Heart disease prediction

Mathematics in Machine Learning course project

Paolo Calderaro
s288831

Politecnico di Torino

Francesco Vaccarino
Mauro Gasperini

Contents

1 Introduction

The aim of this project is to develop a machine learning pipeline to detect the presence or absence of heart disease based on medical patient data. The analysis consisted in the development of a pipeline composed of the following steps:

- data cleaning
- categorical features handling
- features scaling
- outliers detection and removal
- dimensionality reduction
- resampling technique for unbalance data
- K-fold cross validation for hyperparameters tuning

The models tested for the classification task were KNN, decision tree, random forest and support vector machine. The work propose also mathematical insights about the algorithm used. The code used to analyze the dataset and train the model is available here.

2 Data exploration

2.1 Dataset description

The original dataset (that can be found in UCI archive) is composed with patient's data coming from 4 different databases: Cleveland, Hungary, Switzerland, and the VA Long Beach. Those database contains a total of 76 attributes for each patient. By going through the unprocessed data in the data repository we can notice that a lot of entries has missing information for most of the attributes. This could happen either because of errors during the data collection or because some attributes represents medical information that requires tailored analysis to gather and that might not have been necessary for the specific patient. Despite that, the official dataset repository provide a processed version to a subset of 14 attributes with 303 patient data, hence we assume that this is final and usable version of the dataset. All the already published experiments refers to this processed version too. No further details about the data collection are provided. Here it follows the set of attributes we will use for the task. The list contains the attributes with the original names from the dataset, together with a brief description:

numerical

- age: The person's age in years

- trestbps: The person's resting blood pressure (mm/Hg on admission to the hospital)
- thalach: The person's maximum heart rate achieved
- ca: The number of major vessels (0–3)
- chol: The person's cholesterol measurement in mg/dl
- oldpeak: Anomaly in the ECG (referred as ST depression) induced by exercise relative to rest

categorical

- sex: The person's sex
- cp: chest pain type. 4 different values:
  - Value 1: typical angina[1]
  - Value 2: atypical angina
  - Value 3: non-anginal pain
  - Value 4: asymptomatic
- fbs: fasting blood sugar, blood sugar after an overnight fast (without eating). The attribute is a boolean variable (if ¿ 120 mg/dl , fbs = 1; else fbs = 0)
- restecg: resting electrocardiographic results, 3 values:
  - Value 0: normal
  - Value 1: having ST-T wave abnormality
  - Value 2: showing probable left ventricular hypertrophy
- exang: Exercise induced angina, chest pain that occurs with exercise or physical stress (1 = yes; 0 = no)
- slope: the slope of the peak exercise ST segment -
  - Value 1: upsloping
  - Value 2: flat
  - Value 3: downsloping
- thal: A blood disorder called thalassemia. 3 possible values:
  - Value 3: normal blood flow
  - Value 6: fixed defect (no blood flow in some part of the heart)

[1]Angina is chest pain caused by reduced blood flow to the heart muscle. Source: nhs.uk

  - Value 7: reversible defect (a blood flow is observed but it is not normal)

- target: presence or absence of heart disease in the patient (angiographic disease status, 0 = no disease, 1 = disease)

In order to check if more data could have been used, we filtered the data coming from the other 3 unprocessed file available in the data repository. Unfortunately, most of the patient data had missing information for one or more attribute of interest, hence we could not expand our dataset more. For the target attribute originally the dataset propose 4 different values as target: 0 for presence of heart disease and values from 1 to 3 depending on the intensity of the disease. Since we care only the presence or absence of the disease we will consider values from 1 to 3 as belonging to one single disease class, as specified in the attribute description.

2.2 features distribution

In figure 1 is possible to visualize how the numerical attributes are distributed while in figure 2 we have the frequencies of the categorical attributes. Each attribute is displayed by target label in order to compare the healthy and disease state and display some differences. This exploratory phase allow us to make considerations on the data. We can notice that the average age is higher in patient that had a disease. Most of the patient with heart disease are men, while women present a much lower percentage of disease. Same applies for *chol* that states the cholesterol level, which is slightly skewed towards higher values in patient with disease. Usually high cholesterol values are indicator for bad health condition, and this could relate to the presence of hearth disease. We do not have enough information to make considerations on the other attributes which requires much deeper field knowledge.

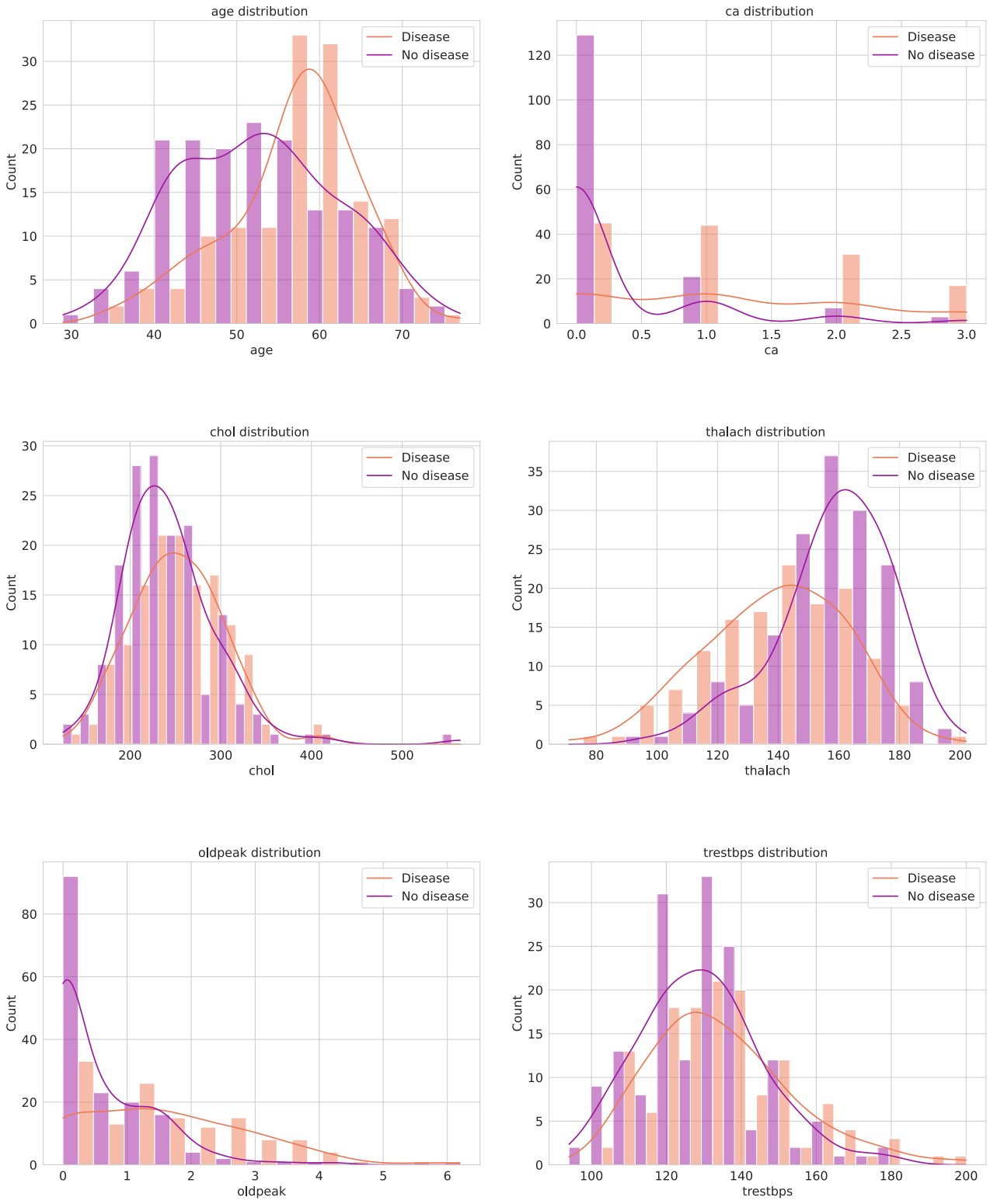
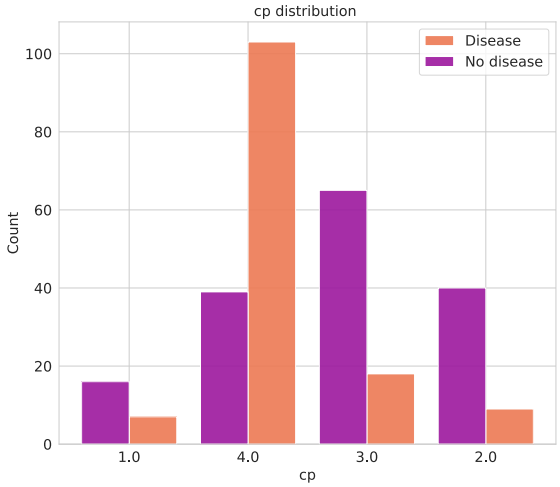
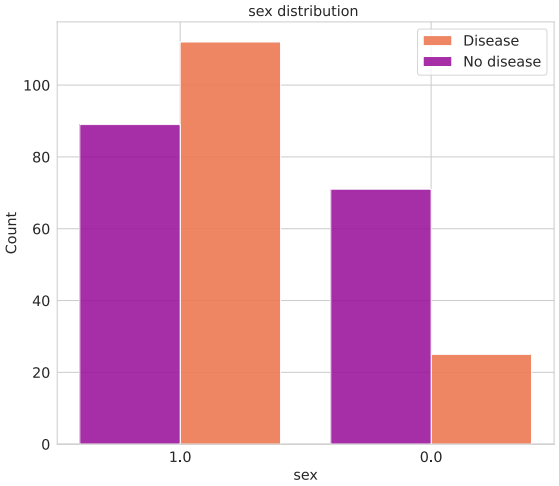
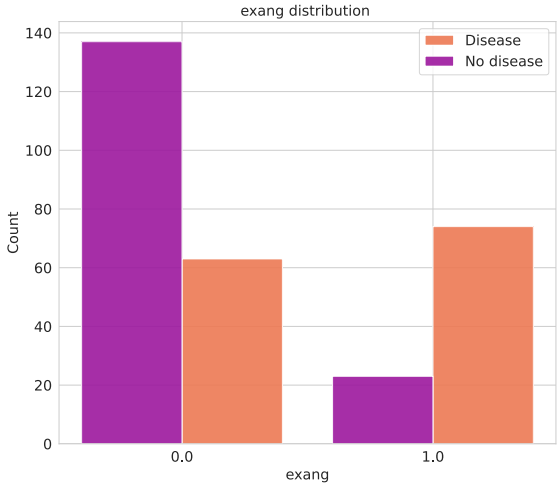
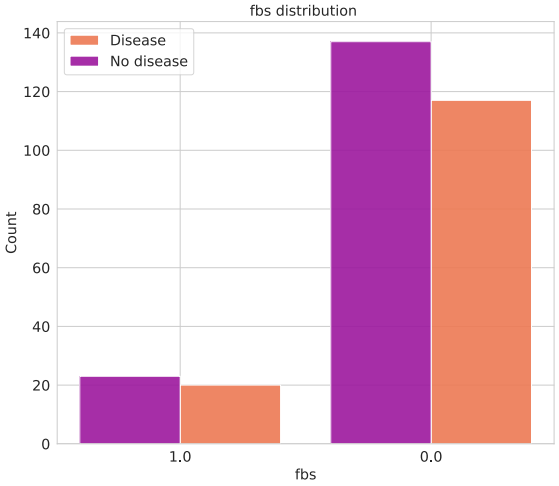


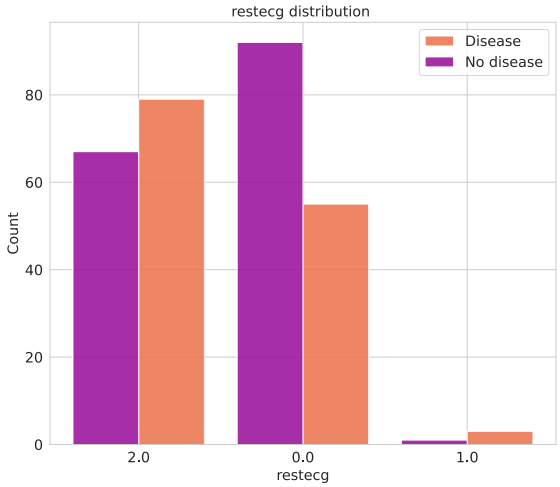
Figure 1: Distribution of the numerical attributes with respect to the target.

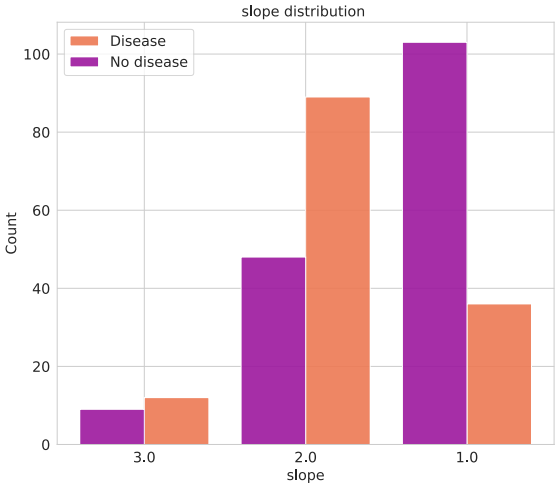

cp distribution
Disease
No disease
Count
cp
1.0
4.0
3.0
2.0


sex distribution
Disease
No disease
Count
sex
1.0
0.0


exang distribution
Disease
No disease
Count
exang
0.0
1.0


fbs distribution
Disease
No disease
Count
fbs
1.0
0.0


restecg distribution
Disease
No disease
Count
restecg
2.0
0.0
1.0


slope distribution
Disease
No disease
Count
slope
3.0
2.0
1.0

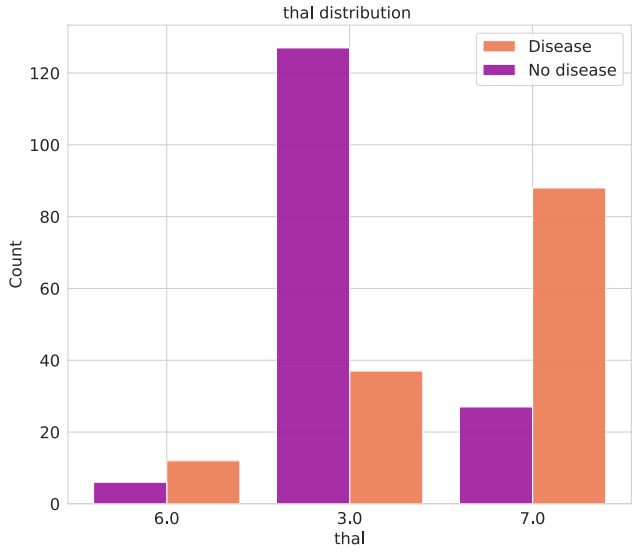


Figure 2: Occurencies of the categorical attributes with respect to the target.

2.3 Target distribution

One of the first aspect to check before starting developing the pipeline is if we are dealing with a balanced or unbalanced dataset. We have a balanced dataset if the dataset has almost the same number of sample belonging to each class.

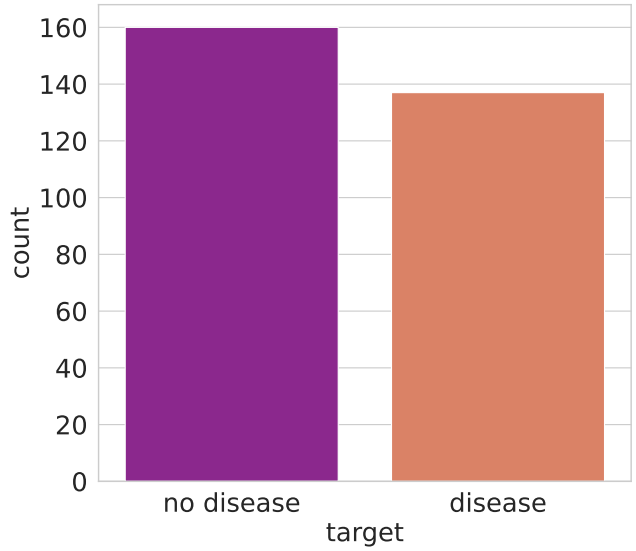


Figure 3: Target distribution.

In case of an unbalanced dataset we need to take some actions during the preprocessing steps in order to mitigate the effects that an unbalanced dataset could cause: some machine learning algorithm are sensitive to proportions of classes and tend to have worse performance

on the minority class. We can notice from figure 3 that data is slightly skewed towards the no disease class with 53% of the data belonging to that class. The dataset is then slightly unbalanced: corrective measures will be taken in order to deal with this gap but we do not expect those to bring significative differences in the performances.

2.4 Features correlation

Further insights about the features relationship can be inferred through the analysis of features correlation. Correlation is a statistical measures that quantifies how much two variables are linearly related to each other. The "Pearson's correlation coefficient" is the most common measure of correlation that tell us if two variables are positively or negatively correlated:

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} \tag{1}$$

Where $Cov(X,Y)$ is the covariance of the variables X and Y, σ_X and σ_Y are their standard deviations. Considering n as the number of instances, we can estimate those parameters in the following way:

$$Cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_X)(y_i - \mu_Y) \tag{2}$$

$$\sigma_X = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_X)^2} \tag{3}$$

$$\sigma_Y = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \mu_Y)^2} \tag{4}$$

With μ_X and μ_Y expected values of the variables X and Y. ρ can assume values between -1 (strong negative correlation) and 1 (strong positive correlation). A value of 0 does not ensure the independence among the two variables since we are measuring just the linear dependence among the two. In figure 4 we present the pairwise correlation among the numerical features of the dataset.

We notice a negative correlation among the attribute *age* and *thalach* (person's maximum heart rate achieved). This means that the maximum heart rate decrease as the age of the patient increase, and this was expected since it is well established that older people has lower heart rate The same happen with the *oldpeak* attribute and again the *talach* attribute. The *oldpeak* attribute indicates the level of anomaly found in the ECG. This time we can not state if the negative correlation with the *talach* reflects the reality since it would require field-specific knowledge.

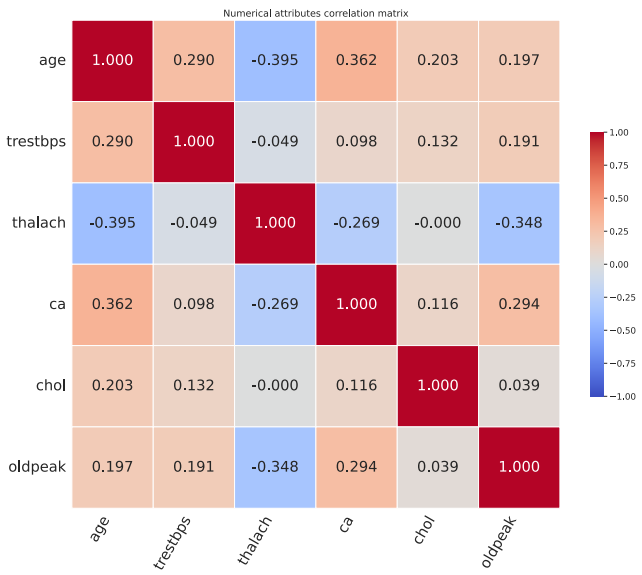


Figure 4: Pairwise correlation among numerical features.

3 Data preprocessing

The data pre-processing step is fundamental in the development of reliable machine learning pipeline. In this section we describe all the preprocessing steps performed in our pipeline to transform data.

3.1 Data cleaning

The first step is to check if the dataset contains null values or dirty entries. In our case the dataset it comes almost clean. There are only 6 entries that contains the character ? instead of the *thal* and *ca* attribute. This is probably due to missing information or errors in the measurements. Since we can not estimate those two values starting from other attributes, we decided to remove the dirty entries reducing the dataset to a total of 297 samples. No further cleaning operations were needed.

3.2 Categorical variables handling

As showed in section 1 we have a total of 13 features, 6 of which are categorical. In order to be able to feed categorical features into the majority of the algorithm we need to convert those into numerical features. We have two major strategies when it comes to encoding categorical features:

- **Ordinal encoding**: we transform each category to one integer ranging from 0 to $n.categories - 1$
- **One-hot encoding (OHE)**: we add a new binary attribute for each unique category

that states if the data entry belong (value 1) or do not belong (value 0) to that specific category

By default the categorical attribute are encoded with ordinal encoding with arbitrary value. The main drawback of using this encoding is that the machine learning algorithm could infer some notion of ordering behind those numbers even when not present. For that reason, one-hot encoding is exploited in this work for numerical encoding.

3.3 Outlier detection: LOF

Outliers are data points that are far from the others and are not coherent with the distribution of the whole dataset. Those entries might be present for measurement error, noise or novelty in the data. Outliers might mislead and lead to inaccurate outputs a machine learning model: for this reason is safe to detect and remove the outliers before feeding the data into a machine learning model. Several techniques for outliers removal are available in the literature. For this project, LOF (Local Outlier Factor) [2] algorithm has been exploited.
The LOF algorithm compute the local outlier factor which consist in a score that measure how much a point is isolated in its neighbourhood. The terms "local" refers to the fact that this strategy identifies outlier based on the density of the neighbourhood rather then "global" approaches that consider the entire dataset. In order to define the LOF value, we need to introduce the following quantities:
k-distance. we define $kdistance(p)$ the distance between a point p and a point o such that:

- for at least k objects it holds that: $d(o', p) \leq d(o, p)$
- For at most k1 object it holds that: $d(o', p) < d(o, p)$

reachability distance. The reachability distance between a point p and a point o is:

$$\text{reachdist}_k(p, o) = \max\{k\,\text{distance}(o), d(p, o)\}$$

The concept of reachability distance help us define the local reachability density, which is the inverse of the average reachability distance of the k neighbors of p:

$$\text{lrd}_k(p) = \frac{1}{\left(\frac{\sum_{o \in k}\ \text{reachdist}\ _k(p,o)}{|k|}\right)}$$

We can finally formally define the local outlier factor (LOF) as:

$$LOF_k(p) = \frac{\sum_{o \in k} \frac{\text{lrd}(o)}{\text{lrd}(p)}}{|k|}$$

The LOF is then the average of the ratio among the local reachability density of p and its k neighbours. If the density of a point is much lower then the one of its neighbors then the point is isolated and far from a dense area and it can be labelled as an outliers. This is

reflected by seeing that if the local reachability of p is low then the LOF value is high, same if the local reachability of the k neighbours rise. Points inside a cluster will most likely have low value of LOF and should not be labelled as outliers. A representation of this can be seen in figure 5. As we can imagine the parameter k need to be set with fine-tuning. The official paper suggest to use $k > 10$ since otherwise the algorithm will label as outliers points even if those are generated from a uniform distribution.

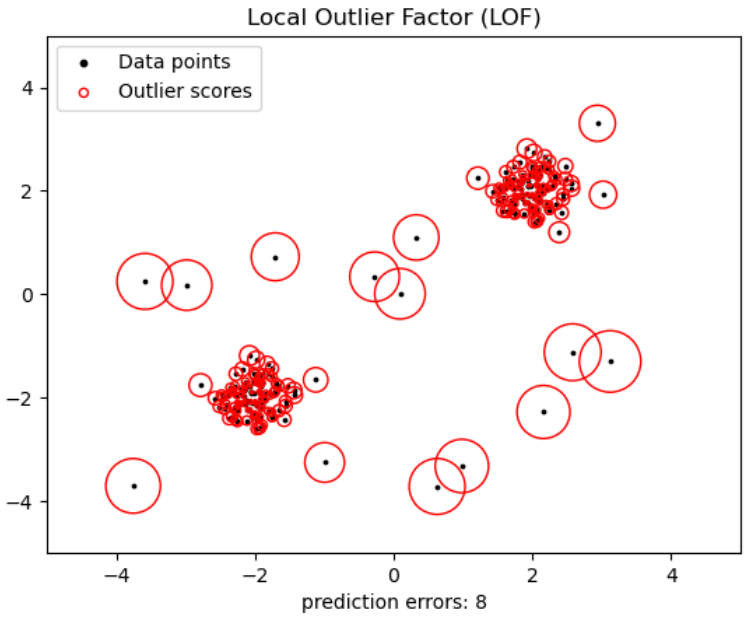


Figure 5: The red circle indicates the LOF score of each data point. The larger the circle, the higher the LOF and the likelihood to be labelled as outlier.

3.4 Features scaling

As we can see from figure 1 the numerical features present different distributions and magnitude. The different magnitude could be problematic for machine learning algorithm based on distance computation since the algorithm will give an unfair importance to all of them. There are several techniques to have features on the same scale. In this work normalization (or min-max scaling) and standardization has been exploited.

3.4.1 Normalization

We can normalize data by computing the new point as:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{5}$$

In this way we are forcing data in a $[0, 1]$ range. The drawback of normalization is that is naturally affected by outliers but since we are applying the LOF algorithm in our pipeline we are already handling this problem.

3.4.2 Standardization

Standardization is one of the most common scaling technique. We can apply standardization to a data point by removing the mean μ and dividing by the standard deviation σ of the feature:

$$x^{'} = \frac{x - \mu}{\sigma}$$

This operation force the feature distribution to look like a gaussian with zero mean and unit variance. Since it center the data standardization is also suggested when applying the dimensionality reduction technique discussed in section 7.
During the experiments the differences in performance between standardization and normalization were non significative, hence standardization was used for all the experiments in order to ensure a fair comparision among the different preprocessing configuration.

3.5 Oversampling: SMOTE

As showed in section 2 the dataset present a partial imbalance towards the no disease class. Many machine learning algorithm make more accurate prediction if trained with balanced data, hence we need to address the dataset unbalance. We have two main strategies to handle this problem: undersampling or oversampling.
Undersampling consist in balancing by keeping all the entries in the minority class and discarding entries of the majority class. This technique lead of course to loss of information and it is mostly suggested when we have a large dataset and we can afford that loss. For those reason we need to discard this option.
With oversampling we increase the number of sample of the minority class by sintetically generating new entries. Depending on how we decide to generate the new samples we have different techniques. The algorithm exploited for this work is SMOTE (Synthetic Minority Over-Sampling Technique). This technique was firstly introduced in 2002 [1]. Figure 6 shows how the algorithm choose to generate a new sample. New samples are generated in the following way:

- Extract a random sample from the minority class
- Take the difference between the sample and its nearest neighbors
- Multiply the difference vector by a random number in $[0, 1]$ range and add it to the starting sample

In this way we are generating new samples that lies between the extracted sample and its neighbor. In this way we do not duplicate data and we make sure that the new samples are not too different from the one in the dataset. Since our dataset is made up of both categorical

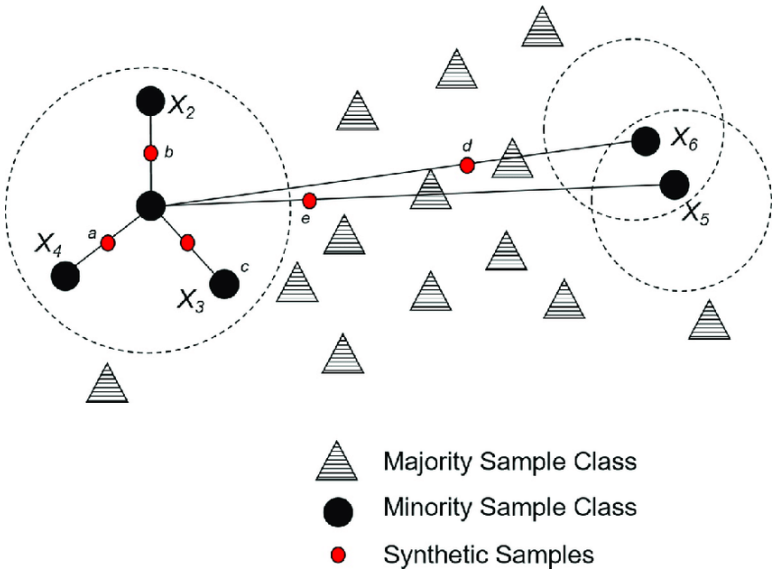


Figure 6: Generation of syntetic point exploiting SMOTE algorithm.

and numerical features we adopted the SMOTE-NC algorithm (SMOTE for Nominal and Continuous). This slightly modified version does not interpolate on the categorical attributes and the new samples belong to the same categories as the starting sample.

3.6 Dimensionality reduction

With dimensionality we refers to the number of features we have in a dataset. The more features we have the more likely the data will be sparse in the features space. We can have an idea from the image: This problem is referred in machine learning as curse of dimensionality since it could make the classification task harder: two data points might be similar in low dimension space but the more the dimension the further these points will look since the volume of the spaces increase. In order to counteract such an issue we employed PCA (Principal Component Analysis) in our analysis, an unsupervised algorithm for dimensionality reduction.
The aim of PCA method is to project our data in a lower dimension space to reduce the dimensionality of our dataset and, at the same time, retain as much information as possible. PCA combines highly correlated variables together in order to form a smaller set of attributes referred as principal components that maximize the variance of each component. Given a dataset composed of $\mathbf{x}_1, \ldots, \mathbf{x}_m$ be m vectors in $\mathbb{R}^d$, where d is the number of features, the aim is to find $\boldsymbol{x} \rightarrow W\boldsymbol{x}$ where $W\boldsymbol{x}$ is the lower dimensionality representation of $\boldsymbol{x}$. If we introduce also $U \in R^{d,n}$ as the matrix to recover from the reduction the problem that we are trying to solve can be expressed as:

$$\underset{W \in R^{d,n}}{\operatorname{argmin}} \sum_{R^{d,n}} \sum_{i=1}^{m} \|\boldsymbol{x_i} - UW\boldsymbol{x_i}\|_2^2 \tag{6}$$

it can be proven that if we find a solution (U, W) to 6 then $W = U^T$ and the problem can be reformulated as:

$$\underset{U \in R^{d,n} UU^T = I}{\operatorname{argmin}} \sum_{i=1}^{m} \left\| x_i - UU^T x_i \right\|_2^2 \tag{7}$$

We can further simplify the expression to optimize with some algebric operations in the following:

$$\left\| \boldsymbol{x} - UU^T \boldsymbol{x} \right\|^2 = \|\boldsymbol{x}\|^2 - \operatorname{trace}\left(U^T \boldsymbol{x}\boldsymbol{x}^T U\right) \tag{8}$$

Where *trace* is the sum of the diagonal entries. The optimization problem can be written in this final form:

$$\underset{U \in R^{d,n} : UU^T = I}{\operatorname{argmax}} \operatorname{trace}\left(U^T \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^{\boldsymbol{T}} U \right) \tag{9}$$

The term $\sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_i^\top$ can be seen as a symmetric matrix A that can be written as $A = VDV^T$ where D is a diagonal matrix containing the eigenvalues of A and columns of V contains the eigen vector. We can finally claim that the solution to 9 is the matrix U whose columns are the n eigenvector corresponding to the largest n eigenvalues. This means that PCA creates components with decreasing eigenvalues which represent the explained variance of the data. The final set of components that we obtain has then high variance and will be uncorrelated to each other since they are orthonormal.

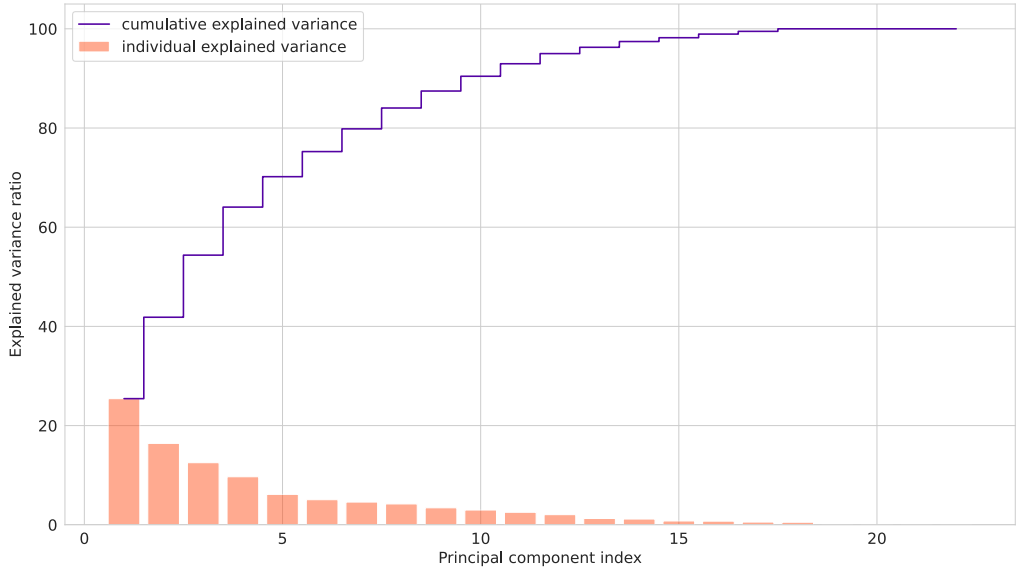


Figure 7: Variance explained of 22 components (number of features after OHE) and cumulative explained variance.

In order to choose properly the number of components we can plot the cumulative explained variance together with the explained variance of each possible components, as shown in figure 7. We then keep the amount of components that are able to capture most of the variance of the dataset. This consist in looking for a knee in the cumulative explained variance curve: in our case with 8 components we are able to catch more then 80% of the variance.

4 Training settings and evaluation

Before delving into describing the model deployed for the classification task here it follows a section that describes the strategies used to do hyper-parameters tuning of the models and to evaluate the results. Before starting with hyperparameters tuning we splitted the dataset into a train set and test set with a 80/20 proportion. The train set will be first used to tune the hyperparameters of the models by cross-validation and then, once we find the optimal combination, the fine-tuned models will be trained on the whole train set and deployed on the test set. The performances will be evaluated on the test set, that will be used only once the models will be completely tuned and trained so that we make sure that the model are tested on completely new data points.

4.1 K-fold cross validation

After partitioning the dataset into train and test set, we reduced the number of data points available to perform hyperparameters tuning of the models. The K-fold cross validation technique is able to partially overcome this problem. In the K-fold cross validation procedure the dataset is partitioned in k subset, called folds.

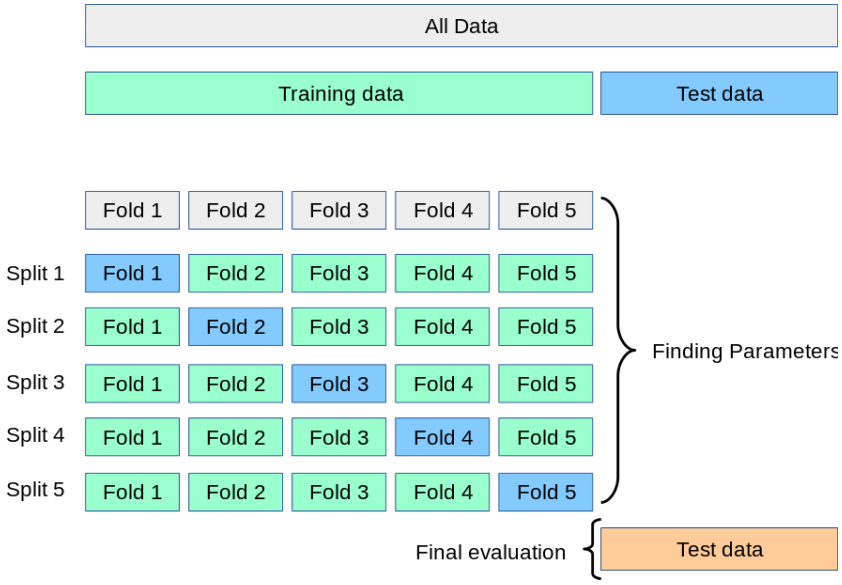


Figure 8: Partitioning with a 5-fold cross validation. The blue fold is used for validation while the green folds are used to train the configuration.

Iteratively for each fold the algorithm is trained on the union of the remaining folds and then validated on the current fold. The final estimate of the error will be the average of the errors measured on each fold. In this way we use all the data points for train and validation exactly once (the folds are made without replacement) and this help us in case such ours in which we have very few data. In our study the value of k is set to k=5. We also decided to implement the stratified version of the technique which ensure that in each fold the percentages

of classes is approximately the same as the original dataset. During cross validation each pre-processing step that required a training step (e.g.: scaler, PCA) was trained only on the current training set and not on the validation fold in order to avoid data leakage that could mislead the validation results. The metrics used as performance metrics was the f1-score.

4.2 metrics

The following section provide a description of the metrics used to evaluate the performances of our model. Since we are dealing with a binary classification task we can define the following quantities:

- TP True Positive: sample with true label=1 (positive class) correctly classified
- TN True Negative: sample with true label=0 (positive class) correctly classified
- FP False Positive: negative sample classified as positive
- FN False Negative: positive sample classified as negative

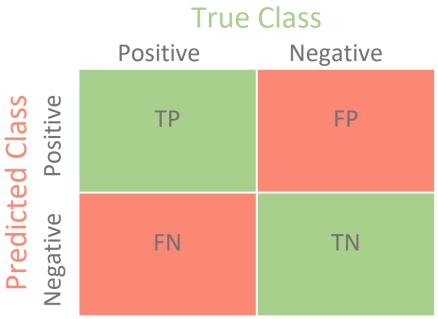


Figure 9: Confusion matrix.

A good way to represent those values is the confusion matrix showed in figure 9 in which each row represent the predicted class while the columns represent the true label. Starting from those quantities we can define the following metrics:

- **Accuracy**: the proportion of the number of correct prediction over the total number of predictions:
$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$
- **Precision**: the proportion of the correct positive prediction over the total of sample predicted as positive:
$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**: the proportion of the correct positive prediction over the total number of positive sample in the dataset:

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

- **F1-score**: Harmonic mean among precision and recall:

$$F1 = 2\frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

As said in the previous section, the metrics that will be referenced during the validation phase will be the f1-score.

4.3 ROC curve

A ROC curve (Receiver Operating Characteristic) is a visualization that shows how well a binary classifier performs. It is constructed by plotting the *recall* on the x-axis (also referred as TPR, True Positive Rate) and the FPR (False Positive Rate) on the y-axis that can be computed as $\frac{FP}{TN+FP}$. Classifiers that gives curves closer to the top-left corner indicate good performances. This correspond to the event of having all the data points being correctly labelled as true positive or true negative.

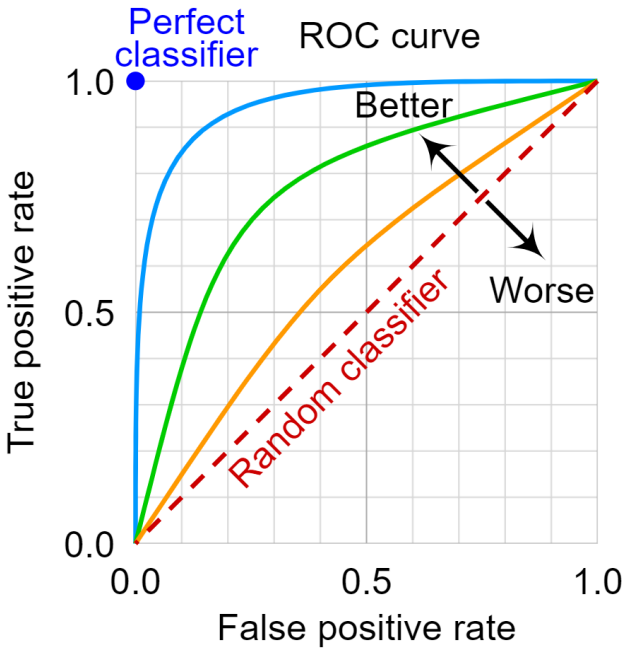


Figure 10: Different examples of ROC curve cases.

The more our classifier gets wrong prediction the more the curve will shift towards the bisector line, which represent a random classifier. A measure to quantify the goodness of the ROC curve is the area under the curve, referred as AUC.

5 Models

5.1 Decision tree

A decision tree predict the label of a data point by travelling from a root node of a tree to a leaf. Starting from the root, the next child is chosen by splitting the features space. Once the tree is built each leaf contain a specific label so it is possible to trace the actions that led to a specific class. This aspect underscore the main advantage of this algorithm: the results are simple to understand and interpret.

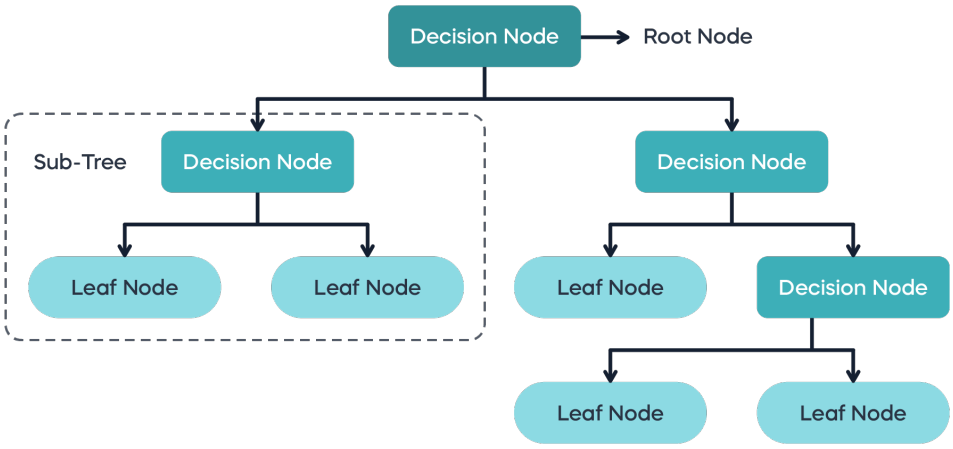


Figure 11: Scheme of a decision tree.

If we allow the decision tree to grow with an arbitrary size by splitting each leaf on thresholding values of the single features we could easily occur in learning a really large tree that may suffer from overfitting. That is why we usually prune the research by limiting the tree's depth. in order to build the tree greedy heuristics are used that work in an iterative way: on each iteration the effect of the splitting is analyzed and we decide a gain measure that quantifies the improvement due to the split. Then, among all the possible splits, we choose the one that maximize the gain or we decide to not perform the split at all and declaring the node as leaf. The splitting criterion analyzed for this project were the following:

- **Gini index**: $\text{Gini}(t) = 1 - \sum_j p(j \mid t)^2$ where $p(j \mid t)$ is the frequency of class j at node t. Once we are able to compute the Gini index for each node, we can evaluate the quality of the split as:

$$\text{Gini}_{\text{split}} = \sum_{t=1}^{k} \frac{n_i}{n} \text{Gini}(t)$$

  where k is the number of partition, n the total number of instances in the node to split and n_i the number of instance in the partition t. The lower score means good split with

low impurity degree since $Gini(t)$ has its minimum value when all record in a node t belong to one class.

- **Entropy**: $\text{Entropy}(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$. As the Gini, this is an impurity measure that quantifies the disorder among classes in a node. To measure the quality of a possible split with entropy we look at the reduction of entropy achieved after the split, the so called information gain:

$$\text{GAIN}_{\text{split}} = \text{Entropy}(p) - \sum_{t=1}^{k} \frac{n_i}{n} \text{Entropy}(t)$$

5.1.1 Results

We performed a grid search over the following parameters:

- tree maximum depth: $\{None, 5, 8, 10, 12, 20\}$
- split criterion: $\{"gini", "entropy"\}$

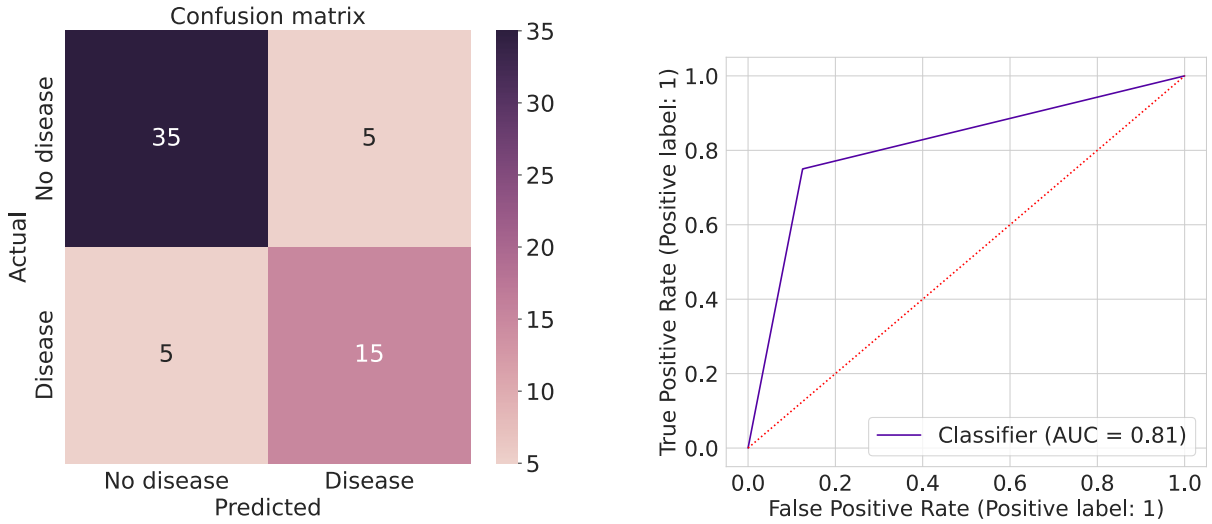


Figure 12: Confusion matrix and ROC curve of the best configuration.

the best configuration found by the grid search performed was $\{'criterion' :' gini',' max\_depth' : 20\}$ and scored a f1 of 0.74. We can see from figure 13 that the model benefits from all the preprocessing steps applied, scoring its best f1 when all the preprocessing steps are performed.

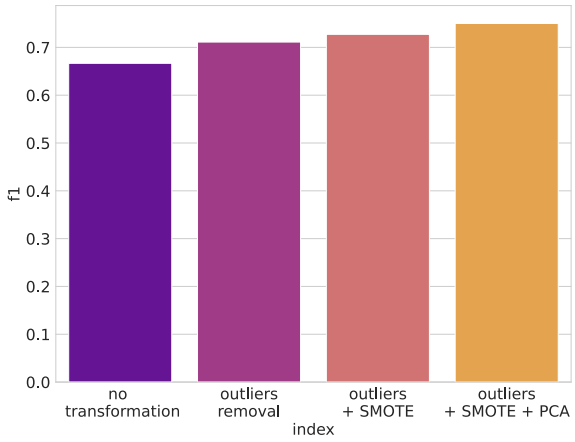


Figure 13: f1-scores of the different setups.

5.2 Random forest

Random forest is an ensemble of decision tree. Ensemble methods combine many simple methods in order to obtain a more powerful one and to reduce the danger of overfitting. The idea behind those type of method, also referred as bootstrap aggregation or bagging methods, is to combine multiple prediction functions learned from different and independent training set sampled from the population and then average the resulting predictions. This strategy improves the performance since if we consider n independent observation $Z_1, \ldots, Z_n$ each with variance σ^2 then the variance of the mean is given by $\frac{\sigma^2}{n}$, hence we reduce the variance of the observation by a factor of n. In order to generate n independent dataset we can use the bootstrap technique, which consist in sampling with replacement from an existing dataset in order to generate $\mathcal{T}_1^*, \ldots, \mathcal{T}_n^*$ dataset and train n separate models. Random forest then decorrelate the trees even further by randomly picking a subset of features during each tree construction. Generally the number of features m is $m \approx \sqrt{n}$. This strategy avoid to generate trees that are similar to each other and that would provide highly correlated predictions which would prevent the reduction in variance by averaging.

5.2.1 Results

We performed a grid search over the following parameters:

- number of trees: $\{50, 100, 200\}$
- maximum depth of each tree: $\{None, 4, 6, 10, 12, 14\}$

The best configuration found by the grid search performed was $\{'max\_depth' : 14, 'n\_estimators' : 200\}$ and scored a f1 of 0.77. Applying oversampling and PCA resulted in a slightly better performances, while the top score is obtained when only the outliers removal procedure is applied.

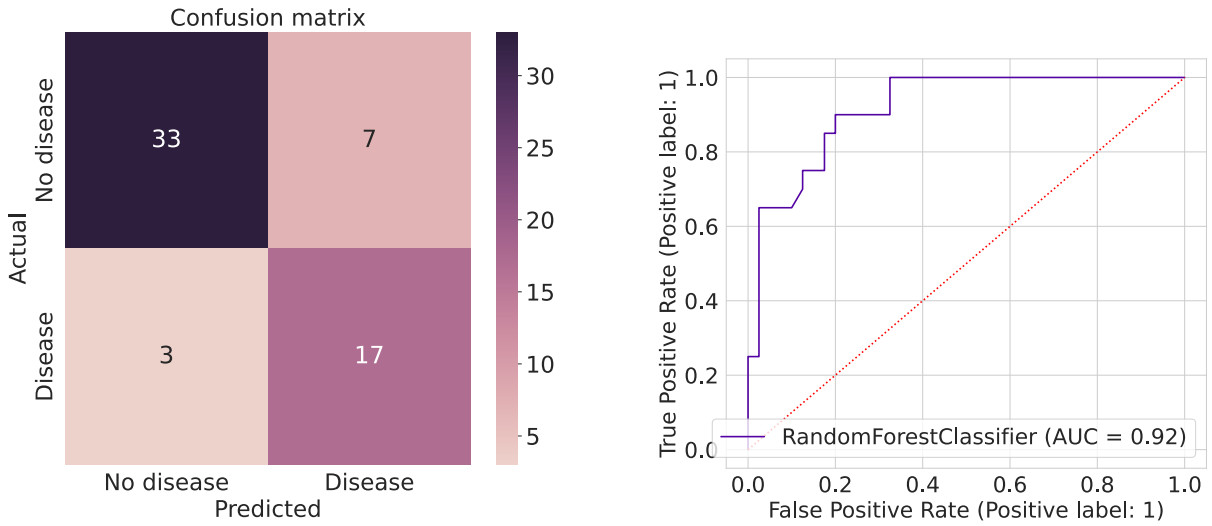


Figure 14: Confusion matrix and ROC curve of the best configuration.

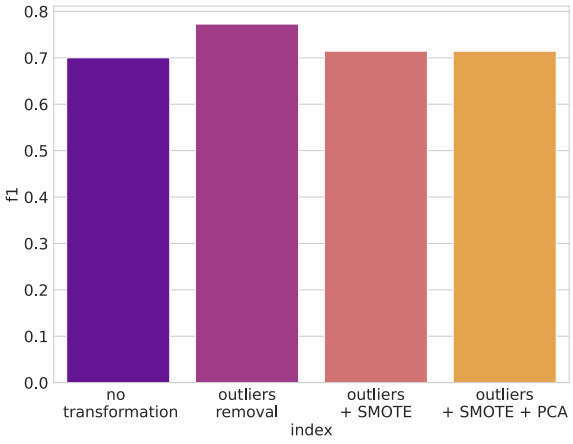


Figure 15: f1-scores of the different setups.

5.3 KNN

The idea behind K-Nearest Neighbors algorithm is to estimate the label of a new instance on the basis of the labels of its closest neighbors in the training set. Given a positive number K and a distance metric and let $\tau(\boldsymbol{x}) := \left\{\left(\boldsymbol{x}_{(1)}, y_{(1)}\right) \ldots, \left(\boldsymbol{x}_{(K)}, y_{(K)}\right)\right\}$ the K instances belonging to the training set closest to a new point x. Then the KNN algorithm classifies x according to the most frequently occurring label in $\tau(\boldsymbol{x})$. The distance that can be exploited for this algorithm are many. For this project we used the Euclidean and the Manhattan distance:

$$\text{Euclidean } = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \qquad \text{Manhattan} = \sum_{i}^{m} (|x_i - y_i|) \tag{10}$$

With m representing the number of features. The choice of K is fundamental for the behaviour of the algorithm: small values make the model sensitive to outliers while high values might lead to overfitting. One of the main drawback of this algorithm is that could become really expensive to compute all distances from the training samples if we have a large training set.

5.3.1 Results

We performed a grid search over the following parameters:

- number of neighbors: $\{'n\_neighbors' : [3, 5, 7, 10, 15, 20, 25, 30, 40]\}$
- weight: $\{"uniform","distance"\}$
- distance metric: $\{"euclidean","manhattan"\}$

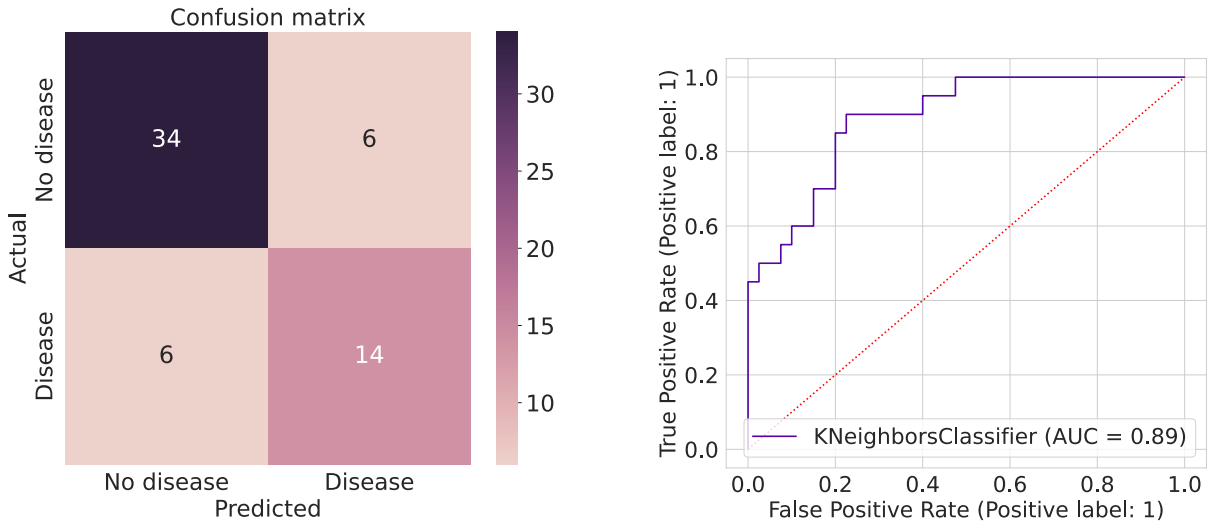


Figure 16: Confusion matrix and ROC curve of the best configuration.

The best configuration found by the grid search performed was $\{'metric' :' manhattan','n\_neighbors' : 40,'weights' :' distance'\}$ and scored a f1 of 0.7. We can see from figure 17 that the model performances improves when PCA is applied, while the outlier removal technique and over-sampling resulted in similar performances.

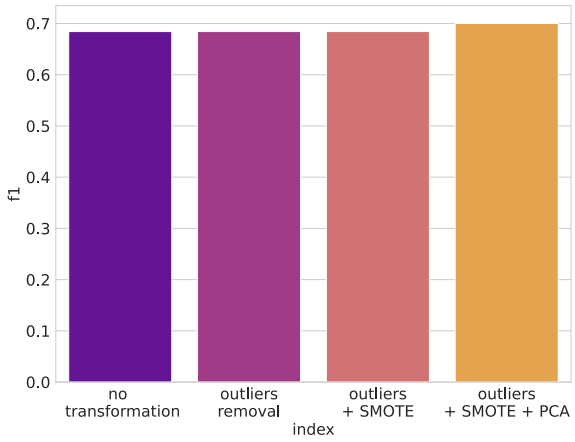


Figure 17: f1-scores of the different setups.

5.4 SVM

Support Vector Machine is a supervised paradigm that can be used for classification tasks. The main rationale behind SVM is to find an hyperplane (often referred as large margin separator) in the features space such that all the training samples belonging to the same class are on the same side of the features space and also far away from the separator.
Let $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be our training set where $y_i \in \{\pm 1\}$. We can say that this set is linearly separable if exist a halfspace $(\boldsymbol{w}, b)$ such that:

$$y_i = \operatorname{sign}\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right) \text{ for all } i \quad \text{or} \quad \forall i \in [m], \quad y_i \left(\langle \boldsymbol{w}, \mathbf{x}_i \rangle + b\right) > 0 \tag{11}$$

Since we have many possible halfspaces that satisfy this condition we need a criterion in order to choose the final one. Depending on the constraints we add to 11 we differentiate between hard-SVM and soft-SVM.

5.4.1 Hard-SVM

To understand what we mean by "hard" SVM we need to define the margin. The margin of an hyperplane can be defined as the minimal distance between the hyperplane itself and a point in the training set. An high margin means that training samples belonging to different classes are distant, hence even if we slightly perturbate the training samples we still have an hyperplane that satisfies 11. In Hard-SVM we choose the hyperplane that has the largedt possible margin. We can reformulate 11 adding this costraint:

$$\underset{(\mathbf{w},b):\|\mathbf{w}\|=1}{\operatorname{argmax}} \min_{i \in [m]} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| \text{ s.t. } \forall i, y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right) > 0 \tag{12}$$

Where $|\langle \boldsymbol{w}, \mathbf{x} \rangle + b|$ is the distance between a point x and an hyperplane $(\boldsymbol{w}, b)$. If we reformulate this problem as a quadratic optimization problem we can find a solution to 12:

$$(w_0, b_0) = \underset{(\boldsymbol{w},b)}{\operatorname{argmin}} \|\boldsymbol{w}\|^2 \quad \text{s.t.} \quad \forall i, y_i \left(\langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b\right) \geq 1 \tag{13}$$

the output of 13 is $\hat{\boldsymbol{w}} = \frac{\boldsymbol{w}_0}{\|\boldsymbol{w}_0\|}, \quad \hat{b} = \frac{b_0}{\|\boldsymbol{w}_0\|}$. This means that we are searching for $\boldsymbol{w}$ whose norm is minimal among all vectors that separates the data: minimizing the norm is equivalent to maximize the margin. This happens because we are forcing the margin to be 1 in order to and this is equivalent to rescale the problem by a factor of $1/\|\boldsymbol{w}\|$, hence the points closest to the hyperplane has now a distance of $1/\|\boldsymbol{w}\|$.

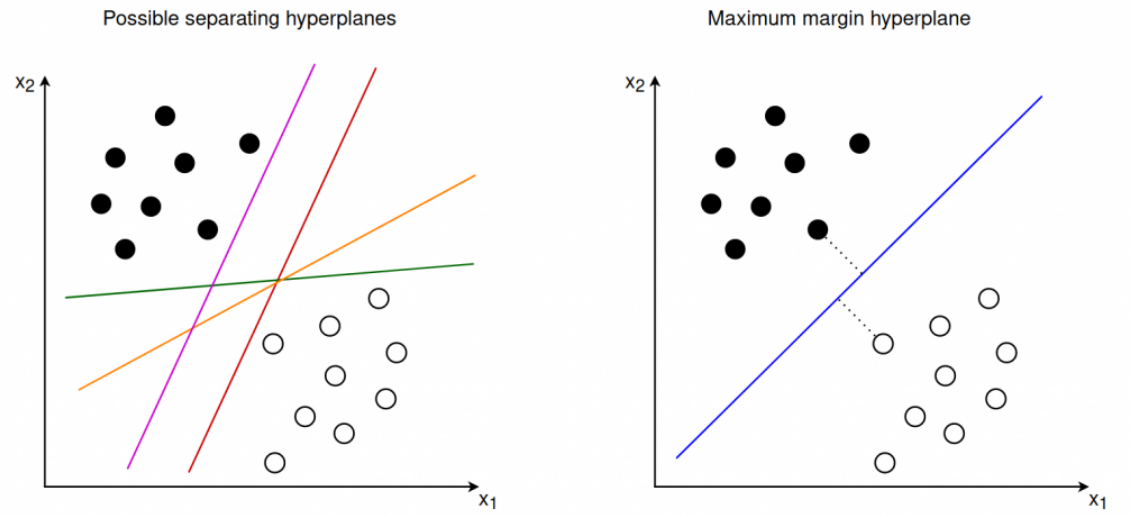


Figure 18: Constraint added by the hard-SVM: among all the possible hyperplanes (figure on the left) we choose one by maximizing on the margin (figure on the right).

5.4.2 Soft-SVM

The hard-SVM formulation is based on the strong assumption that the training set is linearly separable, which often is not the case. Soft-SVM counter this problem by relaxing the constraints in 12. The relaxation consist in allowing some training sample to violate the constraint and fall inside the margin. Mathematically this can be done by introducing slack variables:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \left(\lambda \|\boldsymbol{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right)$$

$$\forall i, y_i \left(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b \right) \geq 1 - \xi_i \quad \text{with} \quad \xi_i \geq 0$$

As we can notice the new optimization problem aims at minimizing the norm of w and the average of ξ_i which quantifies to the amount of violation of the costraint. the parameter λ balance the tradeoff between those terms.

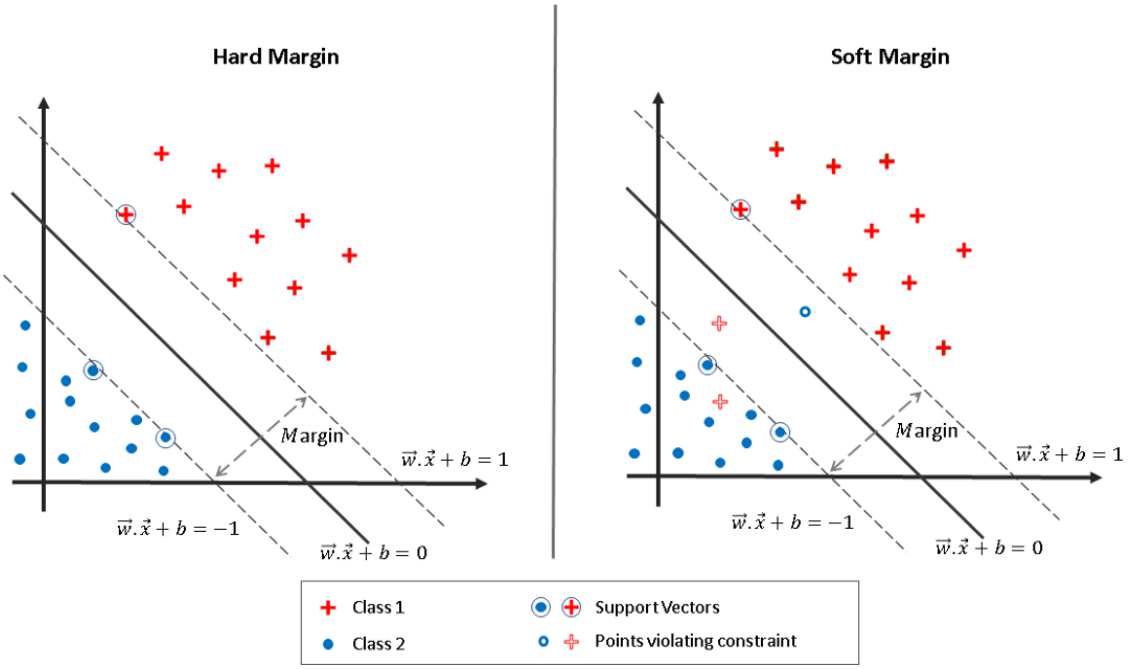


Figure 19: Main difference between hard-SVM and soft-SVM.

5.4.3 Kernel trick

In some cases data are not linearly separable at all. If this happen even the relaxed soft-SVM is not able to find an hyperplane. To overcome this limitation we can make further consideration if we consider the dual formulation of 12:

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq\mathbf{0}} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \left\langle \mathbf{x}_j, \mathbf{x}_i \right\rangle \right) \tag{14}$$

We can notice that the dual problem only involves inner product between instances $\langle x_i, x_j \rangle$: this key property allows us to apply a non-linear mapping into some feature space and learning the hyperplane in that feature space. We can introduce the so called kernel function $K(\boldsymbol{x}, \boldsymbol{x}') = \langle \psi(\boldsymbol{x}), \psi(\boldsymbol{x}') \rangle$ where ψ is a feature embedding into an Hilbert space. The advantage of using kernel function is that we can avoid to specify point in the space defined by ψ since we just need to define the inner product. This strategy is often referred as "kernel trick" and allows SVM to learn non-linear decision boundaries.

5.4.4 Results

We performed a grid search over the following parameters:

- kernel: $\{"linear", "poly", "rbf"\}$
- C: $\{0.1, 0.5, 1, 2\}$

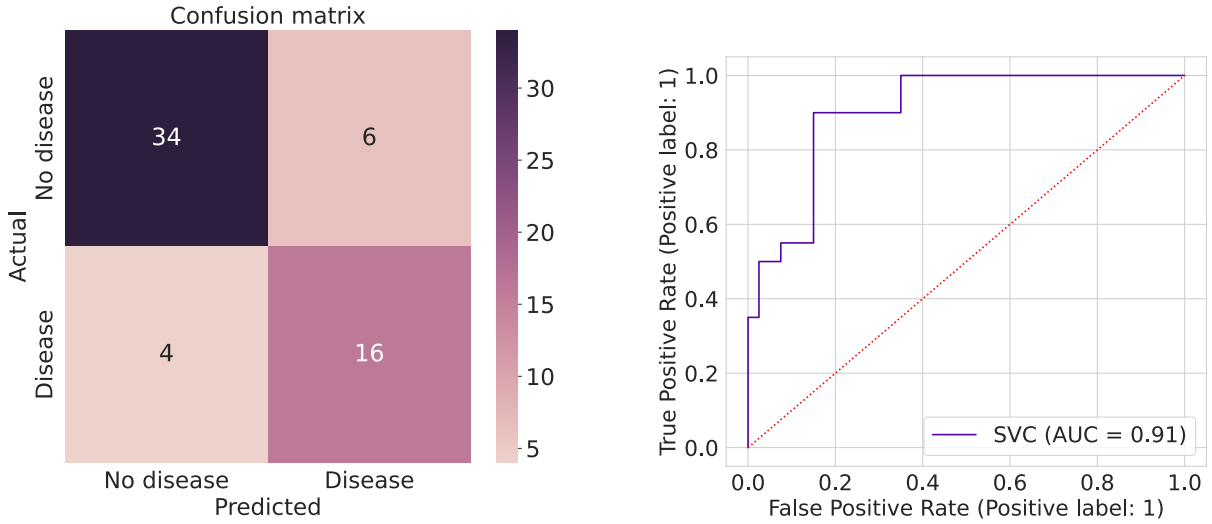


Figure 20: Confusion matrix and ROC curve of the best configuration.

The best configuration found by the grid search performed was $\{'C' : 2,' kernel' :' poly'\}$ and scored a f1 of 0.76. We can see from figure 21 that the model improves expecially when PCA is applied, while the outlier removal technique and oversampling bring slightly worse performances.

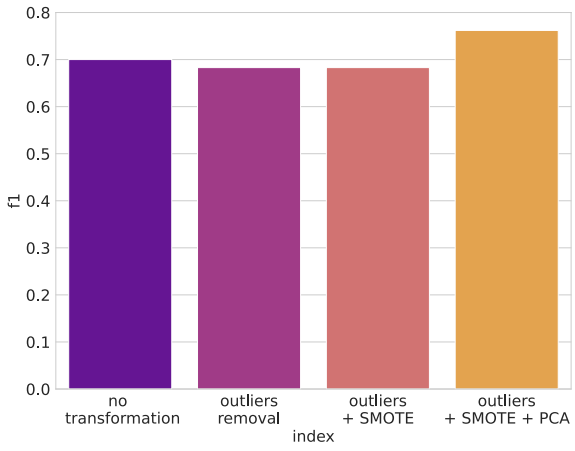


Figure 21: f1-scores of the different setups.

6 Conclusion

The aim of this work was to analyze how different machine learning algorithms could tackle this classification problem. The best performer model in terms of F1 score was the random forest with a score of 0.77 while the worst performer was the KNN with a f1 score of 0.7. We can see from figure 22 to figure 25 all the performance metrics for each of the best model configuration. By looking at the accuracy score we see that we obtain comparable results among the different models: if we do not care much about misclassified samples we can deploy a lighter model then random forest and we would obtain decent results. For this specific task, is way more important to keep a low amount of false negative since this would mean that the model is not discovering the presence of a disease: for this we should take into account the recall score, in which again the random forest algorithm performs best. From the random forest confusion matrix we can also notice that this model has the lowest amount of false negative among all the models.
Overall we can conclude the improvement brought by the preprocessing steps were secondary to hyperparameters tuning since the differences in the performance were not significative for all the model tested. In order to further confirm that a naive decision tree model was trained and tested with an 80/20 hold-out technique on the raw data and scored a f1 score of 0.6. Applying SMOTE was not effective probably because we are dealing with a low percentage of unbalanceness. Furthermore the LOF algorithm was able to remove at most 10 outliers, which means that our dataset did not contain a lot of outliers making this preprocessing step less useful. Overall we can confirm that we were in front of an "easy" dataset in which the preprocessing steps proposed did not brought the expected improvement in the performances.

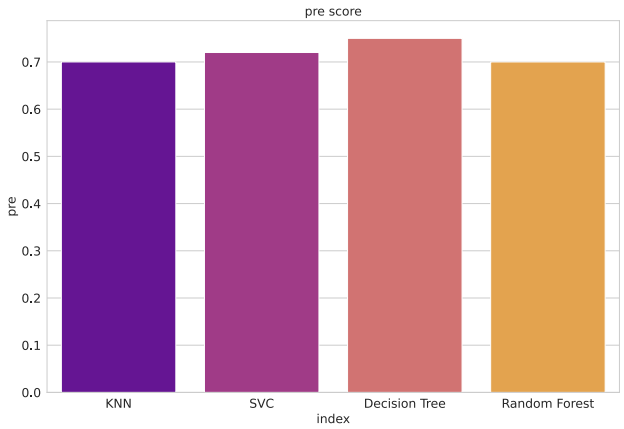


Figure 22: Precision scores.

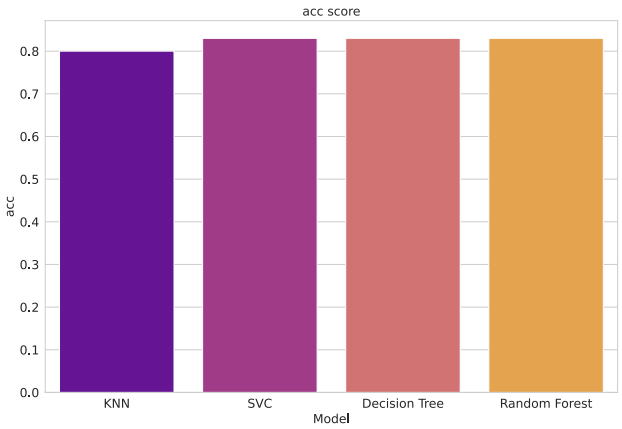


Figure 23: Accuracy scores.

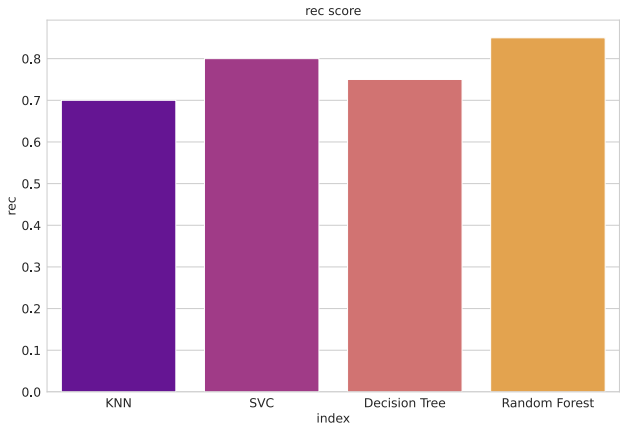


Figure 24: Recall scores.

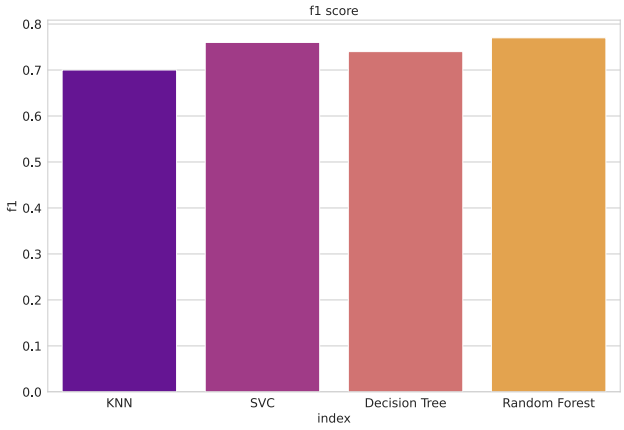


Figure 25: F1 scores.

References

[1] N. V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (June 2002), pp. 321–357. ISSN: 1076-9757. DOI: 10.1613/jair.953. arXiv: 1106.1813 [cs].

[2] *LOF: Identifying Density-Based Local Outliers: ACM SIGMOD Record: Vol 29, No 2.* https://dl.acm.org/doi/10.1145/335191.335388.