POLITECNICO DI TORINO

Mathematics in Machine Learning

Myocardial Infarction Complication dataset analysis

Candidates:

Beatrice Macchia 292178, Riccardo Bosio 291299

Professors:

Francesco Vaccarino, Mauro Gasparini

Settembre 2021

Contents

List of Figures

Chapter 1

Introduction

Myocardial Infarction is one of the most challenging problems of modern medicine. Acute myocardial infarction is associated with high mortality in the first year after it. The incidence of MI remains high in all countries. This is especially true for the urban population of highly developed countries, which is exposed to chronic stress factors, irregular and not always balanced nutrition. In the United States, for example, more than a million people suffer from MI every year, and 200-300 thousand of them die from acute MI before arriving at the hospital. The course of the disease in patients with MI is different. MI can occur without complications or with complications that do not worsen the long-term prognosis. At the same time, about half of patients in the acute and subacute periods have complications that lead to worsening of the disease and even death. Even an experienced specialist can not always foresee the development of these complications. In this regard, predicting complications of myocardial infarction in order to timely carry out the necessary preventive measures is an important task.

Code is available at https://github.com/riccardobosio/MML.

1.1 Dataset

In this tesina we will take into consideration the dataset "Myocardial infarction complications" (1). This dataset is composed of 1700 instances and 124 attributes. We can solve different classification problems: we have 12 features (columns 113-124) we can predict, each one associated with a particular complication. Moreover we can consider 4 possible time moments for complication prediction:

1. The time of admission to hospital: all input columns (2-112) except 93, 94, 95, 100, 101, 102, 103, 104, 105 can be used for prediction.

2. The end of the first day (24 hours after admission to the hospital): all input columns (2-112) except 94, 95, 101, 102, 104, 105 can be used for prediction.

3. The end of the second day (48 hours after admission to the hospital): all input columns (2-112) except 95, 102, 105 can be used for prediction.

4. The end of the third day (72 hours after admission to the hospital): all input columns (2-112) can be used for prediction.

We chose to try to predict the complication 'ZSN', which indicates Chronic heart failure, at the end of the third day. It means that all input columns are available for the prediction.

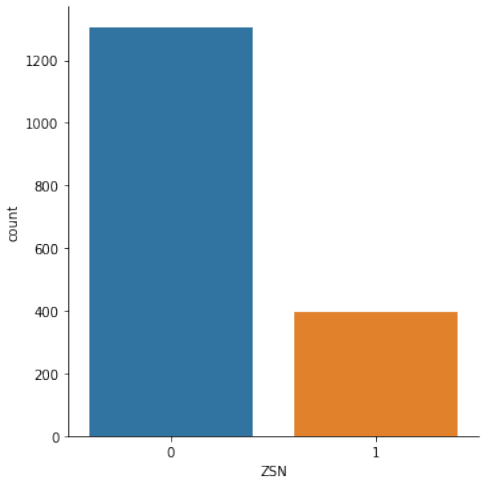
Chapter 2

Data exploration

2.1 Balance

As we can see in figure 2.1, the dataset is unbalanced since the class 0 outnumbers the class 1. This can cause many problems since machine learning classifiers are sensitive to the proportions of classes and fail with unbalanced training datasets. The algorithms tend to favor the majority obtaining misleading accuracies: we are interested in the correct classification of the minority class, but we find high accuracies which are the product of the correct classification of the majority class. To solve this problem we will rebalance data (Section 3.5).

Figure 2.1. 'ZSN' distribution: number of 0: 1306; number of 1: 394.



2.2 Nan

We have to deal with the presence of 15974 nan values: all features except *id* and *sex* have at least one nan. Nan can be handled dropping all the records that contain them, replacing them with a default value or replacing them with a custom value (i.e. mean, median, most present value). We exclude dropping all the records that contain them since each row has at least one nan, but we could consider dropping columns with too many nan. That is what we did, deleting the features $'IBS\_NASL', 'D\_AD\_KBRIG', 'S\_AD\_KBRIG'$, $'KFK\_BLOOD', 'NA\_KB', 'NOT\_NA\_KB', 'LID\_KB'$. Then we fill the remaining nan values with 0.

2.3 Features selection

If $\mathcal{X}$ is a subset of a vector space, each $x \in \mathcal{X}$ is sometimes referred to as a feature vector (4). The way we encode real world objects as an instance space $\mathcal{X}$ is by itself prior knowledge about the problem. Even when we already have an instance space $\mathcal{X}$ which is represented as a subset of a vector space, we might still want to change it into a different representation and apply a hypothesis class on top of it. That is, we may define a hypothesis class on $\mathcal{X}$ by composing some class H on top of a feature function which maps $\mathcal{X}$ into some other vector space $\mathcal{X}'$. We assume $\mathcal{X} = R^d$, that is, each instance is represented as a vector of d features. We want to learn a predictor that only relies on $k \ll d$ features. Predictors that use only a small subset of features require a smaller memory footprint and can be applied faster. Furthermore, in applications such as medical diagnostics, obtaining each possible feature (e.g., test result) can be costly; therefore, a predictor that uses only a small number of features is desirable even at the cost of a small degradation in performance, relative to a predictor that uses more features. Finally, constraining the hypothesis class to use a small subset of features can reduce its estimation error and thus prevent overfitting. Ideally, we could have tried all subsets of k out of d features and choose the subset which leads to the best performing predictor. However, such an exhaustive search is usually computationally intractable. We have used the following two methods: Sequential Backward Selection and Chi-Square Test. By reducing the number of features to 40, they returned two subsets that have 19 features in common (47,5%).

2.3.1 Sequential Backward Selection

The Sequential Backward Selection algorithm takes the whole feature set as input and returns a subset of features. The number of selected features k has to be specified apriori. At each step the algorithm removes from the set of features the feature that is associated with the best classifier performance if it is removed. We repeat this procedure until we have k features left.

It means that we start with I as initial set of features, then we go over all $i \in I$ and apply the learning algorithm on the set of features $I - \{i\}$. We choose to remove the feature i for which the predictor obtained from $I - \{i\}$ performs better.

We chose K-Nearest-Neighbors as learning algorithm with k=4.

2.3.2 Chi-square Test

The chi-square test is a statistical test of independence to determine the dependency of two variables. The chi-square test can be usefull in feature selection by testing the relationship between features.

To explain it, we define the Chi-Square distribution:

$$\chi^2 = \sum Z_i^2$$

where Z_1, Z_2 are standard normal variables.

Moreover the Chi-square distribution depends on the degrees of freedom, which refers to the maximum number of independent values and so it is equal to N-1 (N is the sample size). When the degrees of freedom increase, the Chi-square distribution approximates to normal distribution.

In order to perform a Chi-square test, we need to define the Null Hypothesis. In our case we want to test the independence of two variables, so we set the independence itself as the Null Hypothesis. Therefore, we calculate the Expected value, based on the null hypothesis. It means that if A,B are two independent events, then

$$P(A \cap B) = P(A) * P(B)$$

. Finally, having the observed and expected values, we calculate the Chi-square value:

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

with c=degrees of freedom, O=observed values, E=expected values.

When two features are independent, the observed count is closed to the expected one, and so we will have a smaller Chi-square value. In feature selection, we want to select the features that are highly dependent on the target: the higher the Chi-square value is, the more dependent the feature is on the target, and can be selected for model training.

2.4 Feature correlation

Pearson's coefficient is a measure of linear correlation between two sets of data. It is the ration between the covariance of two variables and the product of their standard deviations. It ranges from -1 to 1, where if the Pearson's coefficient is either 1 or -1, there is a linear mapping from v to y with zero empirical risk. Imagine that we are studying the correlation between two features x and y: if Pearson's coefficient equals zero it means that the optimal linear function from x to y is the all-zeros function, which means that x alone is useless for predicting y. However, this does not mean that x is a bad feature, as it might be the case that together with other features x can perfectly predict y. Looking at the correlation matrix 2.2, we can see that the features $'NA\_BLOOD'$ and $'K\_BLOOD'$ have correlation 0.9. So we can drop one feature of each couple since they almost bring the same information.

Figure 2.2. Feature correlation (after feature selection).

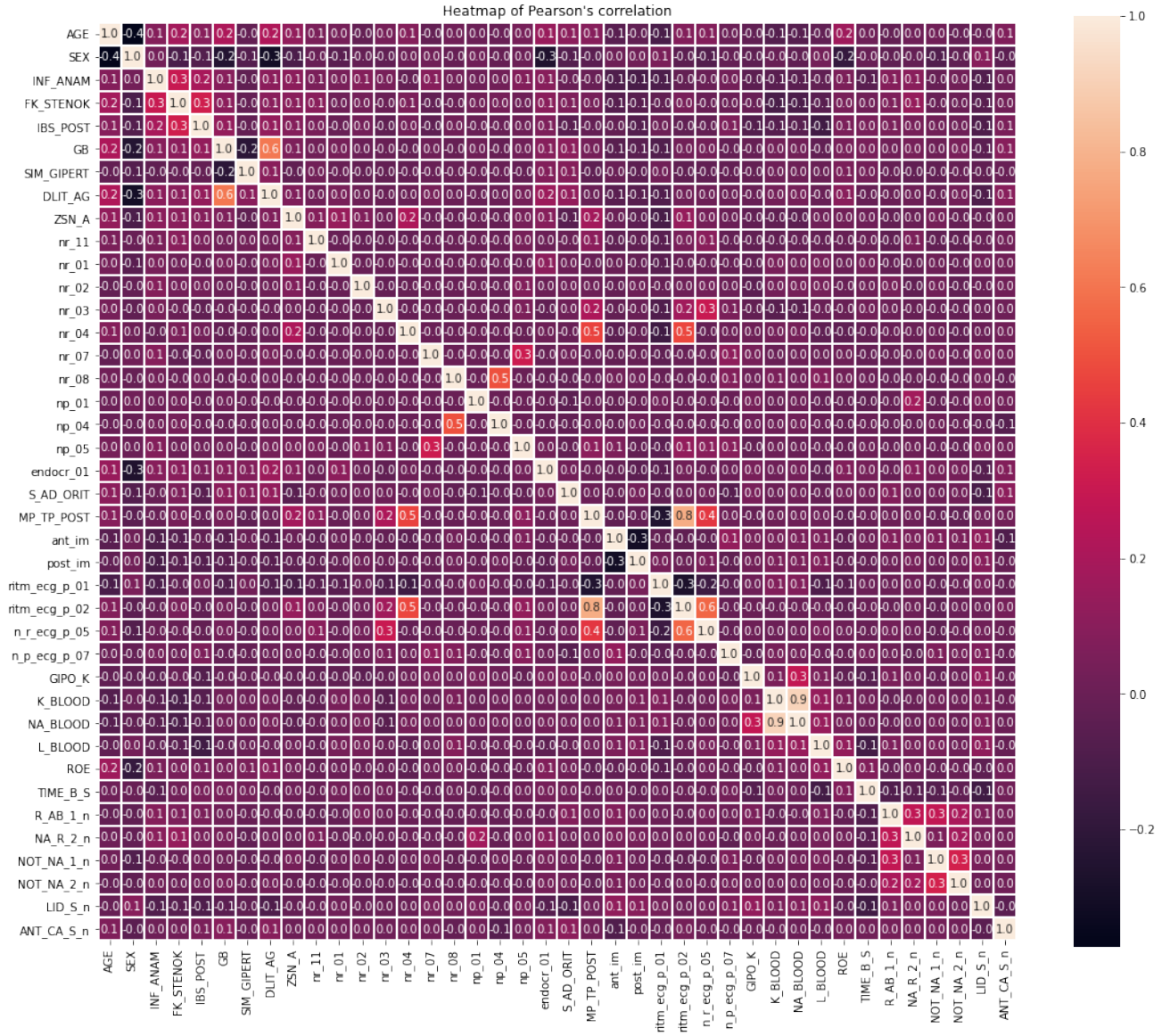
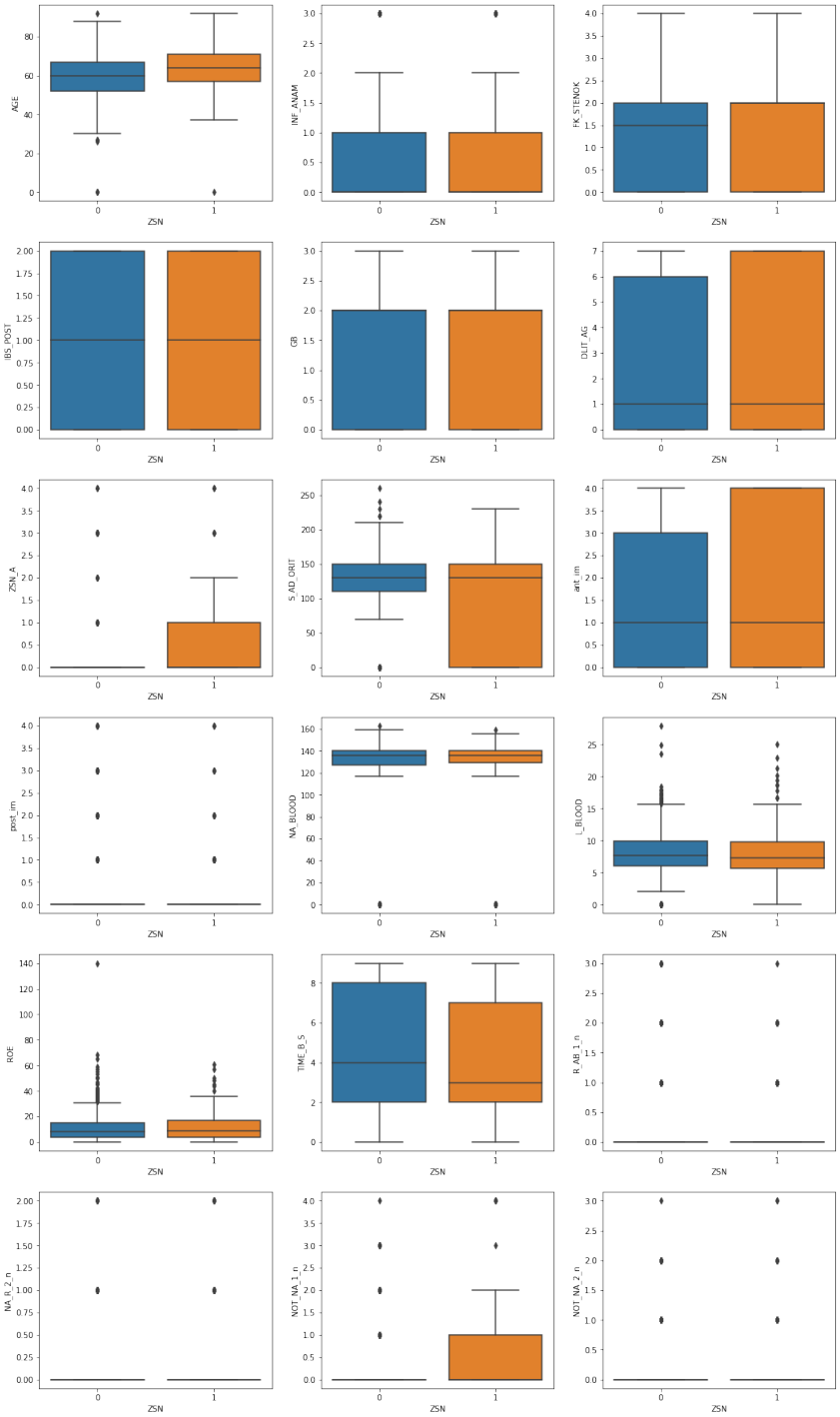


Figure 2.3. Vertical boxplots of non-binary features.



Chapter 3

Data preparation

3.1 Standardization

We perform a standardization of the numerical features, so they will have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 0$.. Standard scores (also known as z scores) of the samples are calculated as

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

Standardizing the features so that they are centered around 0 with a standard deviation of 1 is not only important if we are comparing measurements that have different units, but it is also a general requirement for the optimal performance of many machine learning algorithms. Some examples of algorithms where feature scaling matters :

1. k-nearest neighbors with an Euclidean distance measure if we want all features to contribute equally;
2. SVM if we are using gradient descent/ascent-based optimization, otherwise some weights will update much faster than others;
3. principal component analysis since we want to find directions of maximizing the variance and we want to have features on the same scale so we can emphasize variables on "larger measurement scales" more.

3.2 Outliers

Outliers are extreme observations that deviate from other values. They not always indicate an experimental error: sometimes they represent a variability in a measurement or a novelty. There are two types of outliers: univariate and multivariate. Univariate outliers concern the distribution of values of a single feature, while multivariate outliers can be found in a n-dimensional space. In machine learning the quality of data is as important as the quality of a classification (sometimes even more). For this reason we need to manage them in the proper way.

3.2.1 Domains and Distributions

A boxplot (2) can be viewed as a graphical representation of the five-number summary of the data consisting of the minimum, maximum, and the first, second, and third quartiles. In figure 2.3 we can see vertical boxplots of our non-binary attributes after feature selection. The box is drawn from the first quartile (Q_1) to the third quartile (Q_3). The horizontal line inside the box signifies the location of the median. So-called "whiskers" extend to either side of the box. The size of the box is called the *interquartilerange*: $IQR = Q_3 - Q_1$. The upper whisker extends to $Q_3 + 1.5IQR$. Any data point outside the whiskers is indicated by a small dot, indicating a suspicious or deviant point (outlier).

After feature selection, we analyzed the distributions of the remaining features. They are all binary, except for the 18 discrete attributes, whose boxplot is represented in figure 2.3.

3.2.2 Z-score

We can use the Z-score to describe a data point in terms of its relationship to the mean and standard deviation of a group of points. Mapping the data into a distribution with mean 0 and standard deviation 1, we have centered and rescaled data and we can compare them. A standard threshold for Z-score is 3 or -3, so that if a value is outside the interval

$$(\bar{x} - 3\sigma, \bar{x} + 3\sigma),$$

then this value is an outlier. Applying z-score we drop 393 rows obtaining a training set of 831 rows.

3.2.3 Isolation forest

Isolation Forest identifies anomalies by creating decision trees over random attributes. Few instances are in smaller partitions and distinguishable attribute values are more likely to be separated in early partitioning. This random partitioning produces noticeable shorter paths for anomalies and the anomaly will be detected in smaller number of partitions than a normal point (3.1). We stop when a node has a single element. The number of splits reveals the level at which the isolation happened, and it will be used to generate the anomaly score:

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

where $c(n) = 2H(n-1) - \frac{(2(n-1))}{n}$, n = number of leaves. $c(n)$ is the maximum path length from root to leaf, $E(h(x))$ is the average path lengths, from the Isolation Forest and H is the harmonic number. When $E(h(x)) \to 0$, $s \to 1$;

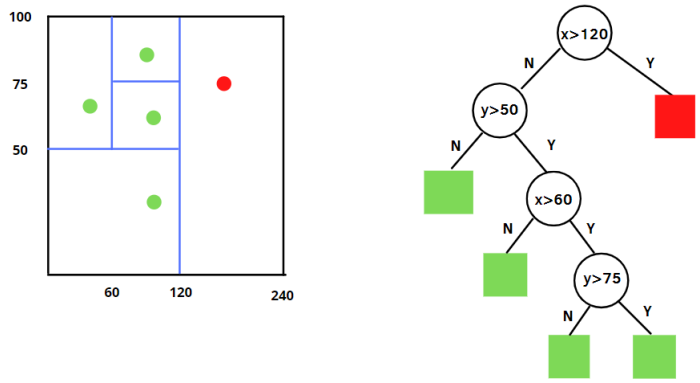
When $E(h(x)) \to n - 1$, $s \to 0$;

When $E(h(x)) \to c(n)$, $s \to 0.5$.

If a point score is closer to 0 it is a normal observation, while if it is closer to 1 it is considered as an anomalous point. In the same way, outliers are easier to isolate in a decision tree, since they require fewer random splits than a normal point.

We can pass the hyperparameter *contamination* to the scikit-learn algorithm: it is the expected proportion of outliers in the dataset. Applying the Isolation Forest algorithm we drop 111 rows so the new training set is composed of 1113 observations.

Figure 3.1. Isolation Forest example.



3.2.4 Local Outlier Factor

The anomaly score of each sample is called Local Outlier Factor. It measures the local deviation of density of a given sample with respect to its neighbors. It is local in that the anomaly score depends on how isolated the object is with respect to the surrounding neighborhood. More precisely, locality is given by k nearest neighbors, whose distance is used to estimate the local density. By comparing the local density of a sample to the local densities of its neighbors, one can identify samples that have a substantially lower density than their neighbors. These are considered outliers.

Using a right number k isn't straight forward. While a small k has a more local focus, i.e. looks only at nearby points, it is more erroneous when having much noise in the data. A large k, however, can miss local outliers.

We denote with k-distance(A) the distance of the object A to the k-th nearest neighbor and with $N_k(A)$ the set of k nearest neighbors, which includes all objects at this distance even though there might be more than k objects. If k was 3, the k-distance would be the distance of a point to the third closest point.

The Reachability Distance is calculated by applying the maximum between the k-distance of b, the second point, and the distance between the two points.

$$reach - dist(a,b) = max\{k - distance(b), dist(a,b)\}.$$

If a point is within the k neighbors of point b, the $reach - dist(a,b)$ will be the k-distance of b; otherwise it will be the real distance of a and b.

The Local Reachability Density is the inverse of the average of the reachability distance of a to all its k nearest neighbors. This means that the longer the distance to the next

neighbors, the less dense the point is.

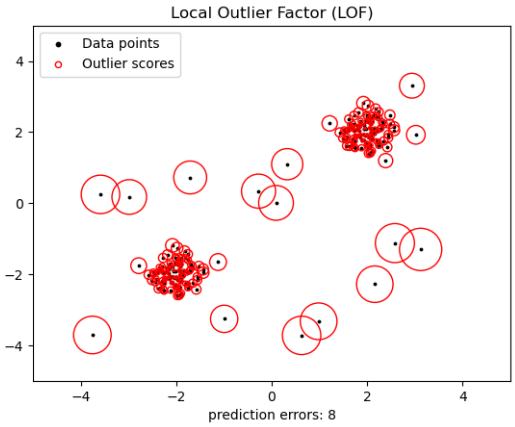
$$lrd_k(A) = \frac{1}{\frac{\sum_{B \in N_k(A)} reach\text{-}distance_k(A,B}{|N_k(A)|}}.$$

The Local Reachability Density tells us how far we have to travel from our point to reach the next points: the lower it is, the longer we have to travel. The LOF for a given point a is calculated by computing the average Local Reachability Density of its k neighbors and then dividing it by its Local Reachability Density.

If the density of a point is much smaller than the densities of its neighbors, the LOF is much greater than 1 and so the point is detected as an outlier. A LOF value of 1 or less, instead, is a good indicator of an inlier.

Using the Local Outlier Factor we drop 41 rows so the new training set is composed of 1183 observations.

Figure 3.2. LOF example.



3.2.5 One-Class SVM for novelty detection

Schoelkopf introduced the idea of using one-class SVM to learn a decision boundary that is able to reach the maximum separation between the points and the origin. It uses a transformation function ϕ defined by the kernel to project data in an higher dimensional space and soft margins and separates data from the origin. Only a small portion of points is allowed to lay on the other side of the decision boundary and those are outliers (3.3). We must find a subset S such that the probability that a test point lies outside S is bounded by a priori specified value. We define

$$f(x) = \begin{cases} +1, & \text{if } x \in S \\ -1, & \text{if } x \in \bar{S} \end{cases}$$

$f(x)$ is defined as the $signg(x)$, where $g(x) = w^T\phi(x) - \rho$, where ρ is the bias term and $\|w\|$ is the margin. The objective of one-class SVMs is:

$$min_{w,\epsilon,\rho}\frac{1}{2}\|w\|^2 + \frac{\sum_{i=1}^{n}\epsilon_i}{vn} - \rho$$

s.t. $w^T\phi(x_i) >= \rho - \epsilon_i$, $\epsilon_i >= 0 \forall i = 1, ..., N$, where ϵ_i is the slack variable for point i that allows it to lie on the other side of the decision boundary, n is the size of the training dataset and v is the regularization parameter. v sets an upper bound on the fraction of outliers and it is a lower bound on the number of training examples used as Support Vector.

By using Lagrange techniques and using a kernel function for the dot-product calculation becomes:

$$f(x) = sign(w^T\phi(x) - \rho) = sign(\sum_{i=1}^{n}\alpha_i K(x, x_i) - \rho).$$

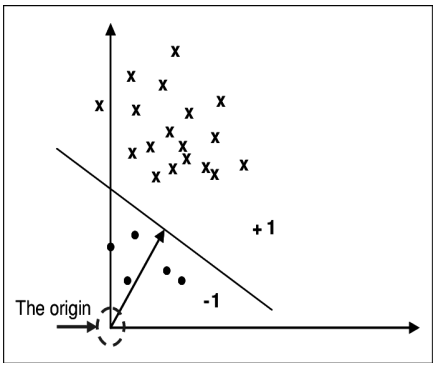
Applying the one-class SVM algorithm we would drop 785 rows so the new training set would be composed of only 439 rows.

Analyzing the results of these 4 outliers detection methods we observe that:

- One Class SVM reduces too much the training dataset size;
- Local Outlier Factor and Isolation Forest have 1080 predictions in common (out of 1224), corresponding to the 88%;
- z-score and Isolation Forest identify 827 inliers in common.

So we decide to use z-score.

Figure 3.3. One-class SVM example.



3.3 Dimensionality Reduction

3.3.1 Principal Component Analysis

Dimensionality reduction is the process of taking data in a high dimensional space and mapping it into a new space whose dimensionality is much smaller. There are several reasons to reduce the dimensionality of the data. First, high dimensional data impose computational challenges. Moreover, in some situations high dimensionality might lead to poor generalization abilities of the learning algorithm. Finally, dimensionality reduction can be used for interpretability of the data, for finding meaningful structure of the data, and for illustration purposes.

In Principal Component Analysis (PCA) (4), the reduction is performed by applying a linear transformation to the original data. If the original data is in R^d and we want to embed it into R^n $(n < d)$ then we would like to find a matrix $W \in R^{n,d}$ that induces the mapping $x \mapsto Wx$. A natural criterion for choosing W is in a way that will enable a reasonable recovery of the original x. In general, exact recovery of x from Wx is impossible.

In PCA the method finds the linear transformations for which the differences between the recovered vectors and the original vectors are minimal in the least squared sense.

Defining the recovered version of x as $\tilde{x} = Uy$ where $y = Wx$, we aim at solving the problem

$$argmin_{W \in R^{n,d}, U \in R^{d,n}} \sum_{i=1}^{m} \|x_i - UWx_i\|_2^2.$$

This 1 is a pseudocode of PCA:

Data: A matrix of m examples $X \in R^{m,d}$, number of components n
if $(m > d)$ **then**
  $A = X^T X$;
  Let $u_1, ..., u_n$ be the eigenvectors of A with largest eigenvalues
else
  $B = XX^T$;
  Let $v_1, ..., v_n$ be the eigenvectors of B with largest eigenvalues
  for $i = 1, ..., n$ set $u_i = \frac{1}{\|X^T v_i\|} X^T v_i$
end
Result: $u_1, ..., u_n$

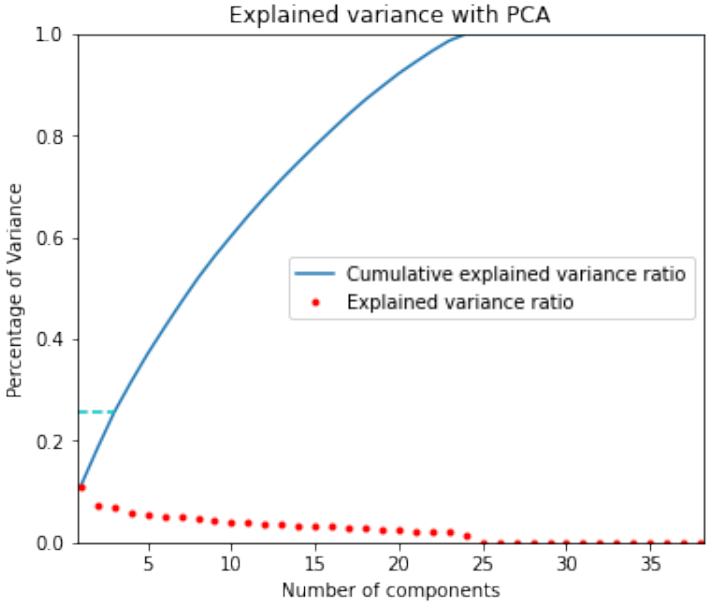
Algorithm 1: PCA

Studying cumulative explained variance (3.4), we decide to set the number of components to 25.

Biplot

This graph, where points are the projected observations and vectors are the projected variables, has these properties:

Figure 3.4. Cumulative explained variance ratio and explained variance ratio.



- the cosine of the angle between pairs of vectors shows the correlation between the corresponding variables. Vectors that point in similar directions are highly correlated, while perpendicular ones are uncorrelated. A small angle implies positive correlation, while a large one means negative correlation;
- the cosine of the angle between a vector and an axis indicates the importance of the contribution of the corresponding variable to the principal component;
- Close points represent observations with similar values.

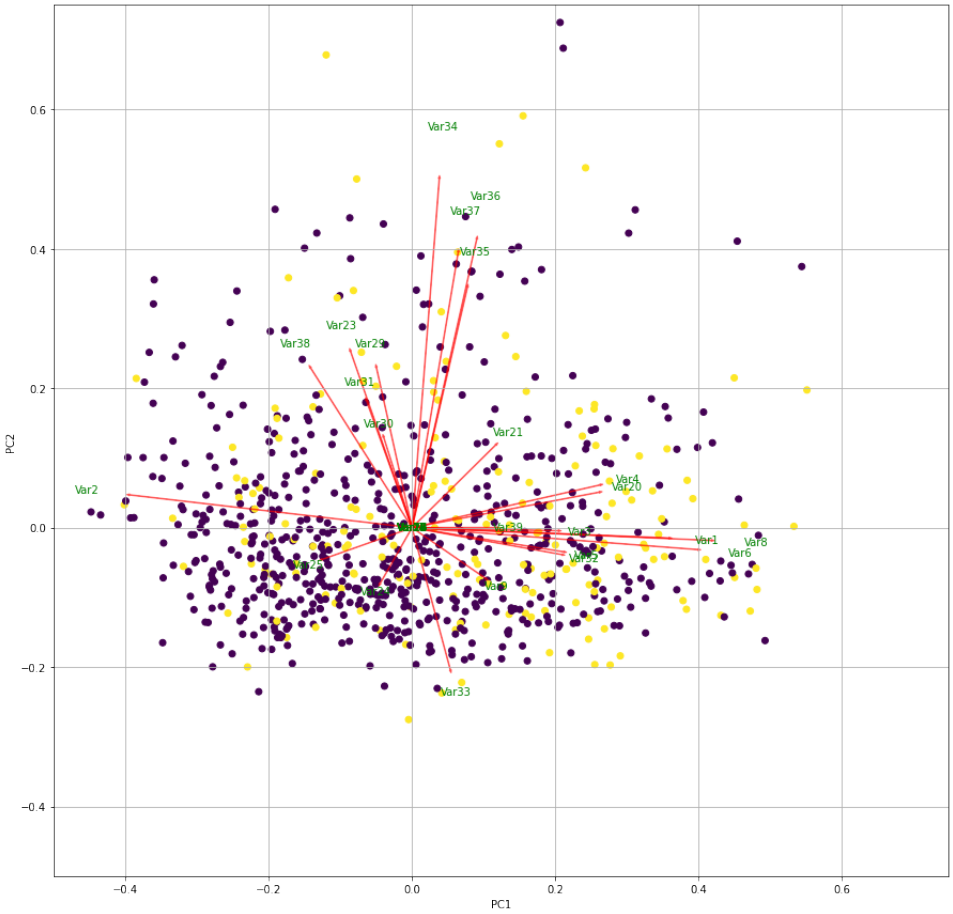
3.3.2 Linear Discriminant Analysis

Linear Discriminant Analysis performs supervised dimensionality reduction by projecting the input data to a linear subspace consisting of the directions which maximize the separation between classes. The dimension of the output is necessarily less than the number of classes and so it is in general a rather strong dimensionality reduction. In our case we are forced to set the number of components to 1 (since we have two classes). So it doesn't make sense in our specific case.

3.4 Rebalancing

We have seen the dataset is unbalanced towards the class 0. We rebalance data in order to avoid overfitting and wrong predictions, which are often taken considering the majority

Figure 3.5. Biplot for the first 2 most important attributes, after using PCA

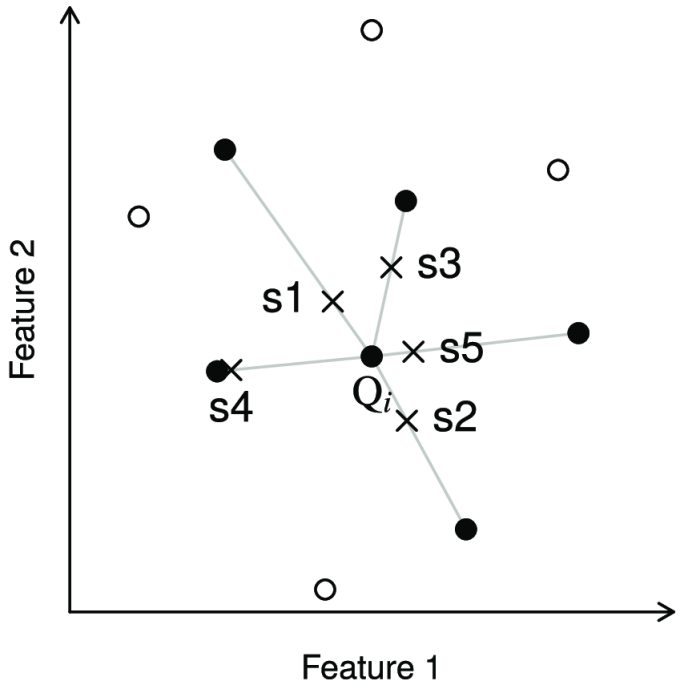


class.

3.4.1 Syntetic Minority Oversampling TEchnique

SMOTE (3) is an oversampling approach in which the minority class is oversampled by creating "synthetic" examples rather than by oversampling with replacement. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors (3.6). Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen. Synthetic samples are generated in the following way: take the difference between the feature vector (sample) under consideration and its nearest neighbor. Multiply this difference by a random number between 0 and 1, and add it to the feature vector under consideration. This causes the selection of a random point along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general.

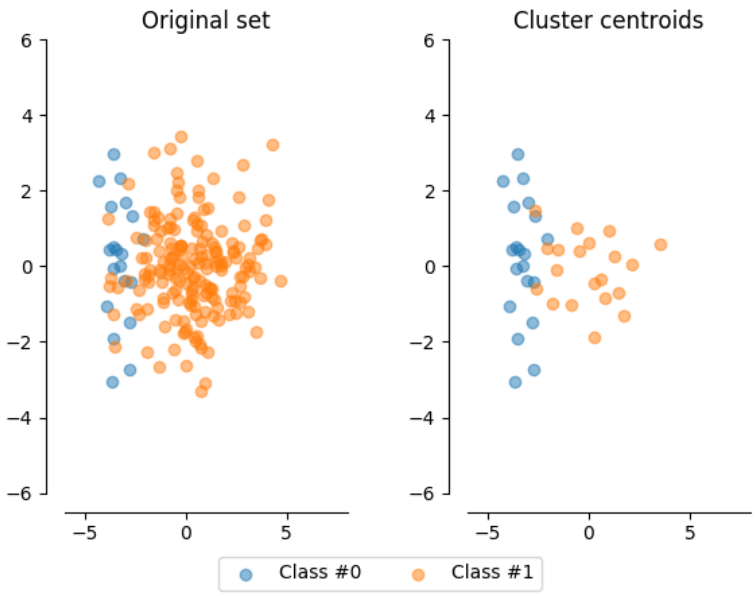
Figure 3.6. SMOTE with k = 5 and oversampling rate N = 5.



3.4.2 Cluster Centroids

Cluster Centroids is a method that under samples the majority class by replacing a cluster of majority samples by the cluster centroid of a K-means algorithm. This algorithm keeps N majority samples by fitting the K-means algorithm with N cluster to the majority class and using the coordinates of the N cluster centroids as the new majority samples (3.7).

Figure 3.7. Cluster centroids undersampling.



Chapter 4

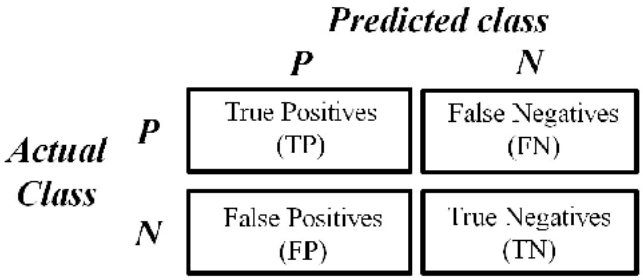
Methodology

4.1 Metrics

A confusion matrix (2) is a matrix where the (j, k)-th element counts the number of times that, for the training or test data, the actual (observed) class is j whereas the predicted class is k. We can divide this confusion matrix into four groups (4.1):

- true positive (TP): the diagonal elements, that is the numbers of correct classifications for each class.
- true negative (TN): the sum of all matrix elements that do not belong to the row or the column of this particular class.
- false positive (FP): the sum of the column elements (corresponding to a class) without the diagonal element.
- false negative (FN): the sum of the row elements (corresponding to a class) without the diagonal element.

Figure 4.1. Confusion matrix.



Given a confusion matrix we can define some metrics: functions that measure how well a method performs.

The accuracy score is the ratio between the number of correctly classified samples and the total number of samples.

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

The recall define what proportion of actual positives is identified correctly.

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

The precision measures what proportion of positive identifications is actually correct.

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

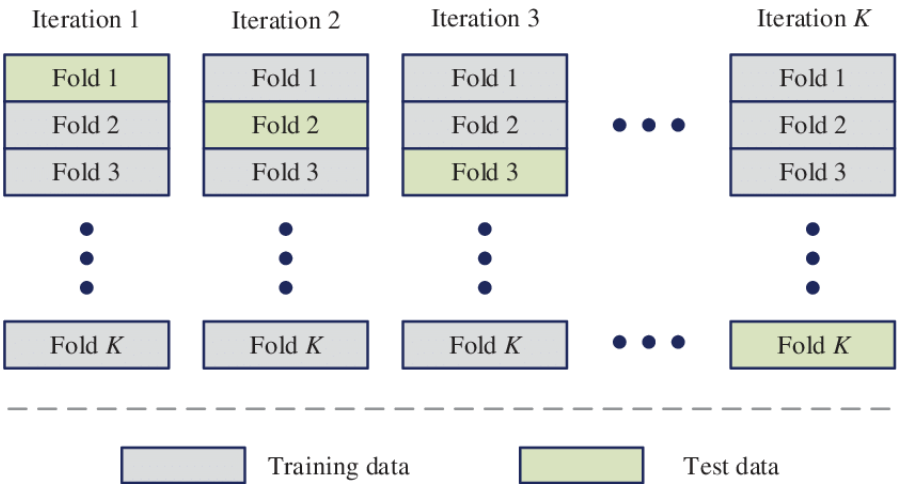
F1 takes into account both recall and precision.

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

4.2 Cross Validation

The k-Fold Cross Validation (4) is a validation technique designed to give an accurate estimate of the true error without wasting too much data. The original training set is partitioned into k subsets (folds) of size m/k, where m is the dataset size (for simplicity, assume that m/k is an integer). For each fold the algorithm is trained on the union of the other folds (4.2) and then the error of its output is estimated using the fold. The average of all these errors is the estimate of the true error. Once the best parameter is chosen, the algorithm is retrained using this parameter on the entire training set. Since we want training and test to have similar distributions, we use stratify to maintain the proportions of the data, considering the binary classification.

Figure 4.2. K-fold Cross Validation.



Chapter 5

Model selection

5.1 Support Vector Machines

Support Vector Machines (SVM) are supervised learning models whose aim is to find the hyperplane that best separates data. Since in general, the larger the margin, the lower the generalization error of the classifier, the best hyperplane has the biggest margins from the closest data points of each class, that are called support vectors.

It often happens that data are not linearly separable solving the hard margin problem. One of the alternative is the soft margin problem: we introduce slacks variable and allow for some mistakes. The hyperparameter C is the cost of misclassification: when C is large, the algorithm tries to maximize the number of points correctly classified, while a smaller C will encourage a larger margin at the cost of training accuracy.

The hyperplane is the result of this optimization problem:

$$min_w \frac{1}{2} w^T w + C \sum_i max(0, 1 - y_i w^T x_i)$$

SVM can be used with Kernels. Kernels are symmetric functions that correspond to a scalar product in an higher dimensional features space. This solution is much more efficient than computing the mapping directly: the hyperplane is computed in the new space where data could become linearly separable.

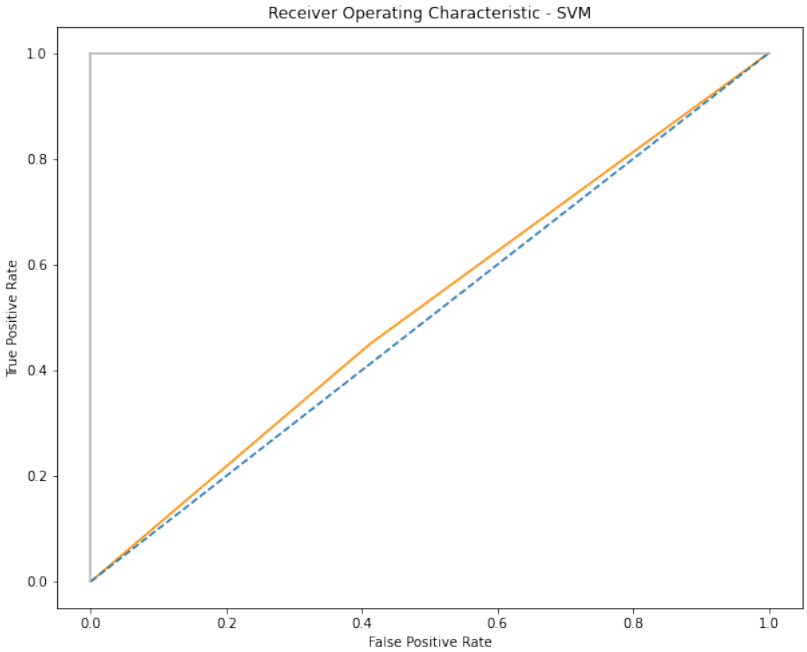
We trained the SVM and got the following results:

	precision	**recall**	**f1-score**	**support**
0	0.79	0.45	0.57	238
1	0.23	0.59	0.33	68
accuracy			0.48	306
macro avg	0.51	0.52	0.45	306
weighted avg	0.67	0.48	0.52	306

Table 5.1. SVM metrics.

We also report ROC curve (5.1) and Confusion matrix (5.2).

Figure 5.1. SVM ROC curve.



5.2 Logistic Regression

Logistic Regression is statistical learning technique used for binary classification. Given an input observation x, represented by a feature vector $[x_1, x_2, ..., x_n]$ where n is the number of features, and a target variable which can be 0 and 1, we want to know the probability $\mathcal{P}(y = 1|x)$ that this observation is a member of class 1. Logistic Regression learns a vector of weights, one for each feature, and a bias term. Each weight w_i is a real number that represents how important x_i is for the classification: it can be positive (feature associated with the class) or negative (feature not associated with the class). The bias term b (intercept) is another real number that is added to the weighted inputs.

$$z = (\sum_{i=1}^{n} w_i x_i) + b = wx + b.$$

To create a probability z is passed through the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-z}}$. $\mathcal{P}(y = 1)$ and $\mathcal{P}(y = 0)$ must sum to one:

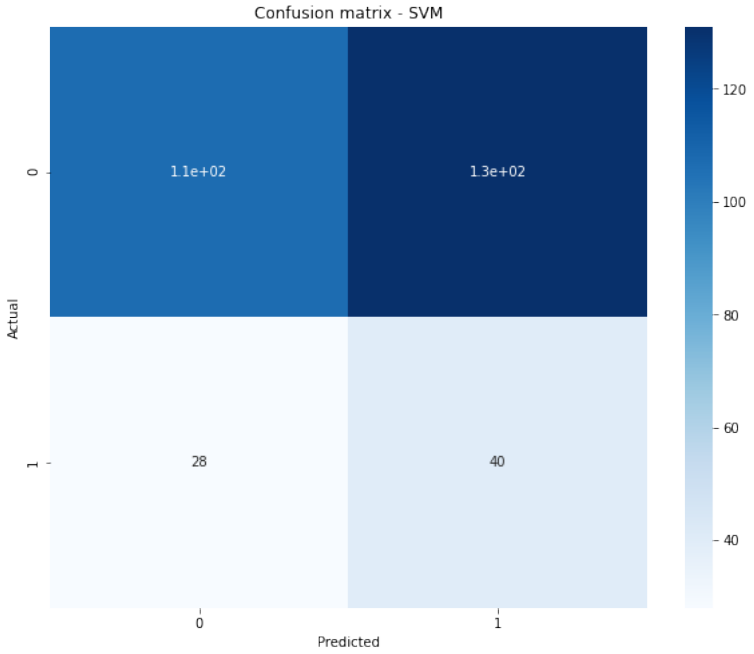
$$\mathcal{P}(y = 1) = \sigma(wx + b) = \frac{1}{1 + e^{-(wx+b)}}$$

$$\mathcal{P}(y = 0) = 1 - \mathcal{P}(y = 1)$$

The prediction $\hat{y}$ is obtained as follows:

$$\hat{y} = \begin{cases} 1, & \text{if } \mathcal{P}(y = 1|x) > 0.5 \\ 0, & \text{if } \mathcal{P}(y = 1|x) <= 0.5 \end{cases}$$

Figure 5.2. SVM confusion matrix.



We trained the Logistic Regression and got the following results on validation set:

	precision	**recall**	**f1-score**	**support**
0	0.86	0.48	0.61	238
1	0.28	0.72	0.41	68
accuracy			0.53	306
macro avg	0.57	0.60	0.51	306
weighted avg	0.73	0.53	0.57	306

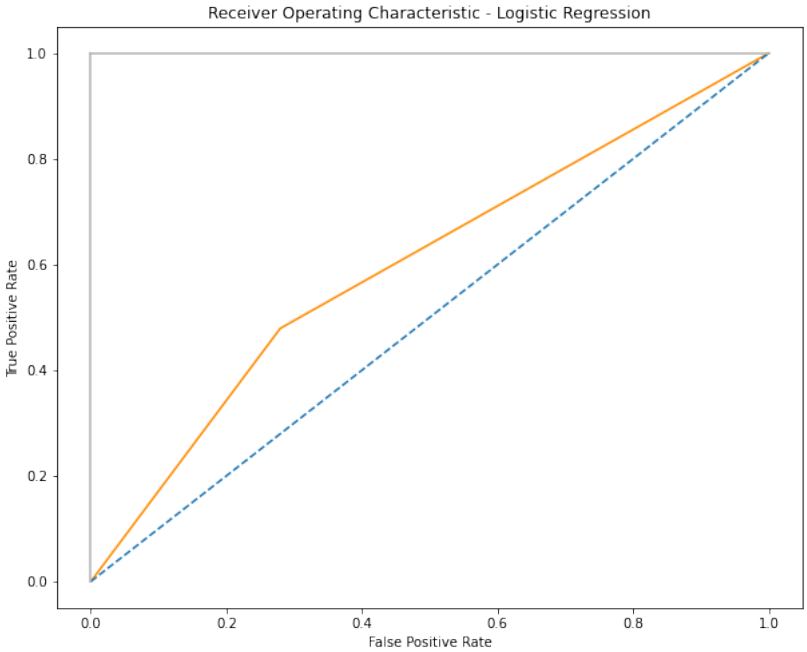
Table 5.2. Logistic Regression metrics.

We also report ROC curve (5.3) and Confusion matrix (5.4).

5.3 Decision Tree

A decision tree (4) is a predictor, that predicts the label associated with an instance x by traveling from a root node of a tree to a leaf. A leaf contains a specific label. Practical decision tree learning algorithms are based on heuristics such as a greedy approach, where the tree is constructed gradually, and locally optimal decisions are made at the construction of each node. Such algorithms cannot guarantee to return the globally optimal decision tree but tend to work reasonably well in practice. A general framework for

Figure 5.3. ROC curve of Logistic Regression.



growing a decision tree is as follows. We start with a tree with a single leaf (the root) and assign this leaf a label according to a majority vote among all labels over the training set. We now perform a series of iterations. On each iteration, we examine the effect of splitting a single leaf. We define some "gain" measure that quantifies the improvement due to this split. The most common ones are:

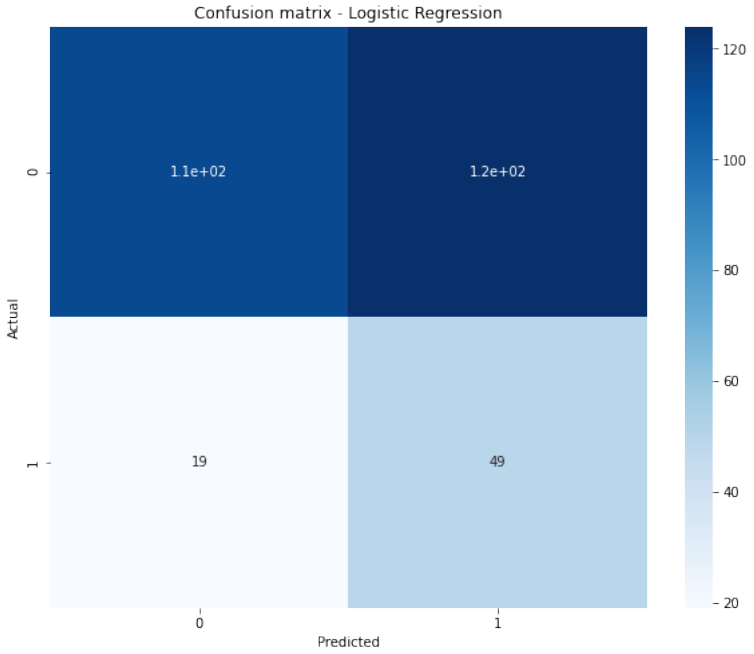
- Gini Index: $\sum_{k=1}^{P} p_{mk}(1 - p_{mk})$ which represents a measure of purity of the node;
- Cross Entropy: $-\sum_{k=1}^{K} p_{mk} log(p_{mk})$.

Then, among all possible splits, we either choose the one that maximizes the gain and perform it, or choose not to split the leaf at all. The process continues until a stopping criterion is reached (for instance, we may continue until no region contains more than a give number of observations). This algorithm may produce good predictions on the training set but worse on a test set (overfitting), leading to a poor model. A possible solution is to use a smaller tree with fewer splits to lower variance and improve generalization, achievable by stopping the growth of the tree by avoiding splits that would lower the entropy no more than a threshold. Tree-based methods are simple and highly interpretable but they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy; this is why we introduce methods such as Random Forest.

We trained the Decision Tree and got the following results on validation set:

We also report ROC curve (5.5) and Confusion matrix (5.6).

Figure 5.4. Logistic Regression confusion matrix.



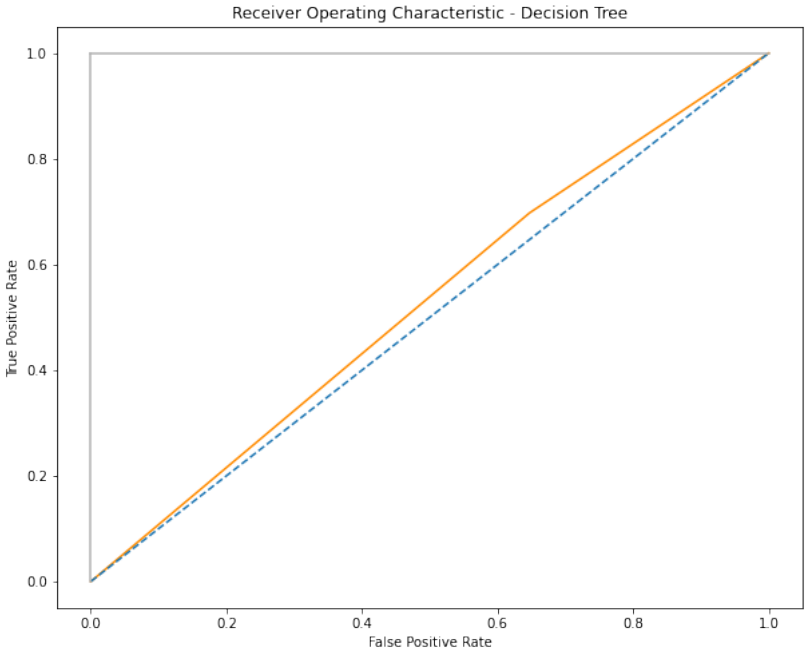
	precision	recall	f1-score	support
0	0.79	0.70	0.74	238
1	0.25	0.35	0.29	68
accuracy			0.62	306
macro avg	0.52	0.53	0.52	306
weighted avg	0.67	0.62	0.64	306

Table 5.3. Decision Tree metrics.

5.4 Random Forest

Another way to reduce the danger of overfitting is by constructing an ensemble of trees. A random forest is a classifier consisting of a collection of decision trees, where each tree is constructed by applying an algorithm A on the training set S and an additional random vector θ, where is sampled i.i.d. from some distribution. The prediction of the random forest is obtained by a majority vote over the predictions of the individual trees. (5.7) We generate as follows: first, we take a random subsample from S with replacements; namely, we sample a new training set S' of size m' using the uniform distribution over S. Second, we construct a sequence $I_1, I_2, ...$, where each I_t is a subset of $[d]$ of size k, which is generated by sampling uniformly at random elements from $[d]$. All these random variables form the vector θ. Then, the algorithm A grows a decision tree based on the sample S', where at each splitting stage of the algorithm, the algorithm is restricted to

Figure 5.5. Decision Tree ROC curve.



choosing a feature that maximizes the gain from the set I_t. Intuitively, if k is small, this restriction may prevent overfitting.

We trained the Random Forest and got the following results on validation set:

	precision	**recall**	**f1-score**	**support**
0	0.76	0.11	0.19	238
1	0.22	0.88	0.35	68
accuracy			0.28	306
macro avg	0.49	0.50	0.27	306
weighted avg	0.64	0.28	0.23	306

Table 5.4. Random Forest metrics.

We also report ROC curve (5.8) and Confusion matrix (5.9).

5.5 K-Nearest Neighbors

The K-NN algorithm stores all the training data with the labels and for each input computes the distances from all other points. Then the k nearest neighbors are considered and the label is assigned by majority voting. Most common distance measure is Minkowski: $(\sum_{i=1}^{D} |a_i - b_i|^p)^{\frac{1}{p}}$. It is easy to implement but requires an high computational cost in

Figure 5.6. Decision Tree confusion matrix.

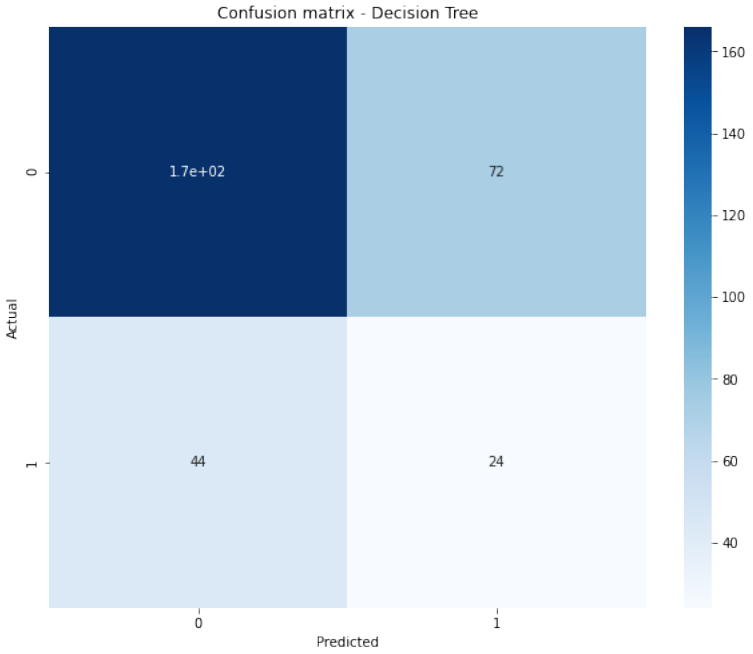
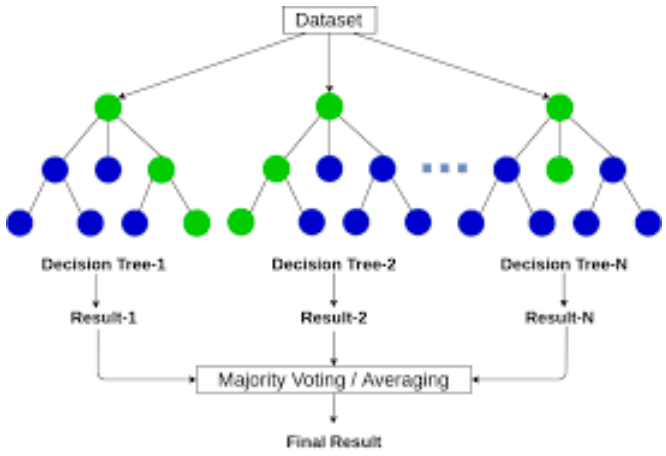


Figure 5.7. Random forest algorithm.

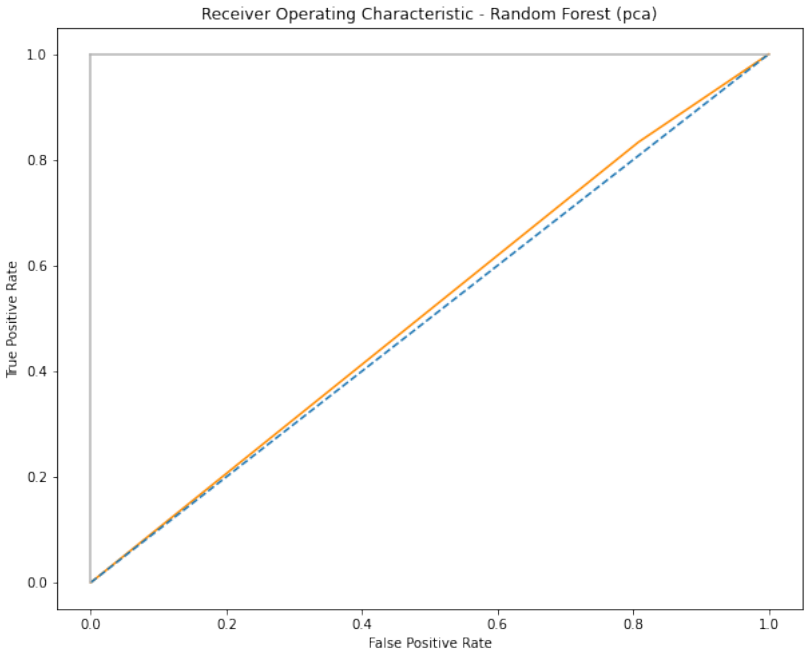


computing all distances. The choice of k is complicated: a small k means classification is more sensitive to outliers, while a big one can make the algorithm consider neighbors that are far apart and maybe irrelevant.

We trained the K-Nearest-Neighbors and got the following results on validation set:

We also report ROC curve (5.11) and Confusion matrix (5.12).

Figure 5.8. Random Forest ROC curve.



	precision	**recall**	**f1-score**	**support**
0	0.80	0.55	0.65	238
1	0.25	0.51	0.33	68
accuracy			0.54	306
macro avg	0.52	0.53	0.49	306
weighted avg	0.68	0.54	0.58	306

Table 5.5. KNN metrics.

Figure 5.9. Random Forest confusion matrix.

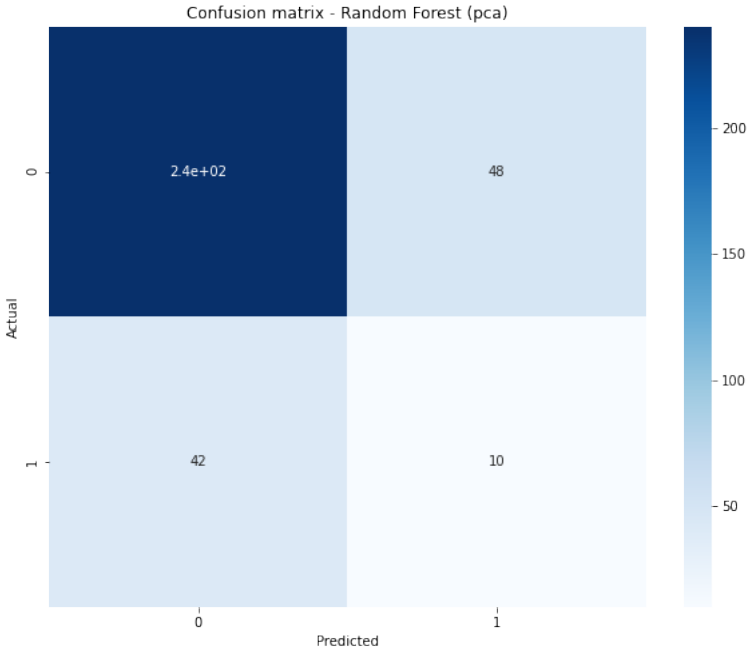


Figure 5.10. K-NN algorithm with k = 5.

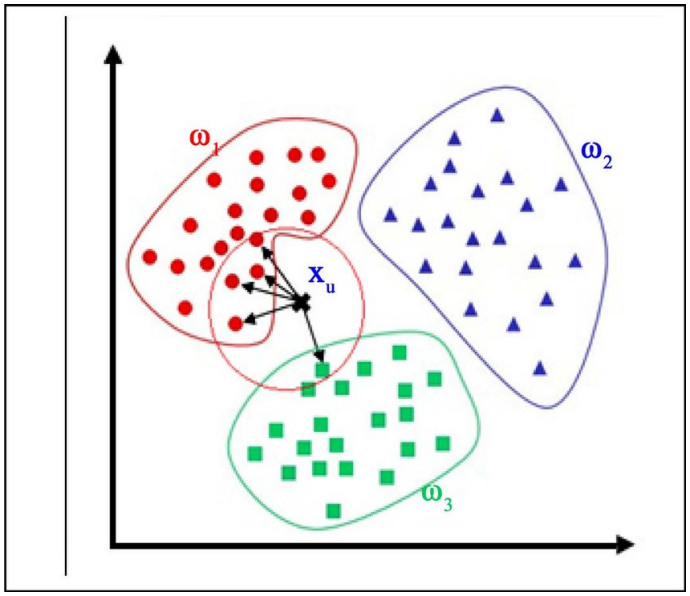


Figure 5.11. K-NN ROC curve.

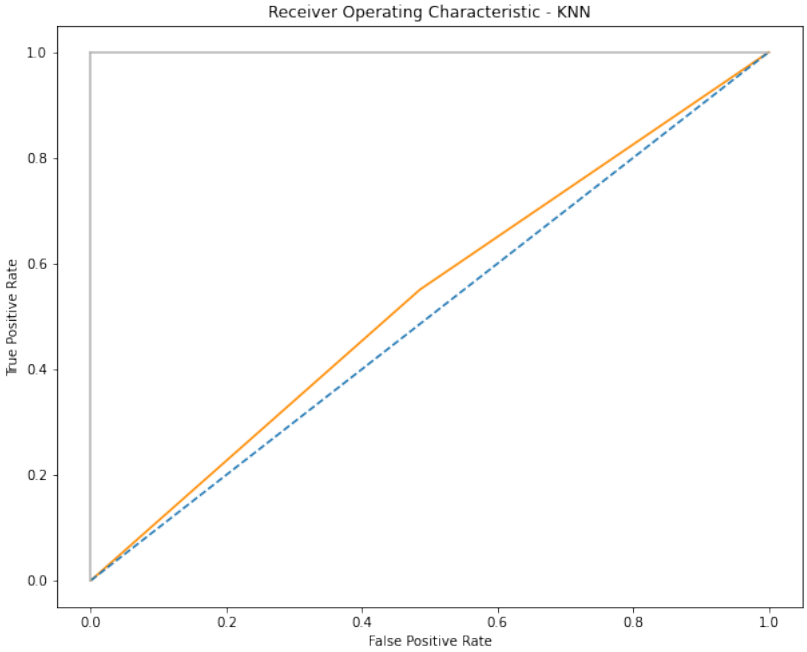
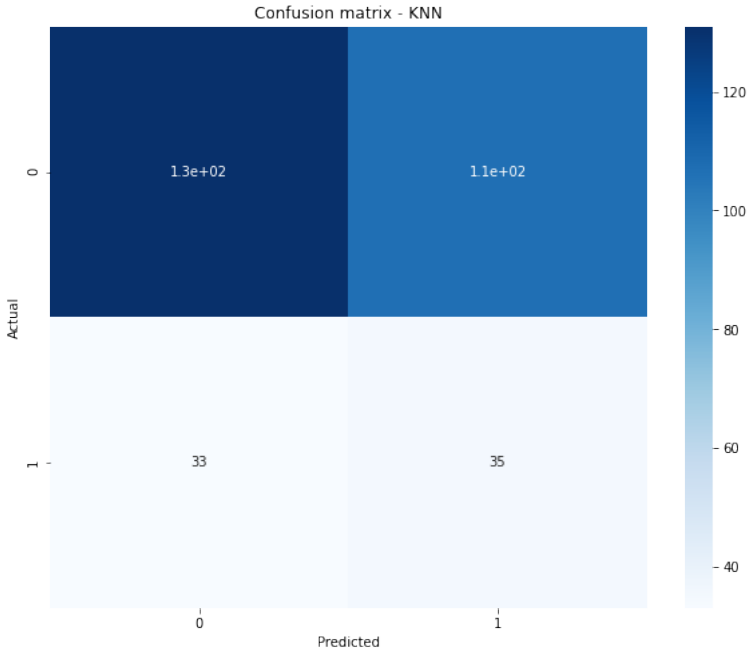


Figure 5.12. K-NN confusion matrix.



Chapter 6

Results

Model	Pipeline	f1-score
SVM	a	0.335
Logistic Regression	a	0.407
Decision Tree	a	0.293
Random Forest	b	0.353
KNN	a	0.333

Table 6.1. Results in terms of f1-score on validation set. For each model we reported the pipeline that works better. Pipeline a: Sequential Backward Selection + Z-score + SMOTE; Pipeline b: Sequential Backward Selection + Z-score + PCA.

In table 6.1 we reported the f1-scores obtained by different models, specifying the associated pipeline that works better for them. The metrics are calculated on validation set.

Looking at f1-score results, we decide to tune hyperparameters of Logistic Regression and Random Forest. We can do it manually and plot the behaviour of the model with different values of a certain parameter. For example for Random Forest in figure 6.1 we analyzed the impact of $n\_estimators$ parameter and we can see that the best value is 100 .

Otherwise we can use a grid-search: an exhaustive search over specified parameter values for an estimator.

As a final step we tested these two models on the test set, obtaining worse results than on the validation set. In particular, Logistic Regression gives an f1-score of 0.277 while Random Forest 0.233. This can be due to both overfitting and difference in test and valid sets' distributions.

Keeping in mind the aim of our predictive model, which is to identify patients that will suffer Chronic Heart Failure, we have to carefully compare the different confusion matrices of the different models. In particular, it is better to predict someone to suffer of this complication, if it will not happen, rather than predict someone not to have it, when he will. Therefore we have to minimize the number of False Negatives (FN). In table 6.2

Figure 6.1. Impact of *n_estimators* for Random Forest in terms of f1-score.

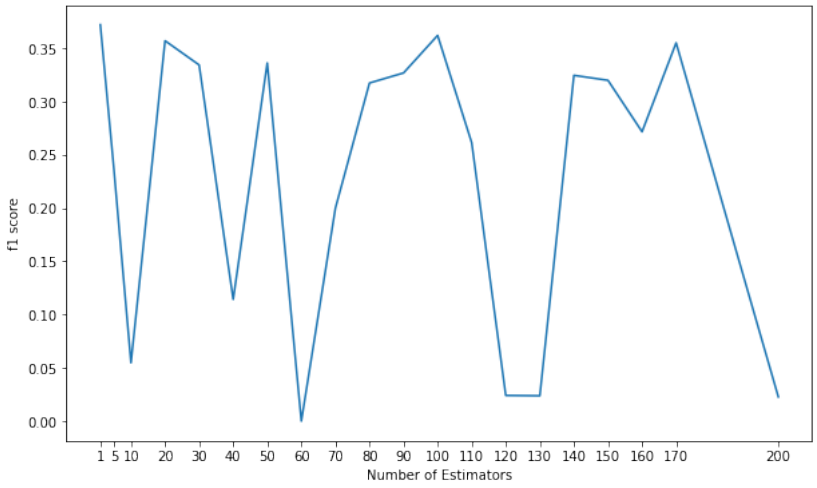


Table 6.2. False Negatives of the different models.

	FN
SVM	28
Logistic Regression	19
Decision Tree	44
Random Forest	42
KNN	33

we can see the number of FN for the different models. Logistic Regression has a lower number of FN with respect to the other models. SVM and KNN should be taken into consideration because, even if they have lower f1-score than Random Forest, they have a lower number of FN as well.

Bibliography

[1] *https://archive.ics.uci.edu/ml/datasets/Myocardial+infarction+complications*. 2020.

[2] Thomas Taimre Radislav Vaisman Dirk Kroese, Zdravko Botev. *Data Science and Machine Learning*. 2020.

[3] Lawrence Hall Philip Kegelmeyer Nitesh Chawla, Kevin Bowyer. Smote: Synthetic minority over-sampling technique. *https://arxiv.org/pdf/1106.1813.pdf*, 1607.

[4] Shai Ben-David Shai Shalev-Shwartz. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.