Mathematics in Machine Learning

Default of credit card clients dataset analysis

Luca Campana
s290085

Professors: Francesco Vaccarino, Mauro Gasparini

Academic Year: 2021/2022

Contents

1 Introduction

In early 2000s, a discrete number of Taiwanese banks were facing the situation of having low market shares. In order to overcome this, some of them began to over-issue credit cards: those were given to anyone requesting them, without making any sort of check on the efficiency of the applicants. As a result, most of card holders started to spend more money than what they could repay, thus accumulating heavy debts with the banks. The whole situation led to a deep economic crisis, which took a lot of time and effort to be overtaken.

To prevent the repetition of such a situation, it is useful to perform some risk prediction on card applications. By exploiting the previously collected financial information, it is possible to predict customers' levels of risk, and therefore to avoid uncertainty on their willing to repay debts. Here, in fact, the risk is represented by the probability of a payment default, that is to say the likelihood that a certain customer has to delay or miss the repayment of his accumulated debt.

For the exposed reasons, the purpose of this analysis is to explore the involved data, in order to find machine learning algorithms that could extract insights from them and perform the final binary classification, and so to distinguish potentially defaulting applicants from fulfilling ones. This report will describe the process in all its phases, as summarized below:

1. Exploration of the addressed dataset: the Default of Credit Card Clients Data Set (fields explanation, visualization and correlation).

2. Data preprocessing: outlier detection and removal, normalization, dimensionality reduction (PCA), data balancing through oversampling (SMOTE).

3. Model choice and tuning: testing of various simple algorithms (Decision Tree, Random Forest, Linear SVM, Logistic Regressor).

4. Validation and final results.

2 Data Exploration

2.1 Dataset presentation and fields description

The assessed data is the Default of Credit Card Clients Data Set [Yeh09]. This dataset contains information on credit card owner's characteristics and their payment history in Taiwan from April 2005 to September 2005.

Specifically, the set is composed of 30 000 rows, each of them referring to a distinct credit card holder. Along with the target variable, denoted as **default payment next month** (value: 1 for defaulters, 0 otherwise), there are 23 explanatory variables, both numerical and categorical, described below.

1. Numerical features:

    - **LIMIT_BAL** (NT dollars): amount of credit given by the bank on the card
    - **AGE** (years)

- **BILL_AMT1 - BILL_AMT6** (NT dollars): amount of bill statement, for the months from April to September, in reverse order (1 refers to September, 2 to August, and so on)
- **PAY_AMT1 - PAY_AMT6** (NT dollar): amount of monthly payments, for the months from April to September, in reverse order

2. Categorical features:

- **SEX** (1=male, 2=female, but soon scaled to (0, 1) for binarization)
- **EDUCATION** (1=graduate school, 2=university, 3=high school, 4=others, (0, 5, 6)=unknown)
- **MARRIAGE** (0=unknown, 1=married, 2=single, 3=others)
- **PAY_1 - PAY_6**: chronology of past payment states, for the months from April to September, in reverse order (-1=pay duly (no payment delays), 1=one month of delay, ... , 9=9 or more months of delay, (-2, 0)=unknown)

The dataset has no missing values in any of the fields, so there is no need to perform any imputation method. Instead, there are 35 duplicate rows, that have to be removed in order to prevent overfitting. The next phase consists in the handling of unknown values among the categorical variables. In fact, as mentioned in the description above, some of those fields have values that are not explained in the official documentation, so it is needed to figure out a method not to include those values in the analysis. This has been accomplished following two strategies:

- for the columns 'EDUCATION' and 'MARRIAGE', unknown values affect respectively 345 and 54 rows, so it is possible to directly drop the rows without affecting too much the data cardinality;
- for the columns 'PAY_i', unknown values always affect not less than 14 000 samples, so it is impossible to drop the rows; the only way to proceed is to consider them as pay duly, i.e. mapping all values in (-2, 0) to -1.

At the end of this whole process, the dataset consists of 29 566 rows.

2.2 Data distribution and statistics

2.2.1 Target distribution

To begin with, one of the most important explorations to be made is the one regarding the distribution of target values. Most of ML classifiers heavily suffer from 'unbalanced' data, that is a dataset in which a certain target value is more frequent than any other, since they tend to skew their predictions towards majority classes. So, we must acknowledge if this is the case, in order to set up proper strategies and overcome this drawback.

As the Fig.1 clearly shows, the dataset here is strongly unbalanced, since 77.68% of the samples are marked as non-defaulters. As a consequence, we will need some rebalancing methods, such as undersampling or oversampling.

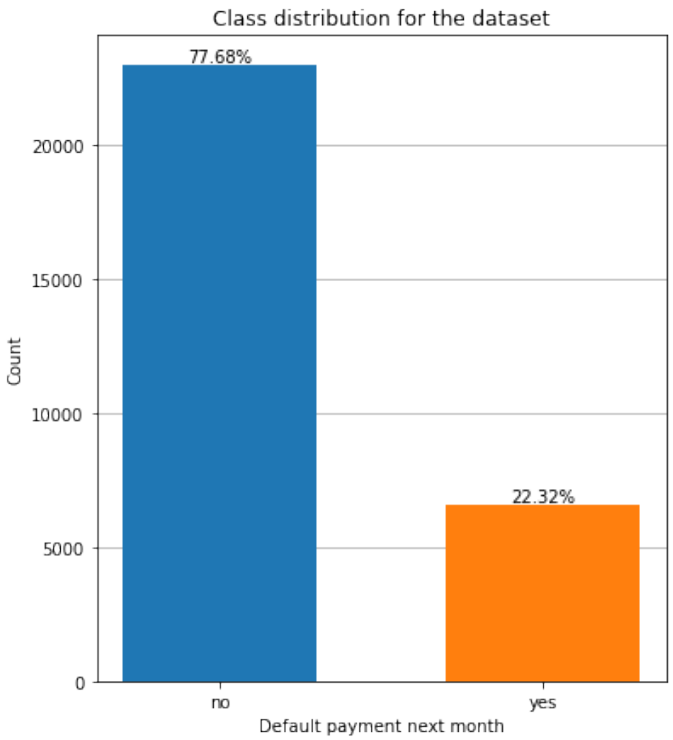

Figure 1: Number of samples for each class.

2.2.2 Fields domains and distributions

This section describes another significant exploration over data. Here, the analysis regards the fields compositions: for each one of them, we generate some statistical plots, over whom we try and make some assumptions. In particular, for numerical variables (Fig.2) the assessed plots are the Probability Density Function and the Box Plot, while for categorical ones (Fig.3) the choice has fallen over PDF and non-normalized bar chart. All plots are drawn separately for each target class, in order to allow the comparison among the two.

The depicted plots allow making some important assumptions. First of all, it is notable that, in none of the cases, the distribution of samples for the two classes is so similar to make us consider that field unuseful for the analysis. In fact, there are cases in which the distributions are clearly different, such as for the fields 'LIMIT_BAL' or 'SEX', but even for fields with similar PDFs, as 'PAY_i' or 'BILL_AMTi', there is a considerable difference in peaks, that could embody some kind of hidden significance. As a consequence, it is not a good choice to drop any of the columns at this stage. Moreover, the Box Plot analysis shows that there is plenty of outliers among the samples, so a strong outlier detection and removal phase will be required in order to obtain a good final quality (Section 3.1).

At this point, before moving on, we have to perform a little transformation. It is a common procedure, in case of categorical fields for which the integer values do not reflect any naturally ordered relationship between each other, to transform them in a manner that prevents the model to learn this misleading information. For this reason, it is useful to apply One-Hot Encoding to the fields 'EDUCATION' and 'MARRIAGE': this introduces a new field for each of the categories' values, so a total of 7 columns that substitute the old 2.

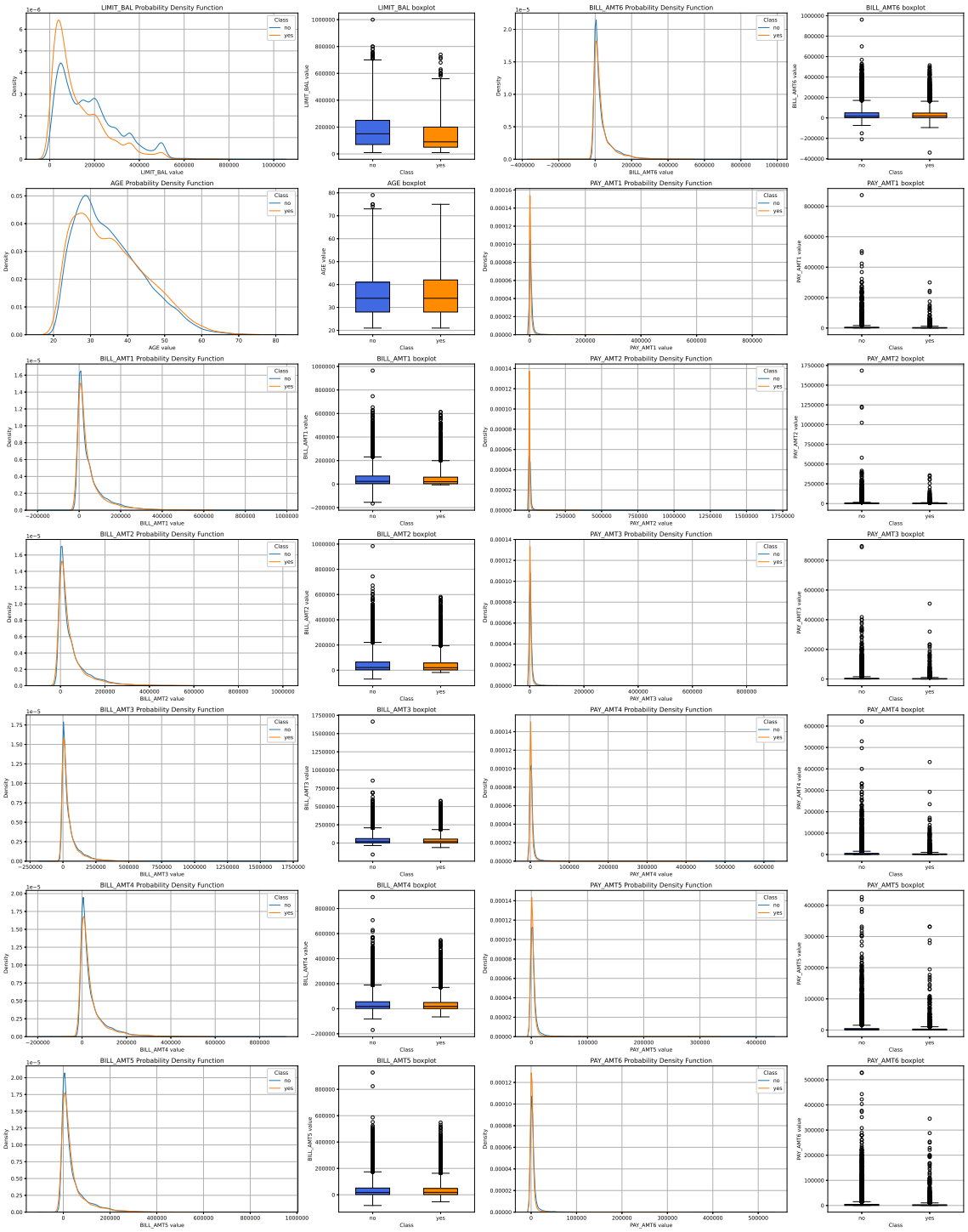

Figure 2: PDF and Box Plot for each of the numerical fields, separately per target value.[1]

2.3 Feature Correlation

The feature correlation analysis is a mandatory step, useful to investigate the features reciprocal dependencies. This can often help to identify a proper strategy to infer missing values among the data, but this is not the case, since there are no missing values. In our case, it is used to assess if there are groups of highly-correlated

[1]N.B.: all the graphs have been exported as vector graphics, so it is possible to zoom in as much as one wants.

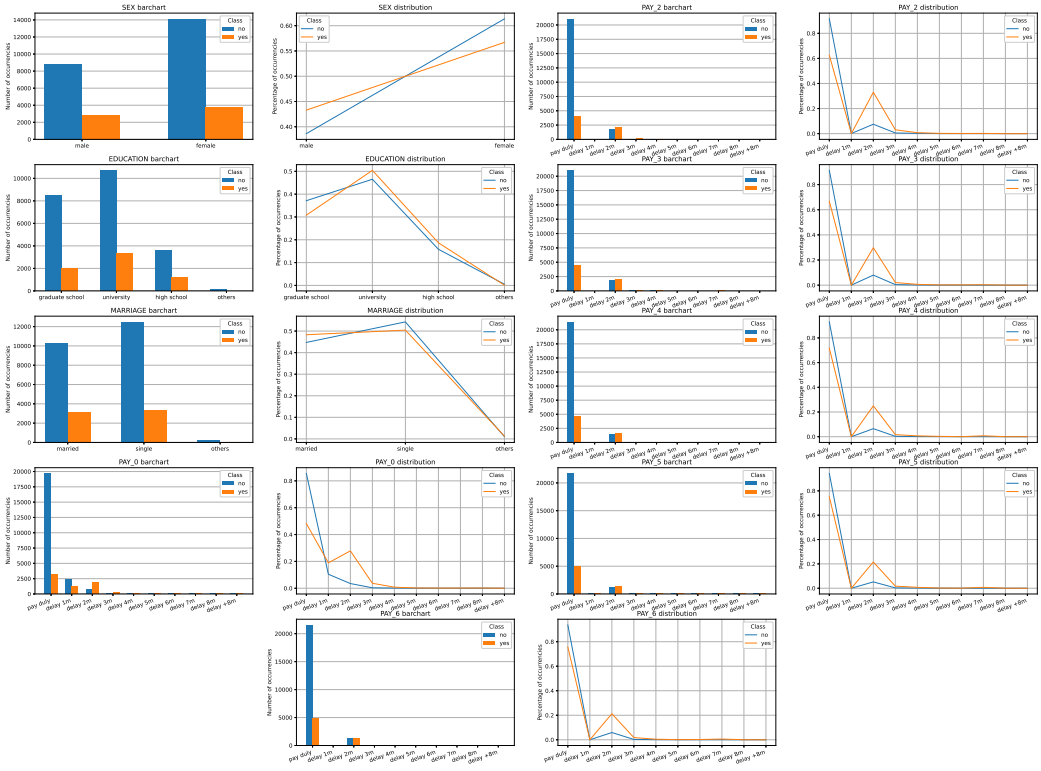

Figure 3: PDF and bar chart for each of the categorical fields, separately per target value.[1]

features, such that analyzing all of them instead of only one would not add a sufficient variance increase to boost the model performances. This is the phenomenon called "multicollinearity": two features are perfectly multicollinear if, from one of them, we can exactly predict the other, It is important to clarify that multicollinearity negatively affects only a certain specie of models, since others automatically detect and handle this type of characteristic.

To investigate linear dependency, that is the most common type of collinearity, it is useful to measure the *Pearson Correlation coefficient*: given the two features X and Y, their correlation measured over the sample space Ω is:

$$\rho_{XY} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\bar{X})(Y-\bar{Y})]}{\sigma_X \sigma_Y} = \frac{1}{\sigma_X \sigma_Y (n-1)} \sum_{i \in \Omega} (x_i - \bar{x})(y_i - \bar{y}) \quad (1)$$

The coefficient range is (-1, 1): the closest it is to its boundaries, the strongest is the correlation among the features. If we apply this formula for each pair of features, we obtain the correlation heat map, represented in Fig.4.

There is plenty of things to notice here. Firstly, we have to highlight the high negative correlation between the first two fields of each One Hot encoded feature: these correspond to the most populated categories, and the rationale between this fact is easily understandable. Seen this, it is safe to remove the fields 'MARRIAGE_1' and 'EDUCATION_1'. Moreover, logically there exists a weak relationship also between the marital state and the age of the subject.

Other correlated groups are the ones regarding the same measurement among different months. This is particularly evident for the fields 'BILL_AMTi': obviously,

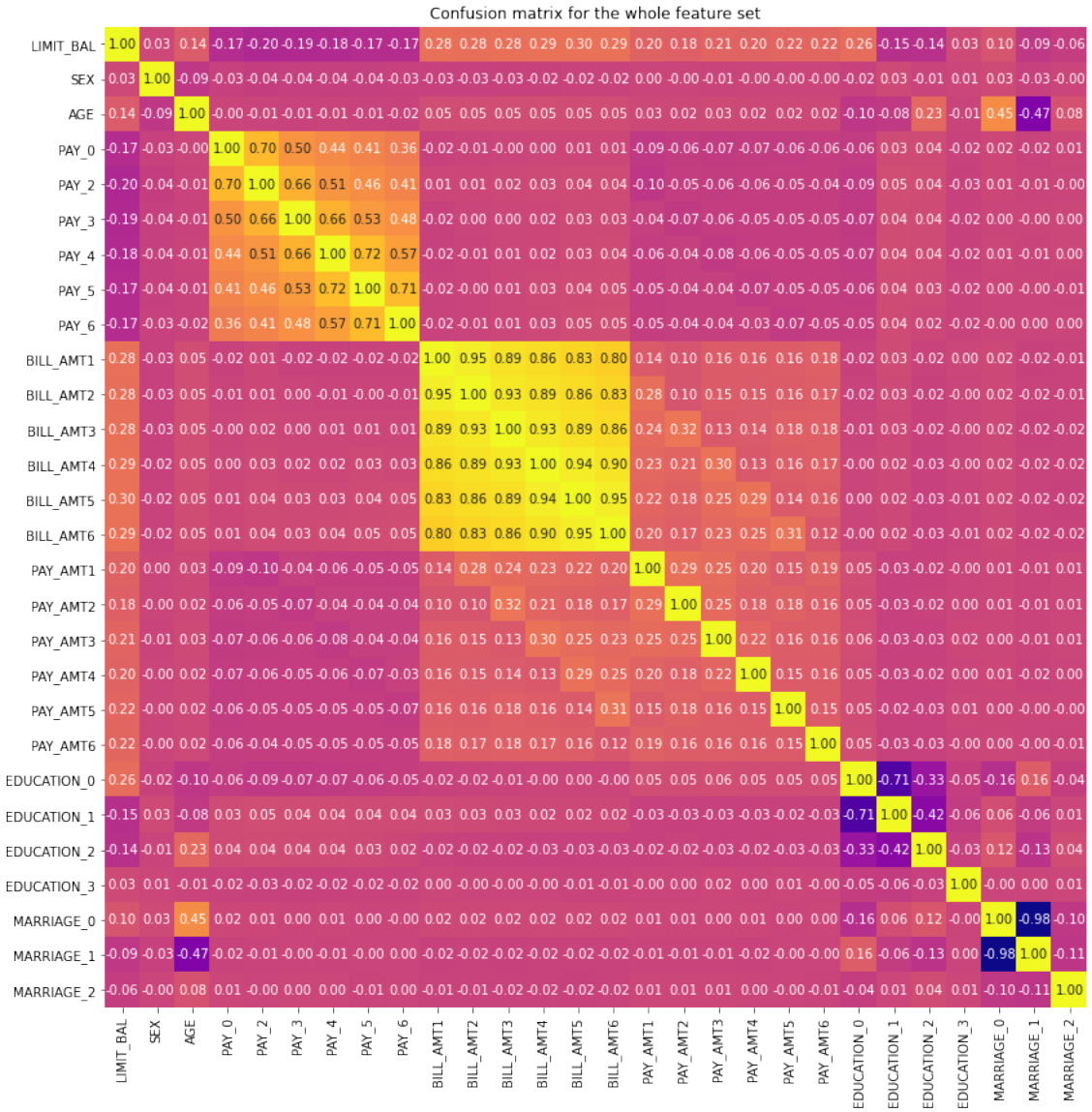

Figure 4: Pearson correlation heat map.

one's bill statement amount in a certain month strongly depends from previous amounts. As a consequence, it might be a good strategy to maintain only the field 'BILL_AMT1', since it makes more sense for the analysis to consider the most recent bill statement amount.

3 Data Preprocessing

3.1 Outliers Management

Outliers are defined as 'wrong' data, that can badly drive the algorithm outcomes. They are mostly generated by human errors, or by extremely noisy measurements. For categorical fields, a quick and simple strategy to detect them, since they are usually few and distant from the distribution 'core', is the so called Box Plot (Fig.5).

Box Plots are a smart way to visualize distributions, basing on five focal values: the distribution median (Q2), the 1^{st} quartile (Q1), the 3^{rd} quartile (Q3), and the whiskers, placed on a distance from the quartiles that is proportional to Inter-

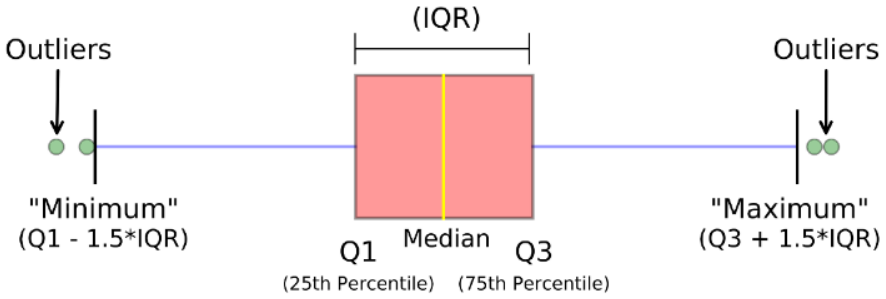

Figure 5: Box Plot in all its parts. Source: [Gal18]

Quartile Range (IQR). All the points that fall outside the whiskers are considered as outliers, since their value is too isolated from the major distribution.

In our case (Fig.2), the whiskers are drawn at $(Q1 - 2.5 * IQR, Q3 + 2.5 * IQR)$: still, there are a lot of values falling outside. However, since the data has few features, removing all samples related to these points will not cause curse of dimensionality and the final performance will benefit from this. After the outliers removal, the cardinality of the dataset drops to 22 462 rows.

3.2 Data Normalization

Normalization is a widely diffused technique in data analysis tasks. By this, we prevent the algorithms to infer wrong pieces of information coming from differently scaled fields. One of the most diffused normalization techniques is Standard Scaling: for each field, for each sample i we compute its new value as $\hat{x_i} = \frac{1}{\sqrt{v}}(x_i - \bar{x})$, where $\bar{x}$ and v are respectively the sample mean and the sample variance, estimated over the sample space. By this transformation, applied over each non-binary field, data have a new distribution with $\mu = 0$ and $\sigma = 1$.

3.3 Dimensionality Reduction (PCA)

Often, when we deal with really high-dimensional data, it is a good practice to try and reduce the number of explanatory variables, in order to obtain a thinner representation, that would be easier to compute and be fed into models. A common technique used for this type of task is PCA (Principal Component Analysis), and it works in the following way: given a sample $x \in \mathbf{R}^n$, we want to know the compression matrix $W \in \mathbf{R}^{d,n}$ and the recovering matrix $U \in \mathbf{R}^{n,d}$ such that $\tilde{x} = Wx \in \mathbf{R}^d$ is the new, lower dimensional representation of x. The way to accomplish this is to minimize the difference, in terms of Euclidean norm, between the original sample and its recovered version. As a consequence, if applied over the whole feature set, the entire transformation turns into a minimization problem:

$$W, U = \underset{W,U}{\operatorname{argmin}} \sum_{i \in \Omega} \|x_i - UWx_i\|_2^2 \tag{2}$$

This would be a quite costly computation, so some simplifications are needed. Firstly, it can be proven that all solutions of (2) are in the form $W = U^{\mathsf{T}}$, so the

objective function can be rewritten as this:

$$\begin{aligned}\left\|x - UU^{\mathsf{T}}x\right\|_2^2 &= \|x\|_2^2 - 2x^{\mathsf{T}}UU^{\mathsf{T}}x + x^{\mathsf{T}}UU^{\mathsf{T}}UU^{\mathsf{T}}x \\ &= \|x\|_2^2 - x^{\mathsf{T}}UU^{\mathsf{T}}x \\ &= \|x\|_2^2 - \text{trace}(U^{\mathsf{T}}xx^{\mathsf{T}}U)\end{aligned} \tag{3}$$

Since the trace is a linear operator, the minimization becomes much more lightweight:

$$U = \underset{U^{\mathsf{T}}U=I;U\in\mathbf{R}^{n,d}}{\text{argmax}} \text{trace}\left(U^{\mathsf{T}}\sum_{i\in\Omega} x_i x_i^{\mathsf{T}}U\right) = \underset{U^{\mathsf{T}}U=I;U\in\mathbf{R}^{n,d}}{\text{argmax}} \text{trace}\left(U^{\mathsf{T}}AU\right) \tag{4}$$

Seen this form of the problem, it is provable that the desired U matrix has for columns the eigenvectors related to the n leading eigenvalues of the matrix $A = \sum_{i\in\Omega} x_i x_i^{\mathsf{T}}$. So, since U columns are all orthogonal to each other, so will be the fields of the new coordinate system, i.e. the *Principal Components*. This allows to explain the original data variance in a lower dimensional way: the 1^{st} component, i.e. the *leading eigenvector* of A, encodes most of the data variance, while every successive one goes in the direction of largest variance in the residual subspace. By this way, it is possible to find the most suited tradeoff between explained variance and number of features, as shown in Fig.6.

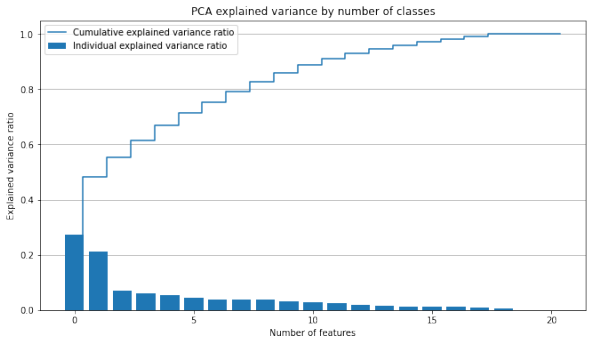

Figure 6: Individual and cumulative explained variance ratio, for each of the PCs.

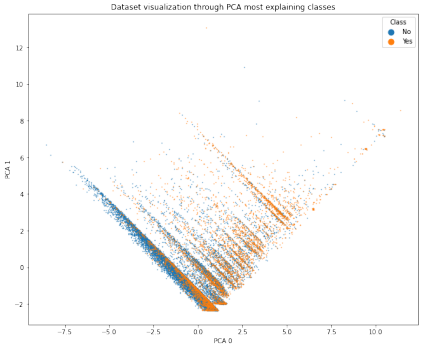

Figure 7: Dataset visualization through the first two PCs.

For our task, the dimensionality reduction is not a useful transformation, since the number of initial features is so small that it would not be a problem for any of the algorithms. Still, PCA can be useful to get a 2D dataset visualization (Fig.7). As can be seen, it is not so easy to detect a clear linear separation between the two classes: this means that the problem is not trivial, and so it would be a possible solution strategy.

3.4 Data Rebalancing: Oversampling (SMOTE)

Having a heavily unbalanced dataset results in damaging the quality of learning algorithms, since the majority of the models tend to be more influenced by most present class. This causes *biased predictions*: the model will classify by following the distribution of the dataset over whom it has been trained, so it will be more likely to output the majority class. However, there exist several ways to overcome this phenomenon:

- undersampling the majority class (but this reduces the dataset cardinality);

- oversampling with replacement the minority class (this is often unuseful, for the reason that it makes more specific, i.e. narrower, the decision region for the minority class, favouring the possibilities of overfitting);
- Synthetic Minority Oversampling Technique (SMOTE) [CBHK02].

This last technique have shown to be fairly better than others. Through SMOTE, we generate synthetic samples for the minority class. Practically, this is done by taking each real sample, drawing a line between it and its k nearest neighbors, and introducing samples on random points along these segments (Fig.8). Algebraically, taken the real sample x, one of its neighbors x_i, and the parameter λ chosen uniformly in $[0, 1]$, the synthetic sample gets calculated as $\tilde{x} = \lambda(x_i - x) + x$.

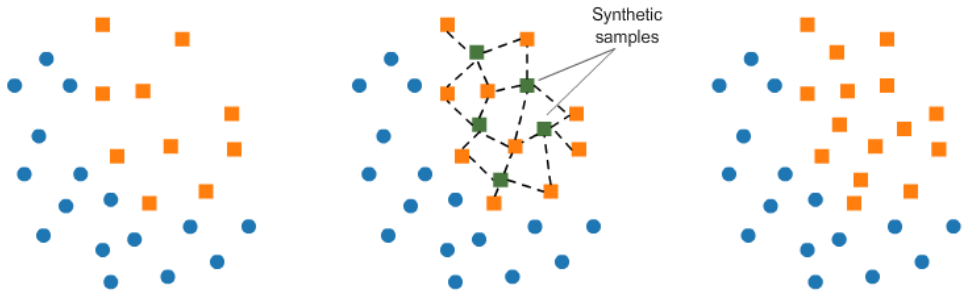

Figure 8: Visualization of SMOTE. Source: [Das19]

By this technique, we are creating a larger and less specific decision region for the minority class, thus enhancing the model generalization power. Of course, it is wrong to have synthetic samples in the test set, so SMOTE has to be performed after the dataset split, that is after k-fold (Section 4.2).

4 Methodology

4.1 Metrics

Metrics are, together with loss measures, the only instruments available to inspect the overall quality of a trained model. Through metrics, we are interested to inspect how the model behaves when it gets fed with never seen data, so we evaluate them over the test set. For classification tasks, most of metrics are built by comparing the actual classes of data versus the model predictions. For binary classifiers, metrics are calculated basing of the quantities of *true positives* (TP), *true negatives* (TN). *false positives* (FP), *false negatives* (FN).

In particular, for the described project, the analyzed metrics have been:

- **Accuracy**, that rates how many right predictions the model got above the total, defined as $\frac{TP+TN}{TP+TN+FP+FN}$. This metric can be misleading for imbalanced data sets, showing high scores for the fact that true positives are higher than every other class.
- **F1-score**, that rates if a model has good precision and recall, namely if the model is good to recognize as positives all and only the real positive samples. This is defined as $\frac{2TP}{2TP+FP+FN}$, and it is a measure that is less influenced by the dataset balancing.

Both the assessed metrics have values in the $[0, 1]$ range, and the closest they are to 1, the better is the algorithm they rate.

4.2 Cross Validation

To evaluate model performances in a consistent way, and to perform the tuning of hyperparameters, it is mandatory to conduct a phase of cross validation. In fact, testing models on already seen data is really dangerous, as it is the main cause of overfitting, but at the same time we do not want to waste some useful training data.

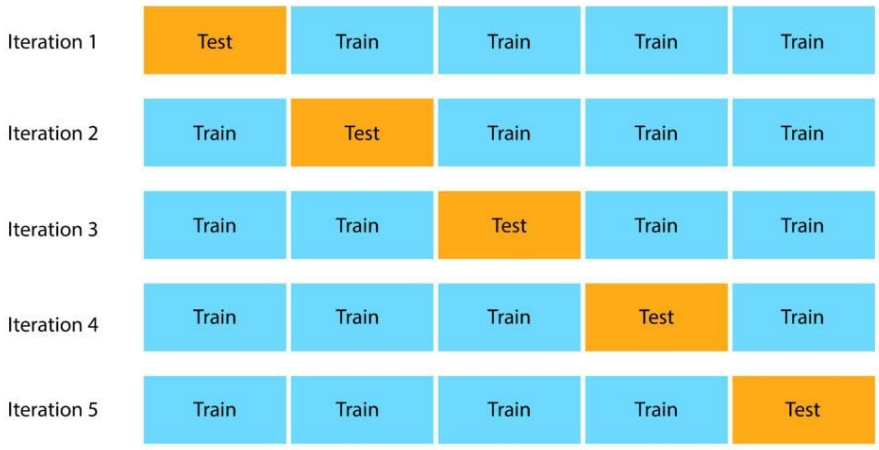

Figure 9: Visualization of k-fold Cross Validation (with $k = 5$). Source: [Aff22]

The solution to this issue is represented by k-fold cross validation (Fig.9). The dataset gets divided into k folds: of these folds, 1 will be selected as the test set, and the other $(k-1)$ will compose the train set. This procedure gets then repeated for a total of k times, such that at the end each sample has been treated $(k-1)$ times as a train sample, and once as a test sample. Naturally, all the runs are independent from each other, so the system is never evaluated over train data, and moreover there is no data wasting. The metrics are computed on every run, and at the end we take the average: by this way, we obtain ratings that are not influenced by the particular sets, but on the contrary can be considered as fairly general.

For this project, k-fold has been performed over 10 total folds. In addition, the splits have been done in a stratified fashion, in order to keep in each fold the same distribution of the entire dataset. This is useful to boost model robustness and therefore get a better generalization.

5 Algorithm Choice

As widely explained in the previous sections, the task addressed by this project is a binary classification, that is a subclass of supervised learning problems. To solve problems of this kind, there exist a wide variety of models, which differ one from another by some specific characteristics. In this section, the used models will be presented, together with the reason behind their selection and their final performances.

5.1 Decision Tree

Decision Trees are a supervised learning method, based on finding particular splitting rules (built up on features' values) that allow to correctly separate target classes. In other words, the splitting rules subdivide the sample space into *non-overlapping hyper-rectangular* regions, that are theoretically homogeneous in terms of target class, in order to place each unknown sample into one of these and produce a prediction according to it. This results in a model that is easy to interpret, whose construction is reasonably slow, but that is extremely fast to classify a new sample.

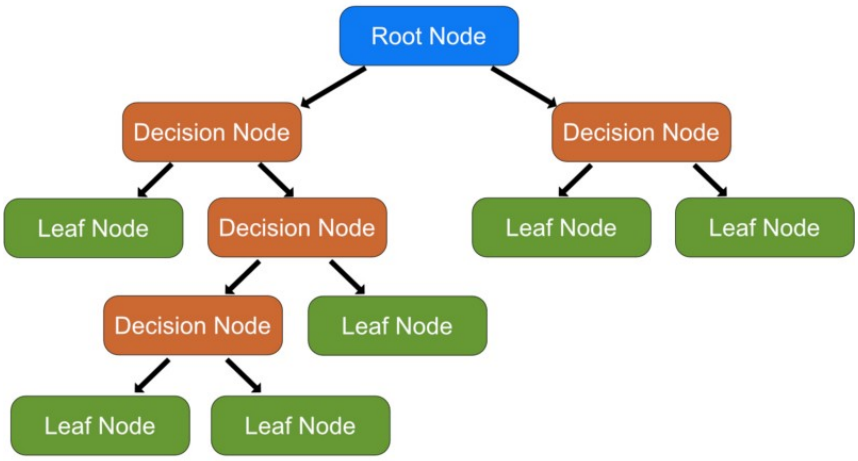

Figure 10: Visualization of a Decision Tree. Source: [Rad21]

Unfortunately, finding the best splits is often computationally unfeasible, so the most common strategy is to follow a greedy approach, in which at each step we select the local best attribute to split on. This is done by addressing the quality of local split, evaluated by comparing the father node and the potential children in terms of their *impurity* (i.e. target class inhomogeneity). There are two indices to rate impurity:

- **GINI index:** $\text{GINI}(t) = 1 - \sum_{j \in \{0,1\}} \mathbf{P}(j|t)^2$
- **Entropy:** $\text{Entropy}(t) = -\sum_{j \in \{0,1\}} \mathbf{P}(j|t) \log_2 \mathbf{P}(j|t)$

For both formulas, t is the addressed node, while j represents all possible target values.

As it is described, Decision Tree is a learning method that has a high risk to overfit on test data. For this reason, it is useful to design some early stopping criteria to avoid bad behaviours. For example, the tree building could end if the number of instances on a certain node goes below a given threshold; if there are no improving splits left; or if the class labels are independent from the available features at a certain stage (verifiable by a χ^2 statistical test). Moreover, we can state a-priori a maximum depth for the tree. However, since in this case we have a limited number of features, early stopping does not introduce any positive effect. This can be seen from the hyperparameter tuning phase.

In fact, we evaluated all combinations of the following parameters:

- **criterion:** ['gini', 'entropy']

- **max_depth:** [None, 10]

and the best configuration was found for ('entropy', None). Seen this, a full training phase has been carried out (10-fold Cross Validation, explained in section 4.2), and the mean values found for accuracy and f1 were **(0.677, 0.424)**. Also, we choose to display the confusion matrix of the most valuable fold (Fig.11), related to accuracy and f1 equal to (0.691, 0.446).

5.2 Random Forest

Random Forest is an ensemble method, that increases the performances of simple Decision Trees by the principle whereby "a class is always smarter than its smartest student". So, the Random Forest is not other than growing a set of DTs, train them separately and then judge according to the majority of single prediction. In addition, to lower the variance and boost the robustness of models, Random Forest is built through *Bagging*, a technique by which each single DT gets trained with a random subset of the training set, over a random subset of dimensions. The obtained trees are far from being overfitted, so the ensemble prediction can be more qualitative.

Also in this case, we performed hyperparameters tuning over the following:

- **n_estimators:** [100, 200]
- **criterion:** ['gini', 'entropy']
- **max_depth:** [None, 10]

and the best configuration was found for (200, 'entropy', None). Seen this, a full training phase has been carried out (10-fold Cross Validation, explained in section 4.2), and the mean values found for accuracy and f1 were **(0.776, 0.532)**. Also, we choose to display the confusion matrix of the most valuable fold (Fig.12), related to accuracy and f1 equal to (0.791, 0.562).

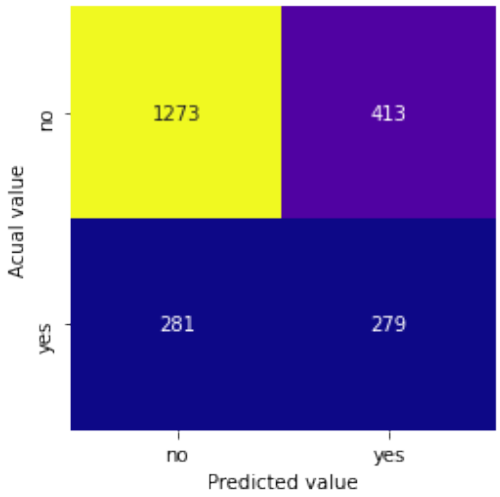

Figure 11: Decision Tree best fold Confusion Matrix.

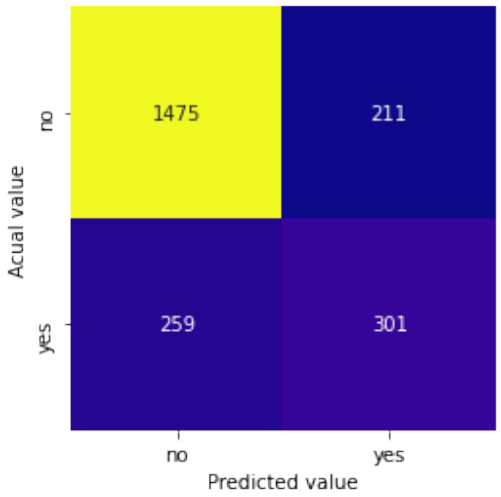

Figure 12: Random Forest best fold Confusion Matrix.

5.3 Support Vector Machine

Support Vector Machines are widely considered among the best performing ML algorithms for classification, and particularly for the binary one. However, their functioning is not so straightforward.

Consider all the samples, that stay on a n-dimensional space (n is the input dimensionality). With SVM we try to find the *best possible linear separator* among the classes: this is not other than a hyperplane, described by a function of the input variable $f(x)$, designed in a way that maximizes the *margin*, i.e. the (equal) distance between the boundary and the nearest point of both classes, the *Support Vectors* (Fig.13). This principle allows to reduce misclassifying possibilities. So, given the weight vector w and the bias term b, the separator is in the form $f(x) = w^\mathsf{T} x + b$. Now, we want to identify regions basing on $f(x)$: in particular, we want $f(x) \leq -1$ for one class, $f(x) \geq 1$ for the other, and $-1 < f(x) < 1$ for the inter-margin region. To do this, called (x_-, x_+) the support vectors, it has to be $f(x_-) = -1$ and $f(x_+) = 1$: this allows to put in relationship margin and separator. In fact, since the separator must be the axis of the SV distance $(x_+ - x_-)$, we can express margin as:

$$\begin{aligned} m &= \frac{w^\mathsf{T}(x_+ - x_-)}{2\,\|w\|} = \frac{1}{2\,\|w\|}\left(w^\mathsf{T} x_+ - w^\mathsf{T} x_-\right) = \\ &= \frac{1}{2\,\|w\|}\left[\left(w^\mathsf{T} x_+ + b\right) - \left(w^\mathsf{T} x_- + b\right)\right] = \frac{1}{\|w\|} \end{aligned} \tag{5}$$

The above equation says that maximizing the margin is equivalent of minimizing $\|w\|$, or more easily $\|w\|^2$. So, finding the best decision boundary becomes a linear optimization problem:

$$w, b \;=\; \underset{w \in \mathbf{R}^n,\, b \in \mathbf{R}}{\operatorname{argmin}} \quad \|w\|^2 \quad \text{s.t.} \quad \forall i \in \Omega : y_i \left(w^\mathsf{T} x_i + b\right) \geq 1 \tag{6}$$

where $y_i \in \{-1, 1\} \forall i$. As every optimization problem, this can be simplified through Lagrangian relaxations.

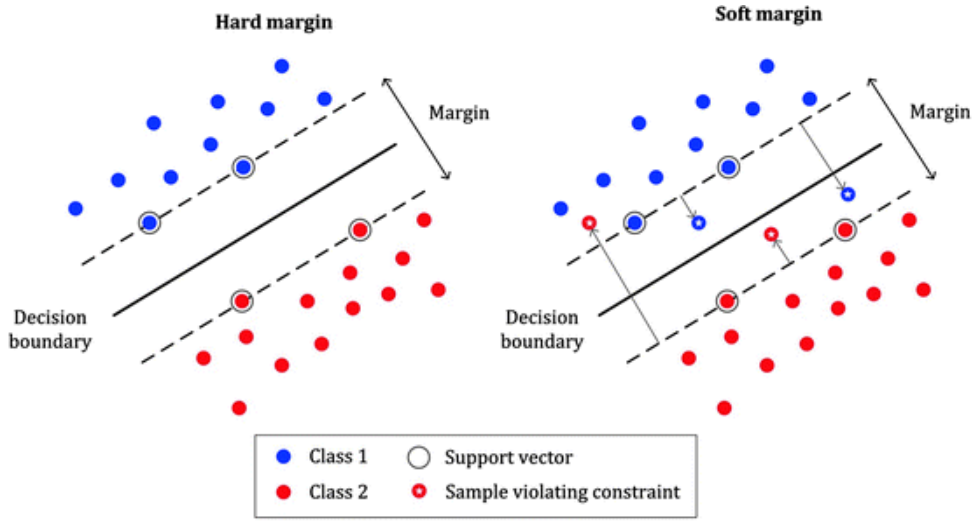

Figure 13: Visualization of a SVM, with hard and soft margin. Source: [MLM19]

Unfortunately, most of the problems are not linearly separable, so often a solution for (6) does not exist. To overcome this, there are two main solutions.

One of them concerns in making the margin "soft", that is allowing some points in the inter-margin region, but penalizing them (Fig.13). The problem then becomes:

$$w, b \quad = \underset{w \in \mathbf{R}^n,\, b \in \mathbf{R}}{\operatorname{argmin}} \quad \left(\frac{1}{2} \|w\|^2 + C \sum_{i \in \Omega} \xi_i \right) \quad \text{s.t.} \quad \forall i \in \Omega : \quad \begin{cases} y_i \left(w^\mathsf{T} x_i + b \right) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases} \tag{7}$$

where C is a regularization parameter, inversely proportional to the number of accepted mistakes.

With this last formulation, we performed hyperparameters tuning over the following quantities:

- **penalty:** ['l1', 'l2']
- **C:** [1, 2]

and the best configuration was found for ('l2', 1). Seen this, a full training phase has been carried out (10-fold Cross Validation, explained in section 4.2), and the mean values found for accuracy and f1 were **(0.765, 0.553)**. Also, we choose to display the confusion matrix of the most valuable fold (Fig.15), related to accuracy and f1 equal to (0.776, 0.576).

Another possibility, together with the soft margin, was *kernelizing* the scalar product of the objective function. This means that, with the aid of a kernel function, the data is mapped into a second feature space, in order to find linear separability. Also this method was tried, with Radial Basis Function (RBF) kernel [Sre20], but this was discarded as it did not show a clear improvement with respect to the linear formulation, although being much more time consuming.

5.4 Logistic Regression

Logistic Regression is one of the most used and earliest methods for classification, and in particular for the binary one. It is based on outputting a probability, i.e. a single value between 0 and 1, corresponding to the probability that the analyzed sample has its label equal to 1. This is then turned into a prediction by comparing it with a certain threshold, which is usually equal to 0,5.

From a practical point of view, the model builds a *sigmoid-like* function $h(z) = (1 - e^{-z})^{-1}$ (Fig.14), where $z = w^\mathsf{T} x + b$. In other words, $h(z) = P(Y = 1|X)$. So, in this case, learning the decision function means estimating the most suitable values for w and b. Denoting as p the regressor output, it can be written:

$$\frac{1}{1 + e^{-(w^\mathsf{T} x + b)}} = p \quad \Leftrightarrow \quad w^\mathsf{T} x + b = \log \frac{p}{1 - p} \tag{8}$$

This last quantity is the *log-odds* of the event $Y = 1$. As it can be noticed, to obtain a good model, we need the z quantity to be high for samples labelled positively, and low for negatively labelled ones. To accomplish this, a useful tool is the *maximum likelihood* method: the whole situation can be turned into a maximization problem, for which the objective function is the *likelihood function*:

$$\ell(w, b) = \prod_{i \in \Omega : y_i = 1} p(x_i) \prod_{i' \in \Omega : y_{i'} = -1} (1 - p(x_{i'})) \tag{9}$$

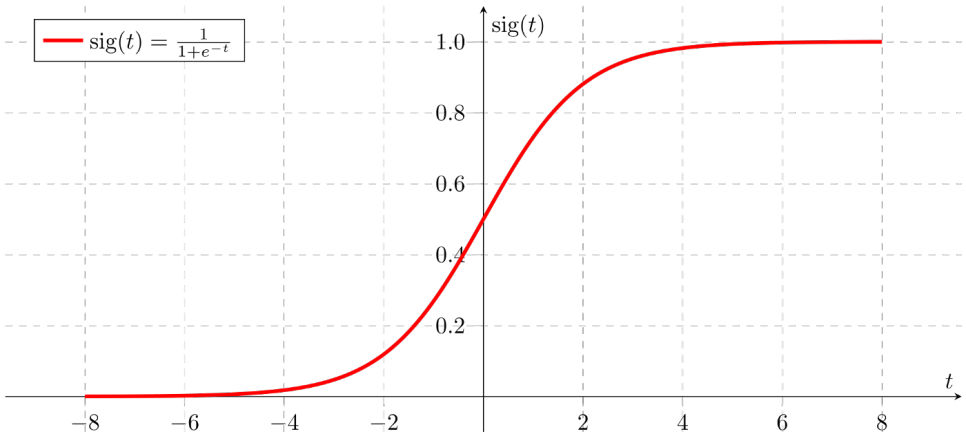

Figure 14: Plot of a sigmoid function. Source: [Arc18]

This also allows some kind of relaxation, that can be also exploited as a form of regularization. So, following the approach of the prevoius section, we performed hyperparameters tuning over the following quantities:

- **penalty:** ['l1', 'l2']
- **C:** [1, 2]

and the best configuration was found for ('l1', 2). Seen this, a full training phase has been carried out (10-fold Cross Validation, explained in section 4.2), and the mean values found for accuracy and f1 were **(0.761, 0.556)**. Also, we choose to display the confusion matrix of the most valuable fold (Fig.16), related to accuracy and f1 equal to (0.769, 0.576).

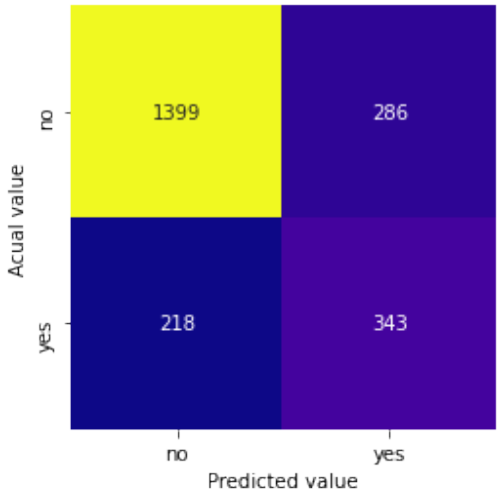

Figure 15: SVM best fold Confusion Matrix.

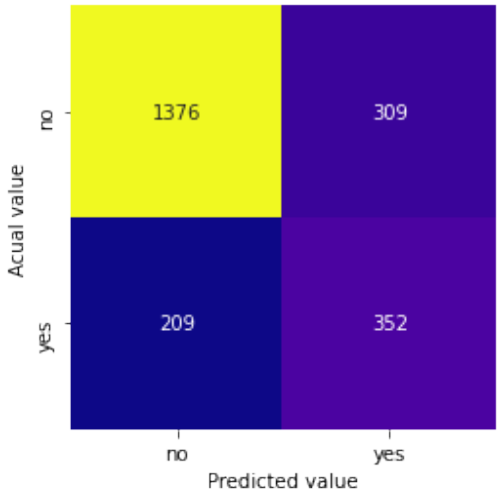

Figure 16: Linear Regression best fold Confusion Matrix.

6 Final Evaluations

Once evaluated the performances of each of the selected models, some final considerations can be done. Through the comparison of the final metrics scores (Fig. 17), a few important facts can be noticed.

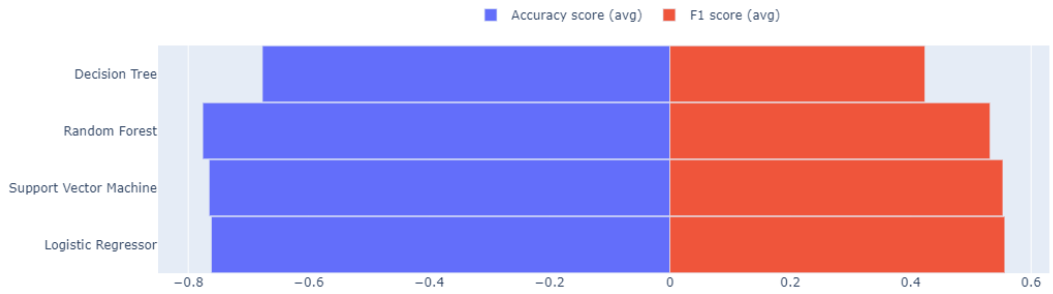

Figure 17: Final metric evaluation for each selected model.

To begin with, all the models show comparable performances, except for Decision Tree, which is by far the worst performing model. Random Forest, instead, shows the best accuracy, but it is slightly worse than SVM and Logistic Regression in terms of f1-score.

However, generally speaking, scores are diffusely not very high. This is due to the presence of a significant number of false positives and false negatives in every confusion matrix. One of the most likely causes is the unbalancing of the dataset: the performed oversampling cannot completely remove the inner data bias, but only mitigates it.

References

[Aff22] Iniabasi Affiah. How to Implement K fold Cross-Validation in Scikit-Learn. https://www.section.io/engineering-education/how-to-implement-k-fold-cross-validation/, 2022.

[Arc18] Arc. Derivative of the Sigmoid function. https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e, 2018.

[CBHK02] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[Das19] Ashesh Das. Oversampling to remove class imbalance using SMOTE. https://medium.com/@asheshdas.ds/oversampling-to-remove-class-imbalance-using-smote-94d5648e7d35, 2019.

[Gal18] Michael Galarnyk. Understanding Boxplots. https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51, 2018.

[MLM19] MLMath.io. Math behind SVM(Support Vector Machine). https://ankitnitjsr13.medium.com/math-behind-svm-support-vector-machine-864e58977fdb, 2019.

[Rad21] Dario Radečić. Master Machine Learning: Decision Trees From Scratch With Python. https://betterdatascience.com/mml-decision-trees/, 2021.

[Sre20] Sushanth Sreenivasa. Radial Basis Function (RBF) Kernel: The Go-To Kernel. https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a, 2020.

[Yeh09] I-Cheng Yeh. Default of credit card clients Data Set . https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients, 2009.