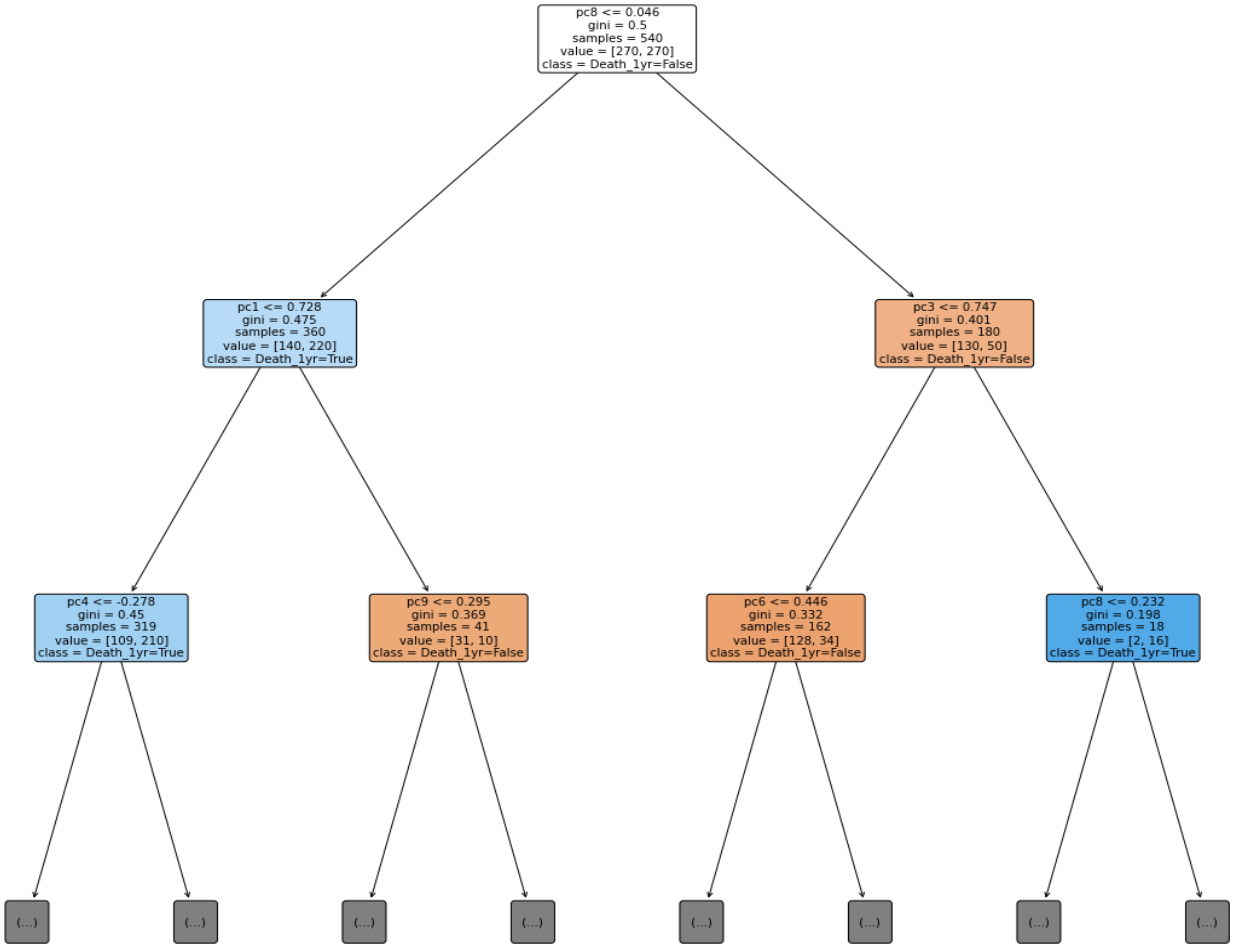


where the three most important principal components are used. The importance of the principal components is given by this bars chart:



So we selected the principal components 1, 3 and 8 in order to provide three meaningful decision boundaries. Notice that, this time, the three components are not exhaustive to explain the model, thus the misclassified points in the decision boundaries above are brought by the remaining not-plotted features.

In the end, we show the generated tree on this dataset:



The above tree is a truncated version of the whole tree, that was not included for its size and can be found in the notebook [at this address]().

Even if the PCA dataset got an interesting result for recall and accuracy, it raises some concerns about the interpretability.

As an example, we can take the following sample randomly taken from the test set:

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	pc9	Death_1yr
135	0.049141	0.534387	-0.465535	-0.536566	-0.116037	-0.271537	0.013658	0.14084	-0.001379	0

And since **pc8>0.046 AND pc1<=0.728**, we can fastly see that the sample is correctly classified both in the truncated and not truncated version. Indeed, the sample reaches a leaf containing the label "*Death_1yr=False*", and effectively *Death_1yr* was false in the ground truth sample. However we made an attempt in order to provide an interpretation of this result.

To do this, we extrapolated the eigenvectors from the principal components analysis, and we selected the eighth eigenvector which maps all the initial features, and which corresponds to the root of the tree. The eighth eigenvector has the following values:

```
array([ 0.08232354,  0.27052835,  0.27409081,  0.1742117 ,  0.46156494,
       -0.0109055 , -0.24688166,  0.0433472 , -0.07164708, -0.70806464,
        0.16551811,  0.00161269, -0.03349487,  0.04117668, -0.01015909,
       -0.04198546])
```

We can use this eigenvector to get the value of the eighth component using the following formula:

$$pc_8 = x^T v_8$$

Where ***x*** represents the features of a patient and $\boldsymbol{v_i}$ is the i-th eigenvector. In other words, the obtained eigenvector acts as weights for the features, and we can easily see that the 5th and 10th weights are the biggest in absolute value (respectively 0.461 and -0.708), thus the most impacting, and they correspond to the features "Pain" and "Tumor_Size".

Accordingly, the rule **pc8<=0.046** can be rewritten, in a more interpretable way, as **0.462*Pain - 0.708*Tumor_Size** (using an approximation that only considers two features).

Support Vector Machine

We decided to use support vector machine since it can achieve good performance also in high dimensional datasets and the use of kernels prevents us from making assumptions on the distribution of the data and it is thus versatile.

The aim of support vector machines is to find a hyperplane which is able to separate two classes. Differently to other linear classifiers, like perceptrons, the support vector machines will always find an optimal hyperplane which maximizes the margin, i.e. the minimal distance between the hyperplane and the closest points of the two classes.

The support vector machine classifier can be formalized in the following formula:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{maximize}}\, M \quad \text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \quad \forall\, i = 1, \ldots, n$$

Where **M** is the margin of the hyperplane as defined above, $\mathbf{x_{ij}}$ represents the points of the dataset, and $\boldsymbol{\beta_i}$ represents the weights to multiply the points and they are the values that we mean to find to optimize the problem. In other words, the optimization problem chooses $\boldsymbol{\beta_0}$, $\boldsymbol{\beta_1}$, ..., $\boldsymbol{\beta_p}$ to maximize **M**.

The formula above is valid in the case a separating hyperplane exists, and it's named "hard margin definition". Though, there are cases where a separating hyperplane does not exist, and we thus introduce the concept of "soft margin definition" in order to adapt the SVM problem:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \ \sum_{i=1}^{n} \epsilon_i \leq C,$$

This formulation is similar to the previous one, but we added the ϵ vectors which represents the slack variables in the optimization problem, and **C** is a non-negative parameter that is inversely proportional to how "soft" our support vector machine can be. For values of **C** that tend to infinity, the formulation gets closer to the hard margin one.

The parameter **C** has been the object of tuning in the grid search for this classifier.

There are cases where a soft margin formulation is not enough to get a high accuracy, due to the non-linear nature of the data. The kernel trick comes to our aid since it lets us project the data in a non-linear space just replacing the inner product to a kernel function.

The kernel trick can be applied just replacing the inner product each time it is needed in the SVM with a generalization of the inner product of the form:

$$K(x_i, x_{i'}),$$

Where **K** is a special function that we call kernel.

We used the **RBF kernel**, which inner product can be expressed in this way:

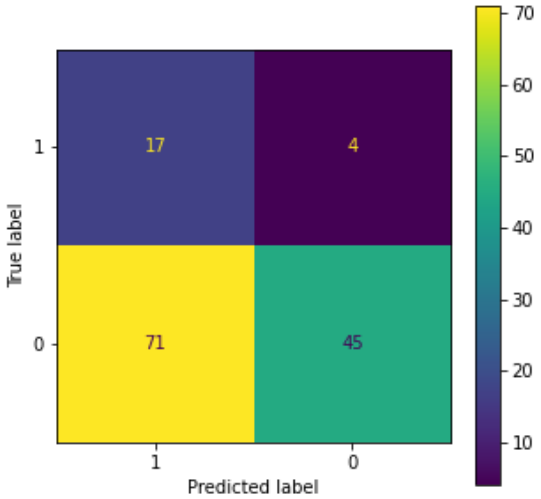
$$k(x_i, x_j) = \exp\left(-\frac{d(x_i, x_j)^2}{2l^2}\right)$$

In **SVC-RBF** is defined the gamma parameter as the inverse of the standard deviation of the RBF kernel, which is used as similarity measure between two points. A small gamma value defines a Gaussian function with a large variance. In this case, two points can be considered similar even if they are far from each other. On the other hand, a large gamma value means defining a Gaussian function with a small variance and in this case, two points are considered similar just if they are close to each other. Thus, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.

Also the gamma parameter was tuned using grid search.

Once again, the best value for recall was reached by the **SMOTE** dataset, which got a recall of **80.95%**.

Considering all the models trained on the **SMOTE** dataset, the best model found by grid search was achieved with a **C=0.1** and **gamma=0.001**, and the resulting confusion matrix is the following:

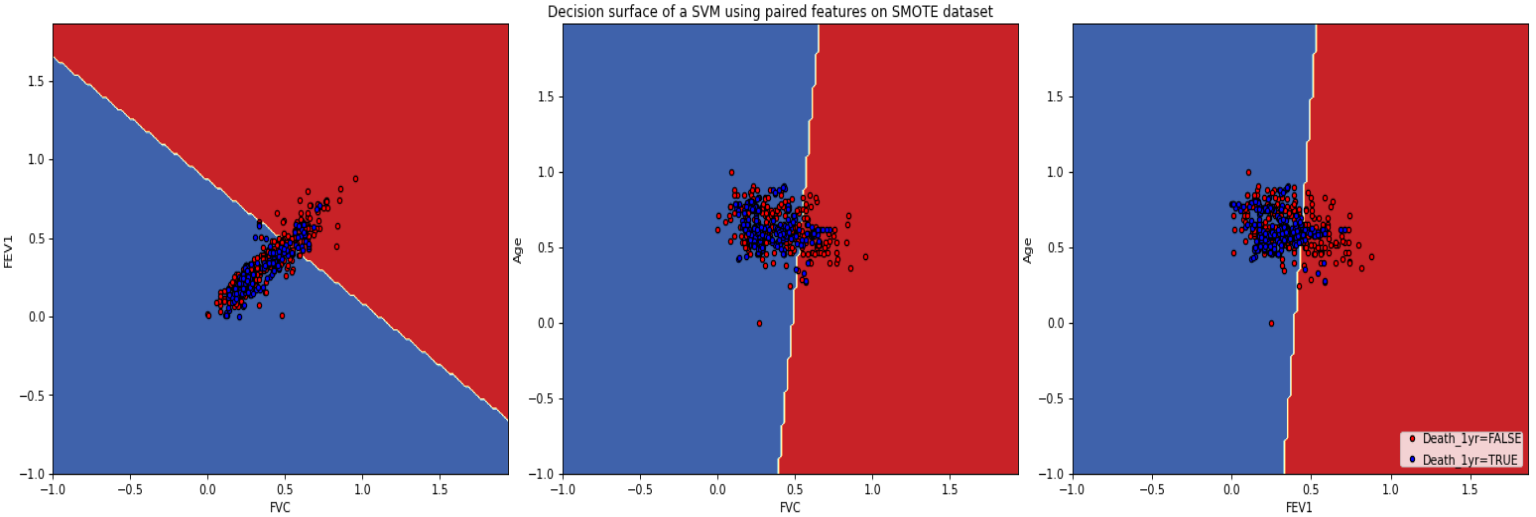


Although the recall is high, this model tends to classify as many samples as possible as positive. Indeed, the accuracy is:

$$accuracy = \frac{tp+tn}{tp+fp+fn+tn} = \frac{17+45}{17+4+71+45} = 45.25\%$$

which is far from being satisfying, whether this is the metric of major interest.

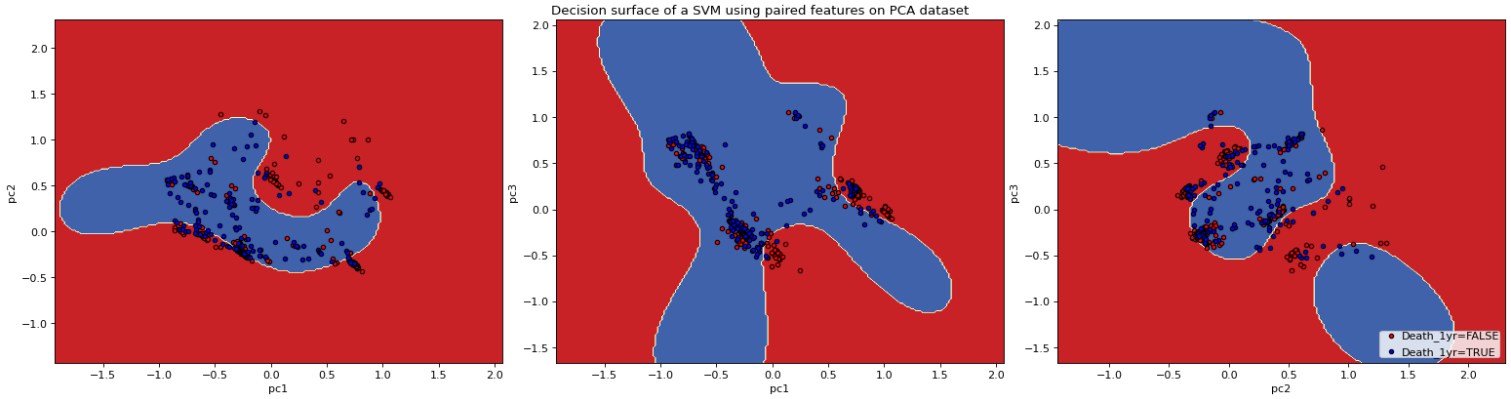
In any case, the decision boundary of this model on the continuous features *FEV1, FVC* and *Age* is the following:



Even if the decision boundary appears to be linear, it is actually a smooth and very zoomed curve, caused by the found gamma value being very small. A discussion about the change of the decision boundary over the change of the gamma and C parameters can be found here.

A clear demonstration of this fact can be seen by checking the SVM model trained on the PCA dataset. Indeed, for the model trained on the PCA dataset and selected for its recall, during the

grid search got a **gamma** parameter equals to **1** (way bigger than the 0.001 found for the SMOTE dataset) and a **C** parameter equals to **100** thus the decision boundary appears to be bubbly:



The PCA dataset reached a score of 70% of accuracy.

Finally, we provide a table containing a score of accuracy and recall for all the datasets:

ACCURACY TABLE

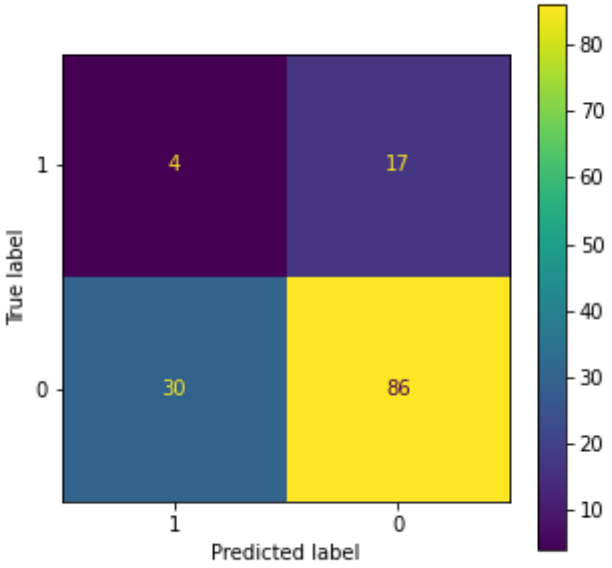
	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
SVC	0.851064	0.846715	0.729927	0.656934	0.70073

RECALL TABLE

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
SVC	0.0	0.047619	0.238095	0.809524	0.285714

About the accuracy score, we can see an indirect confirmation that the SVM model is not much sensible to outliers.

For completeness, we also check the model trained on SMOTE and selected for its accuracy, and the confusion matrix is the following:



Unlike the previous case, this model reached a high accuracy and a low recall.

Logistic Regression

The logistic regression model is used to model the probability that the response variable Y belongs to a particular category. In order to model this probability logistic regression uses the *logistic function*:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

The logistic function ensures that p(X) is always between 0 and 1 and will always produce an S-shaped curve, and so regardless of the value of X, we will obtain a sensible prediction.

Our problem consists in predicting a binary response using multiple predictors. So we use the following *logistic function*:

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

Where X_1, X_2, ..., X_p, are the features and β_1, β_2, ..., β_p are the weights to be learned.

Thus, at validation time, given a set of features X_1, X_2, ..., X_p and a set of weights β_1, β_2, ..., β_p, our logistic function will return the probability that the sample belongs to the positive class. Logistic regression in this setting can be used only for binary classification, which is our case.

We used the variable *Death_1yr=True* as positive class, and *Death_1yr=False* as negative class. Therefore, the logistic function will output the probability of not surviving 1 year after the thoracic surgery.

To estimate the β parameters we can use different approach, one of them is the **Maximum Likelihood Estimation** (MLE), which is a technique consisting in maximizing the following likelihood estimation:

$$\ell(\beta_0, \beta) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i)).$$

This function consists of the product of the probabilities that our data are labeled with 1 ($y_i = 1$) times the product of the probability that they are labeled with 0 ($y_i = 0$).

However, this formulation was not used because it was not included in the library that we used. Instead, we defined a cost function and tested different solvers to find the minimum of this function.

Given **β** = [β_1, $\beta_{2, \ldots,}$ β_p] the vector containing all the weights, $\boldsymbol{X_i} = [x_{i1}, x_{i2}, \ldots, x_{ip}]$ a sample containing *p* features, and y_i the target for the i-th sample, the cost function is defined as:

$$\sum_{i=1}^{N} ln(e^{-y_i X_i^T \beta} + 1)$$

And, eventually, two penalizations can be added, **l1** (lasso regression) and **l2** (ridge regression):

$$||\beta|| + C \sum_{i=1}^{N} ln(e^{-y_i X_i^T \beta} + 1)$$

$$\frac{1}{2}||\beta^T \beta|| + C \sum_{i=1}^{N} ln(e^{-y_i X_i^T \beta} + 1)$$

Where **C** is a parameter that weighs the impact of the penalization, and will be found through cross validation.

To minimize those cost functions, we explored different solvers, and they are:

- **Netwon method**, which approximates the cost function locally to a quadratic function, and then takes a step towards the maximum/minimum of that quadratic function
- **Broyden–Fletcher–Goldfarb–Shanno Algorithm**, which works similarly to Newton method but approximates the derivative to estimate the quadratic function

- **liblinear**, which uses a Coordinate Descent algorithm to solve the optimization problem, by successively performing approximate minimization along coordinate directions or coordinate hyperplanes
- **SAG** and **SAGA**, which approximates the cost function surface to a finite number of smooth convex functions, then uses stochastic gradient descent to find the minimum

The penalization and the parameter **C** have been object of research through cross validation, while the solver has been manually examined and chosen by heuristics.

Once again, we executed the cross validation on all the datasets and used both accuracy and recall as parameters to find the best hyperparameters.

Here we show the results concerning accuracy and recall:

Accuracy:

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
LogisticRegression	0.851064	0.846715	0.773723	0.562044	0.620438

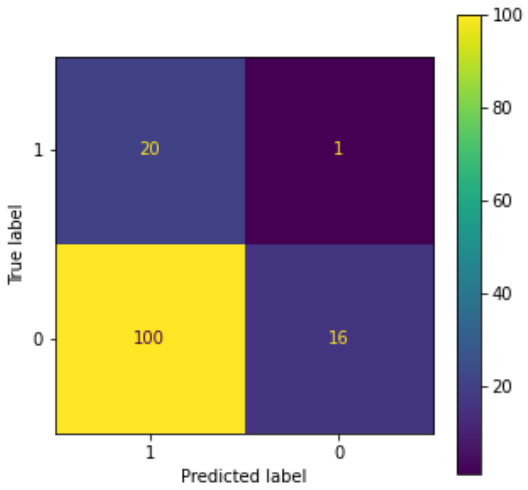
Recall:

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
LogisticRegression	0.0	0.0	0.952381	0.952381	0.714286

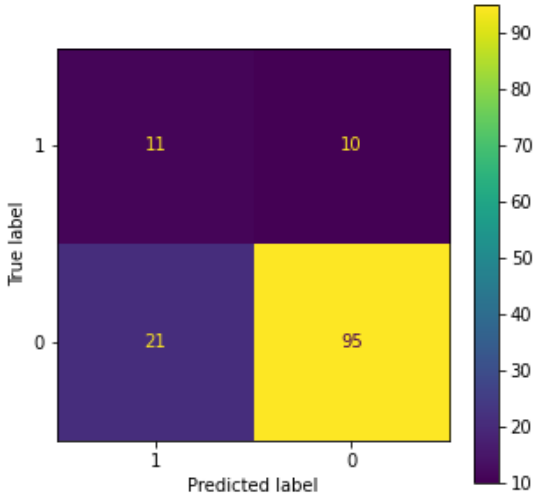
All the datasets reached an accuracy greater than 50%, even if the unbalanced ones may be misleading.

Indeed, we can see how good values for recall are reached only by *random oversampling* (**95.24%**), *SMOTE* (**95.24%**) and *PCA* (**71.43%**). This time, the oversampling technique used had no impact on the recall score.

The confusion matrix for the random oversampling and SMOTE datasets is the following (they are the same):



while, the logistic regression model selected for the accuracy, got an accuracy of 77.37% on the random oversampled dataset:



So it can be of some meaning in the cases where the overall accuracy is more important than the recall, which instead only regards the positive class. This model was obtained using the linear solver with l2 penalty and the C parameter equals to 10.

Instead, if we consider the model trained on the SMOTE dataset and selected for its recall, it was found using the linear solver with l2 penalty and the C parameter equals to 0.001.

We then provide an example of the weights found and their application in the formula.

```
  = [-0.0022, -0.0079, -0.0094, 0.0101, -0.0046, -0.0073, -0.0031, 0.0314,
-0.0086, 0.0116, -0.0072, -0.0010, -0.0025, 0.0179, -0.0010, -0.0024]
```

And let's consider the features (**x**) of the following patient randomly extracted from the test set:

Diagnosis	FVC	FEV1	Performance	Pain	Haemoptysis	Dyspnoea	Cough	Weakness	Tumor_Size	Diabetes_Mellitus	MI_6mo	PAD	Smoking	Asthma	Age
0.166667	0.403292	0.40708	0.5	0	0	0	1	0	1.0	0	0	0	1	0	0.30303

Which has all the features equal to zero, except *Diagnosis, FVC, FEV1, Performance, Cough, Tumor_Size, Smoking* and *Age*.

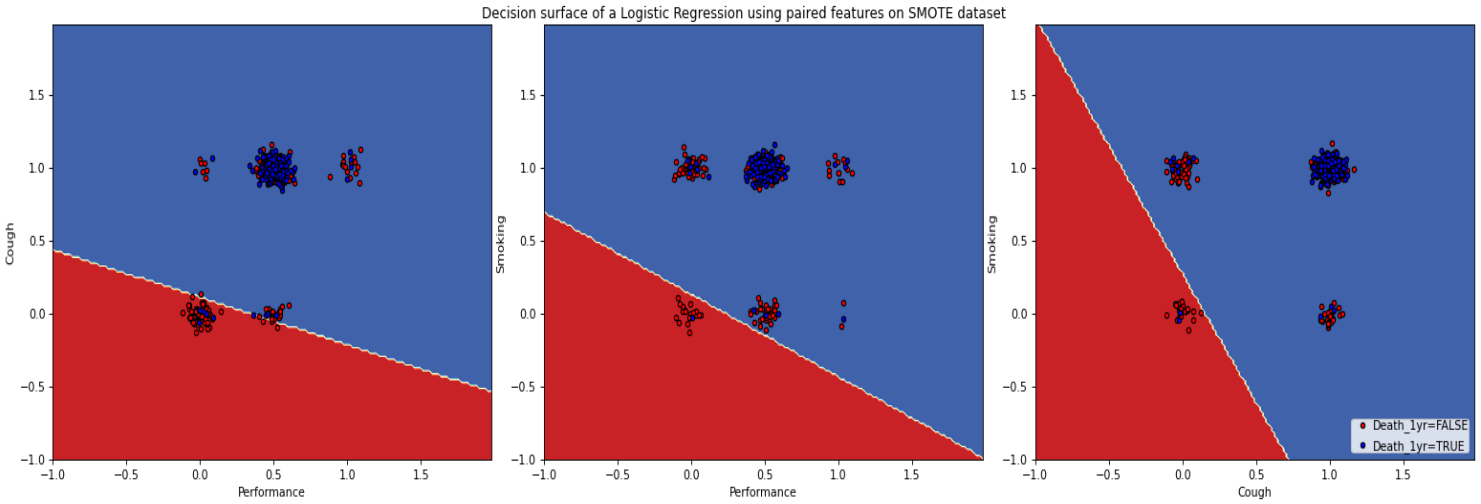
Then we calculate $\mathbf{x}^T = 0.05788$

And we apply it to the formula above:

$$\frac{e^{X^T}}{1+e^{X^T}} = \frac{e^{0.05788}}{1+e^{0.05788}} = \frac{1.05958}{2.05958} = 51.45\%$$

Which is greater than 50% and the predicted label would thus be 1. Checking the ground truth label of the same patient in the test set, we found out that he or she died within 1 year after thoracic surgery. Finally, we can assert that this sample was correctly classified by the model.

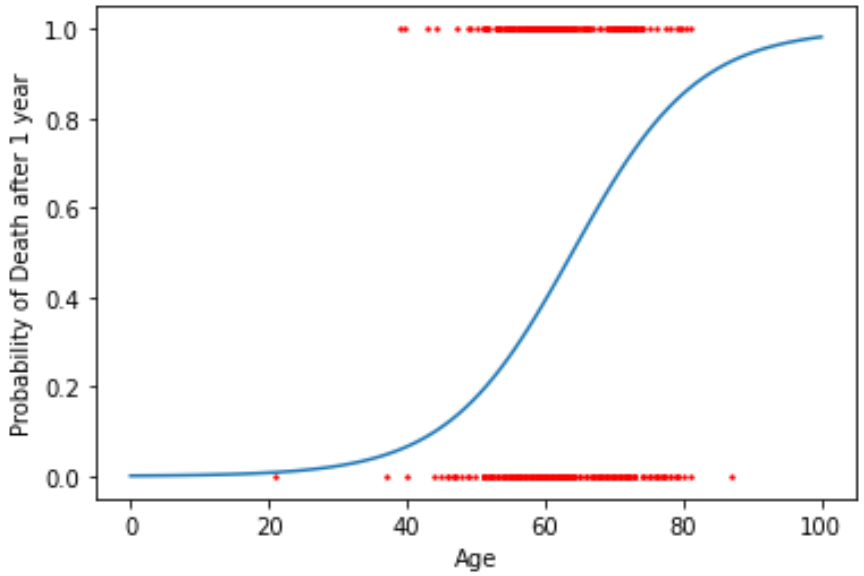
As for other classifiers, we plot the decision boundaries. We decided to show the decision boundaries for the three features that got the highest beta-value during the training, as shown in the table above. They correspond to *Performance*, *Cough* and *Smoking*:



To plot this decision boundary, we decided to add gaussian noise to the points during the plotting, in order to better appreciate the distribution and avoid overlapping (caused by all the features shown being categorical).

In the image above, we can clearly see that the decision boundary is linear.

Finally, on the same dataset, we can see the shape of the logistic function for the feature "*Age*":



Random Forest

Random forest is a type of ensemble method of machine learning. It deals with overfitting problems and increases accuracy compared to a simple decision tree model. This method grows multiple trees, on bootstrapped training samples, which are then combined to yield a single

consensus prediction. Indeed, combining a large number of trees can often result in dramatic improvements in prediction accuracy. However, one of the cons of random forest is the fact that it is less interpretable than a simple decision tree.

When building the decision trees, each time a split in a tree is considered, a random selection of *m* predictors is chosen as split candidates from the full set of *p* predictors. The split is allowed to use only one of those m predictors. So, while in Bagging we use a number *m* which is equal to the total number *p* of predictors, in the case of Random Forest we use a random selection of $m < p$ predictors. This choice of m predictors is repeated for each bootstrapped set. This means that, in the case of Random Forest, we do a bagging on the dataset, but also a feature bagging, drawing random subsets of the predictors. Good practice is to use $m \approx \sqrt{p}$ predictors. This trick **decorrelates** the multiple trees and so helps to reduce the variance when we average them.

For classification tasks (which is our goal), the output of the random forest is the class selected by most trees.

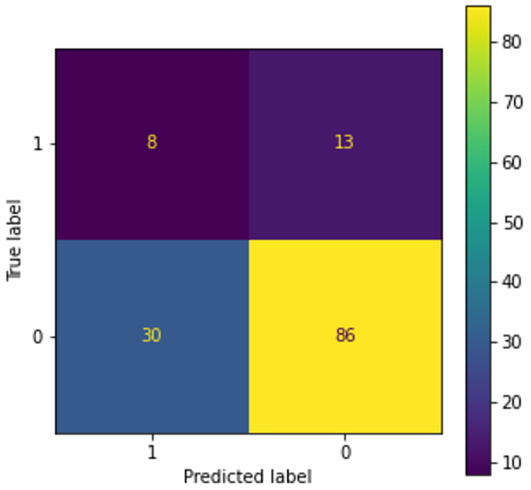
Among the most important hyperparameters of the random forest we have the number of trees that it trains, the number of features to consider when looking for the best split, and other hyperparameters that we have already seen in the case of decision tree, such as the criterion used for the splitting (Gini index or Entropy), the max death and the minimum number of samples required to split an internal node.

Also in this case, we found the best hyperparameters using grid search cross validation. From the obtained results we noticed that gini index outperforms entropy in all cases except in the case of PCA dataset.

As we can see from the table below, that shows the results obtained considering recall as metrics:

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
RandomForest	0.0	0.0	0.285714	0.238095	0.380952

the random forest generally did not achieve good results. The best recall score was obtained on PCA dataset; the associated confusion matrix is the following:



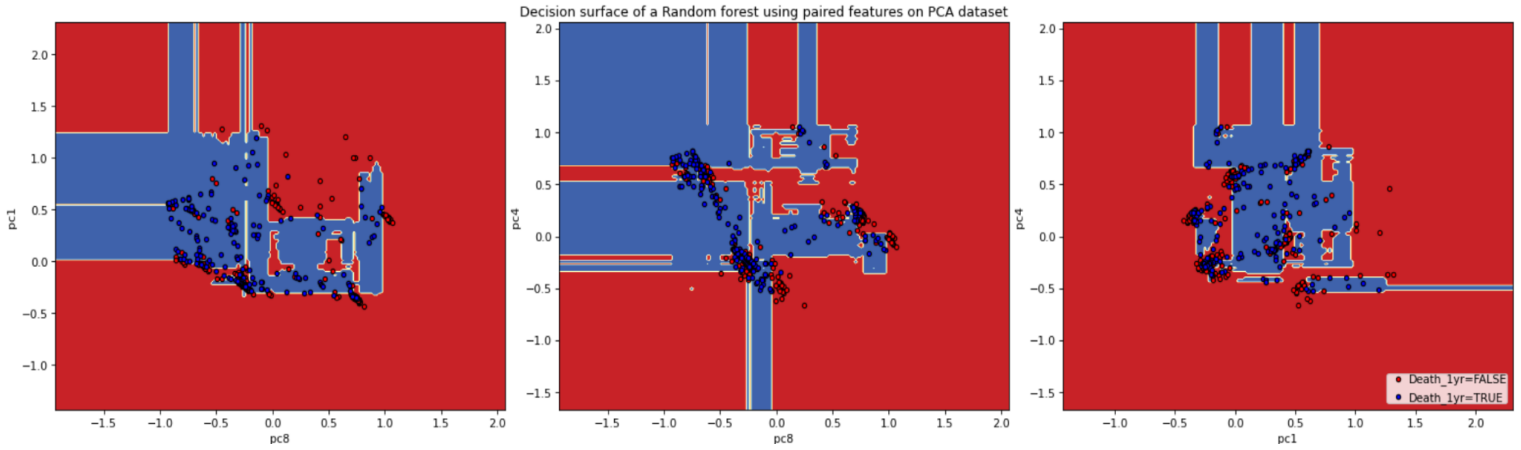
As we can see, the model made several mistakes (30 false positives and 13 false negatives) among which in particular the false negative affected the recall score.

In the case of accuracy, instead, the obtained results are shown below:

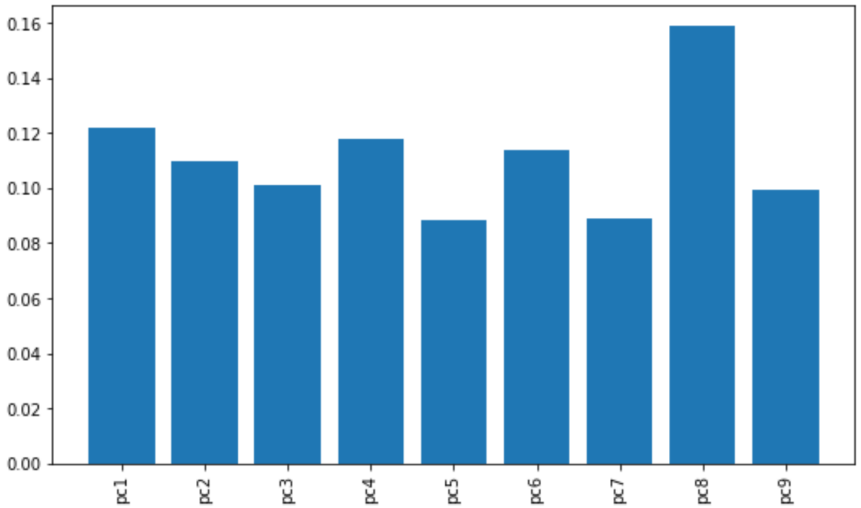
	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
RandomForest	0.851064	0.846715	0.781022	0.664234	0.693431

Also in this case the accuracy scores are generally high, due to the fact that it is a misleading metric in datasets with high unbalanced class, as in the case of our test set.

Below, is shown the decision boundary of the random forest trained on the PCA dataset:



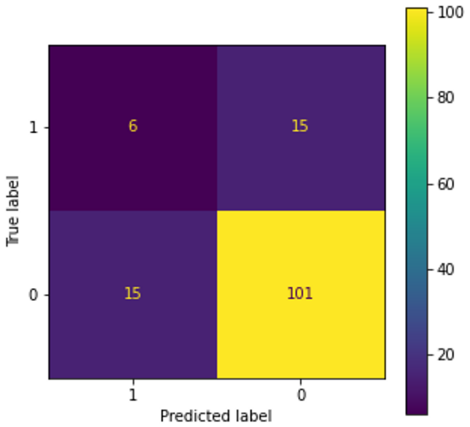
As in the case of decision trees, in order to plot the decision boundaries above, we previously extracted the feature importance for the tuned classifier, and took the pairs of the three most important features. The feature importance bar chart for the PCA dataset is visible in the following figure:



As we can see, the first three principal components in terms of importance in the construction of the model are pc8, pc1, pc4. However, those three components are not exhaustive to explain the

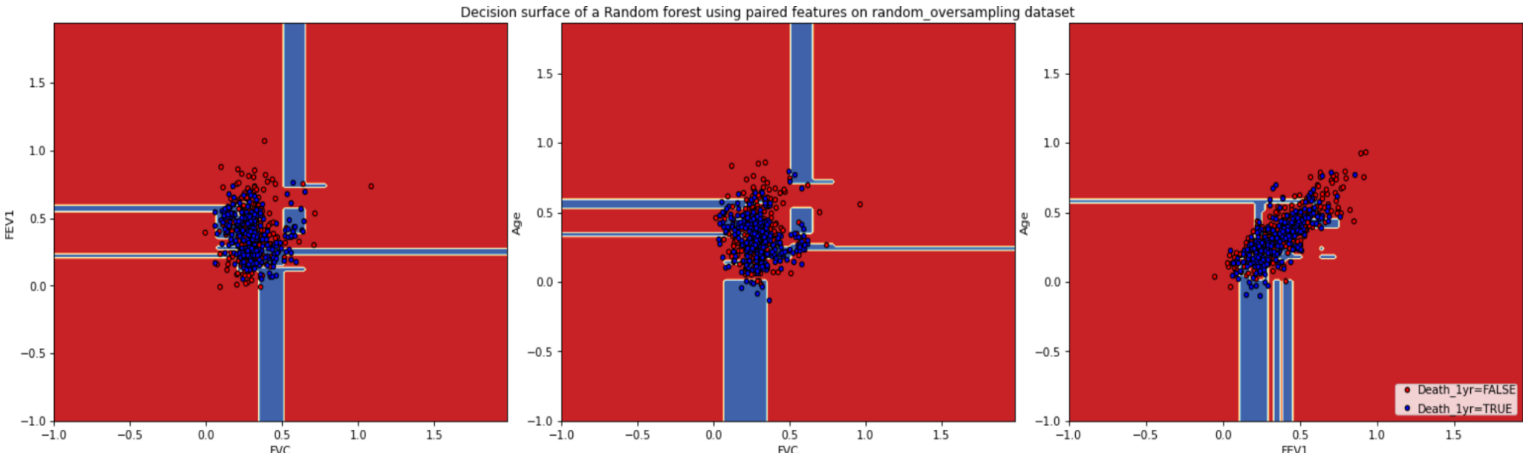
model, therefore the points erroneously classified in the previous decision boundary are determined by the remaining untracked principal components.

The second dataset in which the random forest reached the best performance in terms of recall was the random oversampling dataset. The corresponding confusion matrix is showed below:

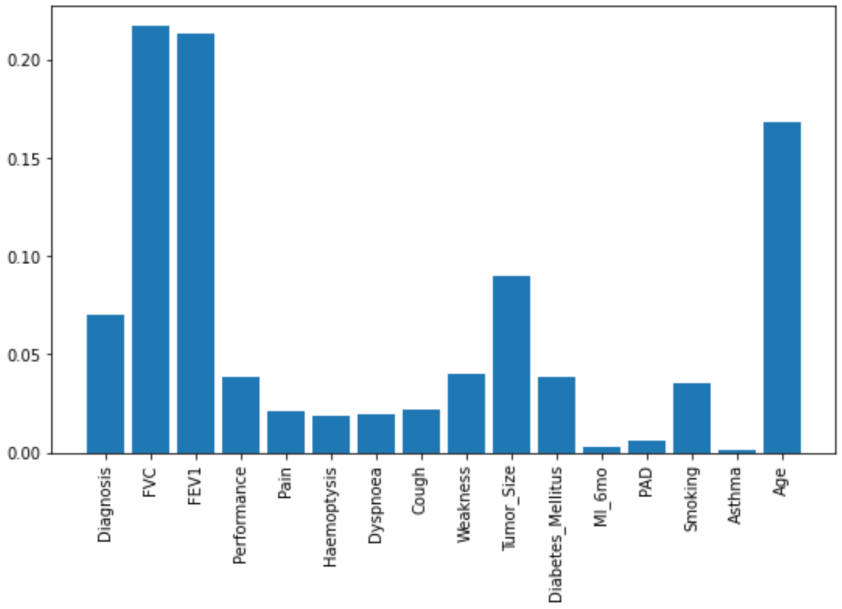


As we can see, also in this case there were several errors (15 false negative and 15 false positive) among which the false negative combined with the low number of true positives affected the recall score.

The decision boundary of the model on this second dataset is :



While the associated feature importance bar chart is:



KNN

K Nearest Neighbours is a simple supervised learning algorithm that uses all available samples in order to classify new samples based on a similarity measure (e.g., distance functions).

Given a set of *m* training samples $\{x_i, y_i\}$ and a new point x that we want to classify, we compute the distance $d_i = d(x, x_i)$ *with i = 1, …, m* between the new point x and all other points of the training set. After that, we order them based on their distance from x ($d_{i1} \geq d_{i2} \geq \ldots. \geq d_{im}$) and take the first k nearest point. The label y* assigned to the new point x is the one that is the most frequent among the k closest points $x_{i1}, \ldots, x_{ik}$.

One of the most important parameters of KNN is the **distance metric**. It defines the similarity between 2 samples and can have a huge impact on the performance of the algorithm. The most used metric is Minkowski distance:

$$L_p(x, y) = \left(\sum_{k=1}^{m} \left(x_k - y_k \right)^p \right)^{1/p}$$

p is another important parameter of the algorithm. For p=2 Minkowski distance is also known as ***Euclidean distance***, while for p=1 we talk about ***Manhattan distance***.

We took in consideration also the “weights” parameter that can assume two different values:

- *‘**uniform**’*: all points in each neighborhood are weighted equally,
- ‘***distance***’ : weight points by the inverse of their distance.

The last important parameter that we took in consideration is the number of nearest neighbours K, which is the core deciding factor. K is generally an odd number if the number of classes is 2. If K=1,

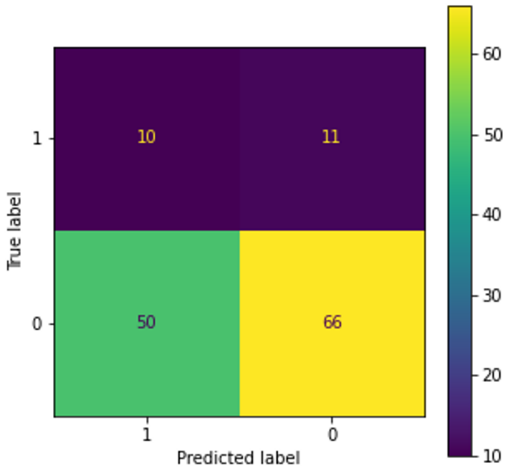
then the sample is simply assigned to the class of its nearest neighbour. However, this is usually not a good choice since it can lead to overfitting.

In order to find the best hyperparameters, we performed a grid search cross validation using different scoring metrics and datasets. As result we obtained that Manhattan distance outperforms the Euclidean one in all cases except in the case of dataset with outliers using the recall scoring, while the uniform weight generally outperforms the distance one, especially in the case of recall score.

The recall scores reached by KNN over the five different datasets are shown in the table below:

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
KNN	0.142857	0.142857	0.428571	0.47619	0.380952

The best recall results were reached on the datasets with SMOTE, with a score of **47.62%**, and with random oversampling, with a score of **42.86%,** while on all other datasets the recall score was under **40%**. As we can see the recall score on the dataset with outliers and the one without outliers (no_oversampling) are the same. This can be explained by the fact that the best hyperparameter K was set to 1 in both cases, so the outliers did not influence the decision. The confusion matrix corresponding to the best recall score reached on SMOTE dataset is shown below:

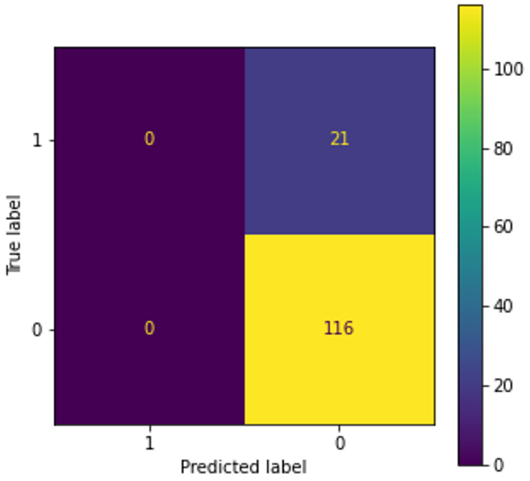


Considering the accuracy, instead, the best results are reached on dataset with outliers (**85.11%**), but in this case good results are generally reached in all datasets:

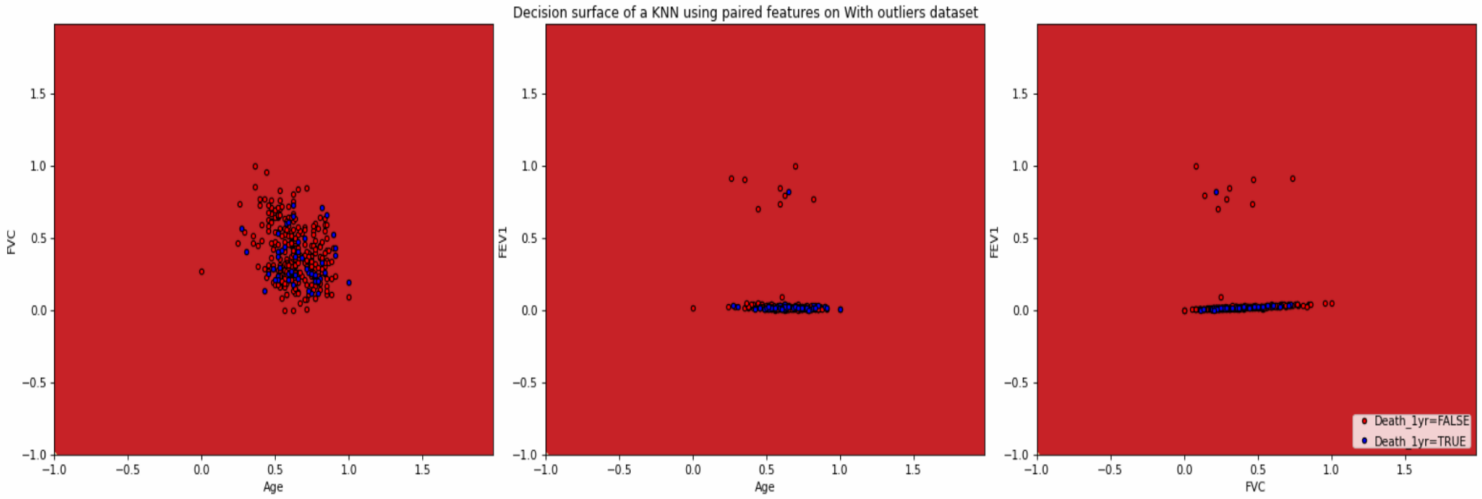
	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
KNN	0.851064	0.846715	0.781022	0.642336	0.744526

This confirms that accuracy is not a good metric in our case, due to the fact that our test set is highly unbalanced. Indeed, if we consider the case of “no_oversampling” dataset, with the same

confusion matrix (shown below), KNN reaches a very high accuracy score but, at the same time, a very poor recall.

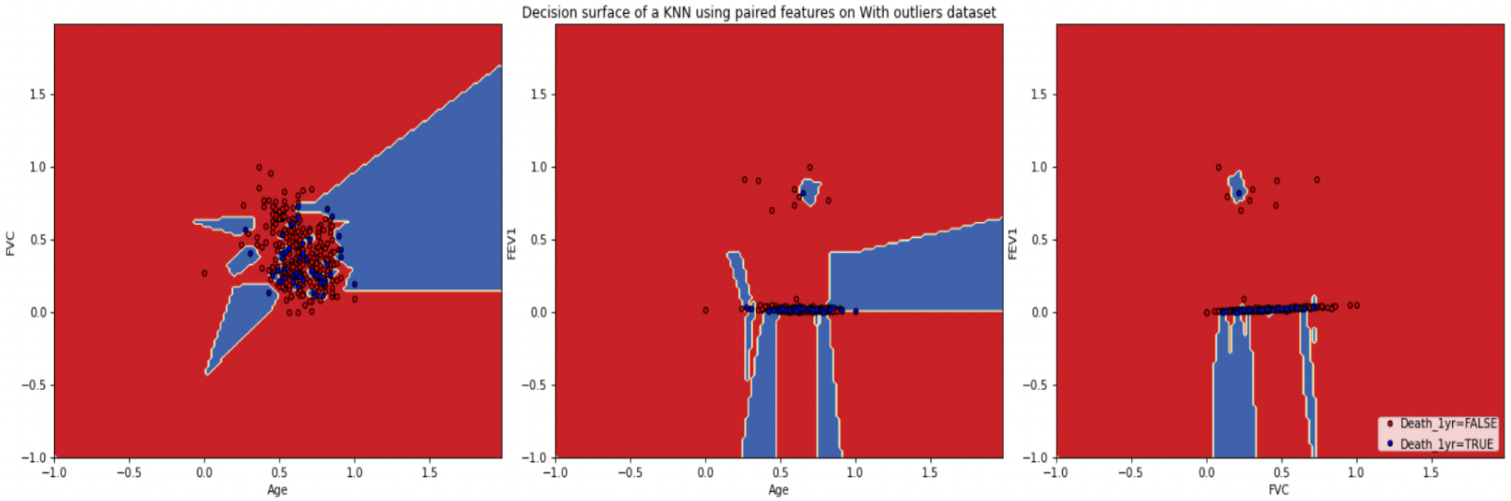


A similar result has been reached by considering the dataset retaining outliers, where the model selected for its accuracy did underfit. During the grid search, the optimal value of **k** has been found to be equal to **13**, probably because of the outliers. The result of this is that the decision boundary is not existent:



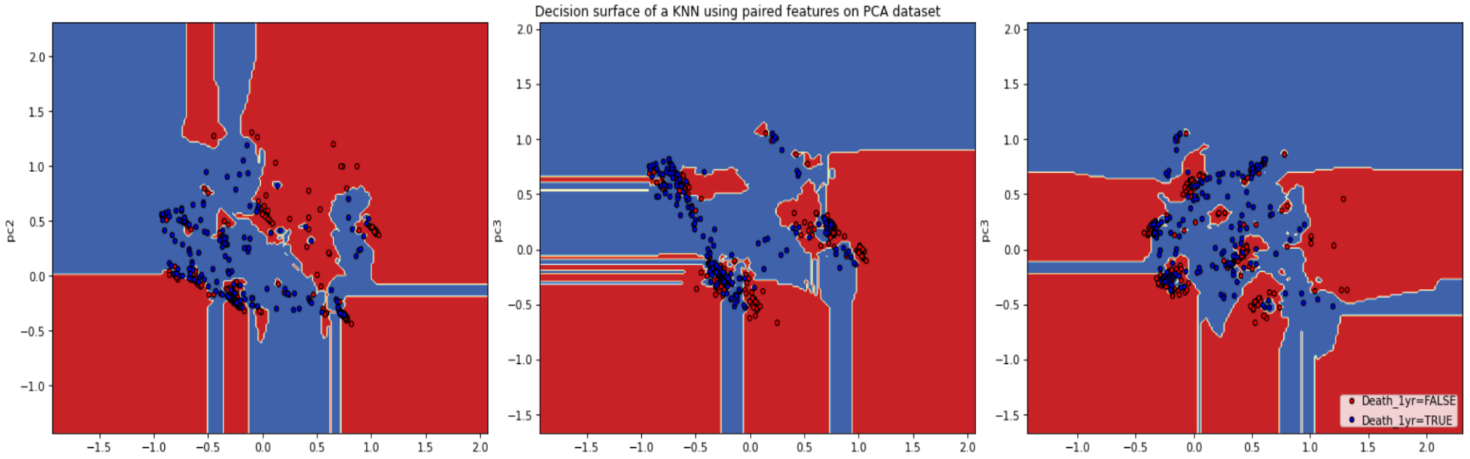
To demonstrate that this model did underfit we can consider the fact that it reached a training accuracy equal to **85.10%** and an identical test accuracy, since the split has been performed with bootstrapping.

On the other hand, when selecting the model for its recall, the model did overfit and the best value of **k** has been found to be equal to **1**. Also in this case, the outliers played a decisive role as visible in the following decision boundary:



To demonstrate that this model did overfit, we can consider that it reached a train accuracy equal to **100%**, but the test accuracy is way lower (even if still high, considering the imbalanced dataset).

Finally, in order to provide a visual interpretation, in the following images is represented the decision boundary of the KNN trained on the PCA dataset using three paired features:



The code of all classifiers can be found at:
https://github.com/GiuseppeMoscarelli/Thoracic-Surgery/blob/main/src/4_classification.ipynb .

Result and conclusions

Our aim was to explore different classifiers and consider the impact of different manipulations applied to the initial dataset. The main difficulties in our work were linked to the scarce dimension of the dataset and the imbalance between the two classes. The dataset contained 16 features, and considering that many models relied only on a handful of them, we can assert that many features were not relevant to determine the target label.

Since the sensitivity of the research, concerning the impact of the decision of a model over the eventuality of undergoing a thoracic surgery and thus over the life of the patients, we have put a

lot of emphasis upon the interpretation and, in conclusion, we have revealed that the features “Tumor_Size”, “Smoking”, “Age” and “FEV1” had the largest impact across all the classifiers.

Downstream our work, despite the results being mostly unsatisfactory, we also obtained different classifiers with a high score of accuracy and recall, that can be of some meaning for a in-field application.

To sum up the score reached by all the classifiers over all the datasets, we report the following accuracy table:

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
DecisionTree	0.851064	0.846715	0.737226	0.678832	0.635036
SVC	0.851064	0.846715	0.729927	0.656934	0.700730
LogisticRegression	0.851064	0.846715	0.773723	0.562044	0.620438
RandomForest	0.851064	0.846715	0.781022	0.664234	0.693431
KNN	0.851064	0.846715	0.781022	0.642336	0.744526

and the following recall table:

	with_outliers	no_oversampling	random_oversampling	SMOTE	PCA
DecisionTree	0.428571	0.190476	0.142857	0.666667	0.428571
SVC	0.000000	0.047619	0.238095	0.809524	0.285714
LogisticRegression	0.000000	0.000000	0.952381	0.952381	0.714286
RandomForest	0.000000	0.000000	0.285714	0.238095	0.380952
KNN	0.142857	0.142857	0.428571	0.476190	0.380952

as it is visible from the table above, the datasets retaining outliers and removing outliers, before performing oversampling, got a consistent accuracy score across all the models. This can be explained because they are both highly imbalanced datasets, and they introduced a bias during the training. For this reason, during grid search some hyperparameters were selected to blindly reach a high accuracy, and this often has been equivalent to a model that classified all the samples as negative.

As an example, the dataset retaining outliers contained **21** positive samples and **120** negative samples. Thus, all the classifiers classified all the samples as negative, reporting an accuracy of

$$acc_{with\ outliers} = \frac{tp+tn}{tp+fp+fn+tn} = \frac{120}{120+21} = 85.10\%$$

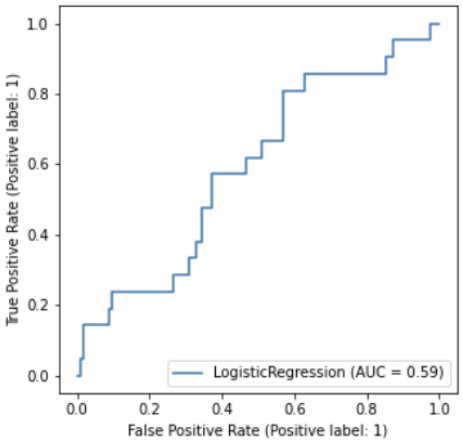
the same approach can be used to explain the other not-oversampled column:

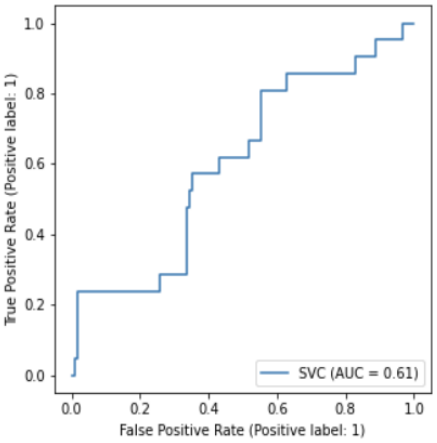
$$acc_{without\ outliers} = \frac{tp+tn}{tp+fp+fn+tn} = \frac{116}{116+21} = 84.67\%$$

for the same reason, many not oversampled models got a recall equal to **0%**.

Conversely, not all the oversampling techniques performed equally well. If we consider the logistic regression algorithm we can clearly see that just a random oversampling soared the recall from **0%** to **95.24%**, thus repeated samples do not impact the performance of this classifier with respect to synthesized samples. On the other hand, other models were impacted by the repeated samples, and the decision tree has even had a drop in the recall score after random oversampling. Generally, the SMOTE dataset performed better across all the classifiers, while the PCA+SMOTE dataset did not perform as well. We calculated a drop in recall of 30% between the SMOTE and PCA dataset, way higher than the ~10% of explained variance that we removed during the PCA, and that we expected to be similar in this context.

In general, the logistic regression and SVM performed best. To better visualize the performance of these two models and make a comparison, we report the ROC curves of both on the SMOTE dataset:





The ROC curve represents the trade-off between the True Positive Rate and False Positive Rate across the different values of discrimination threshold. Since the Area Under the Curve (AUC) is slightly bigger in the SVC model, we can consider this model better.

A slightly better result has been reached considering the Random Oversampled dataset, where the AUC reached by the logistic regression was equal to 0.72: