POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Mathematics in Machine Learning

Default of credit card clients

Professors

Prof. Francesco VACCARINO

Prof. Mauro GASPARINI

Candidate

MATTEO MERLO

September, 2021

Contents

1 Introduction

Credit card default happens when you have become severely delinquent on your credit card payments. For example, the U.S. federal government allows student debt to be delinquent for 270 days before declaring it to be in default. It is a delicate status because it will affect the relationship among the client and the card issuer but more important, it will decrease the probability to get approved for future credit-based services. The goal of this analysis is to predict if a client will go in default in the following month by using the *Default of Credit Card Clients Data Set* as a reference [1].

2 Exploratory Data Analysis

Default of Credit Card Clients Data Set contains 30 000 instances of credit card status collected in Taiwan from April 2005 to September 2005. These information were collected because Taiwan's card issuers expected to reach a *card debt prices* pick in the third quarter of 2006 [2]. In particular, for each record (namely, each client) we have demographic information, credit data, history of payments and bill statements. To be more precise, the following is the complete list of all the 23 predictors that we have to our disposal:

- Categorical predictors
  - **sex** : 1 = male, 2 = female
  - **education** : 1 = graduate school, 2 = university, 3 = high school, 4 = others, 5 = 6 = unknown
  - **marriage** : 1 = married, 2 = single and 3 = others
  - **pay_[1,...,6]** : (past payments from September to April 2005 [1]) -1 = pay duly, 1 = payment delay for one month, ... , 9 = payment delay for nine months and above
- Numerical predictors
  - **id** : ID of each client
  - **limit_bal** : amount of given credit in NT dollars
  - **age** : age in years
  - **bill_amt[1,...,6]** : bill statements from September to April 2005
  - **pay_amt[1,...,6]** : previous payments from September to April 2005

Finally the target is **default.payment.next.month** and it can assume values 0 and 1:

- 1 : the client will go in default
- 0 : the client will not go in default

[1] They are reported in reverse order : September 2005, August 2005, ... , April 2005

2.1 Data distribution

2.1.1 Target

It is possible to see from Figure 1 the distribution of the target variable *default payment next month*. It clearly shows an imbalance towards the 0 class (i.e. no default), with around 78% of the whole dataset.

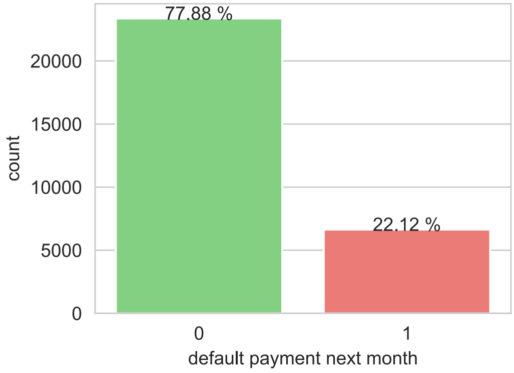

Figure 1: Countplot of *default payment next month*

2.1.2 Categorical predictors

By inspecting the cateogorical predictors *sex, education* and *marriage* in Figure 2 it is possible to state that there there are much more female than males on the dataset, and in particular males have a slightly higher chance to go in default rather than females (0.24% vs 0.21%). Then, the more representative education categories are *University*, followed by *Graduate School* and *High School* with the probability of going in default of 0.19%, 0.24%, 0.25%, respectively. Thanks to this representation we have also found two misrepresented categories $(5, 6)$, both represented as "unknown" and we grouped them together with "others". Finally, the last bar chart is devoted to the marriage status, thanks to which it is possible to see that we have slightly more *single* credit card owners, but also that the probability of married people is higher than the probability for single people (maybe because married people tend to have more expenditures for children, house and so on).

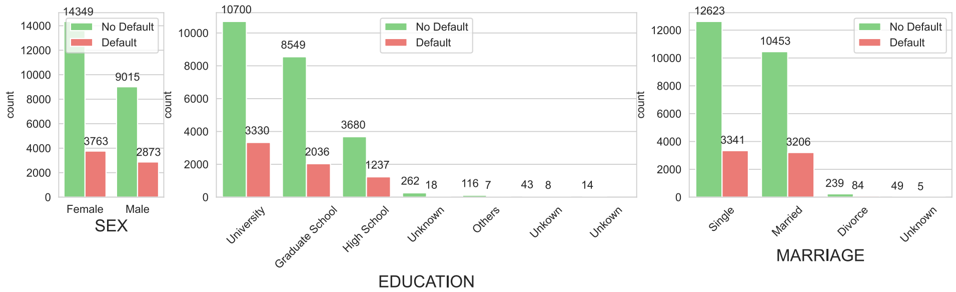

Figure 2: Countplot of *sex, education* and *marriage* grouped by class

The good news is that most of these credit card owners are duly paying their credit bills. In fact, in Figure 3 it is possible to see that most of them paid their debts within the pre fixed deadlines ($PAY\_X \leq 0$). As a direct consequence, the probability of default increase as the number of delayed payments increase.

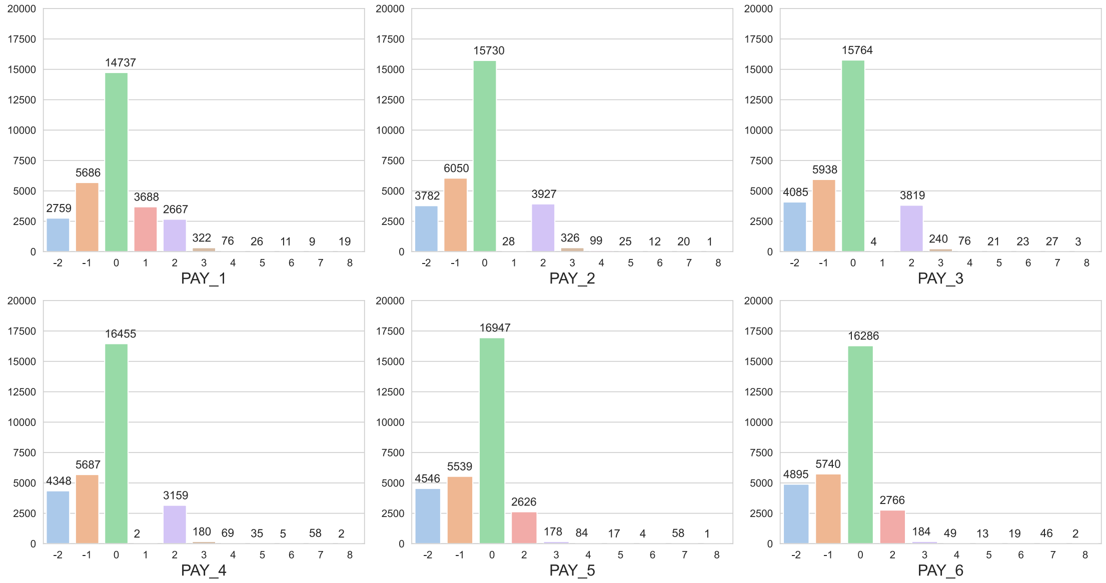

Figure 3: Countplot of *PAY_1, ... , PAY_6* grouped by class

2.1.3 Continuous predictors

In statistics, the Kernel Density Estimation (KDE) is a fairly well known technique for estimating the probability density function in a non parametric way (i.e. it does not assume any underlying distribution). So, for the following continous attribute we explored their KDE plots.

It is possible to observe from Figure 4 that most of the default come from credits with a lower $LIMIT\_BAL$ (i.e. credit amount), in particular they are observed in a range among a few thousands NT dollars to around $\$150,000$. Above this threshold the customers are more likely to repay their debts.

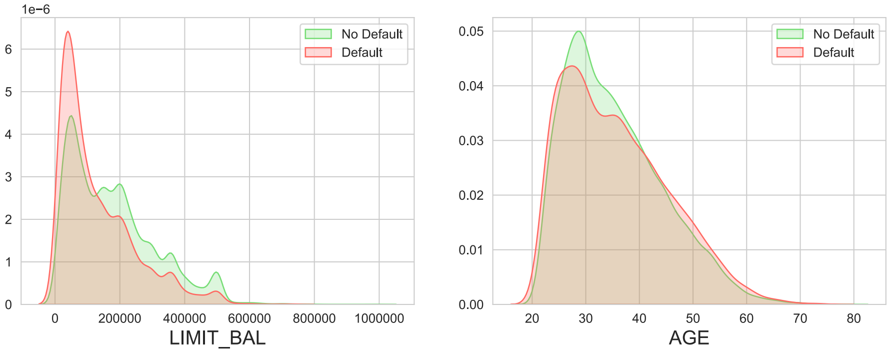

Figure 4: KDE plot of $LIMIT\_BAL$ and AGE grouped by class

Moreover, another interesting observation can be done by looking at $BILL\_AMTX$ in Figure 5. In fact, as one may think, people with a higher bill amount have an higher chance to go in default.

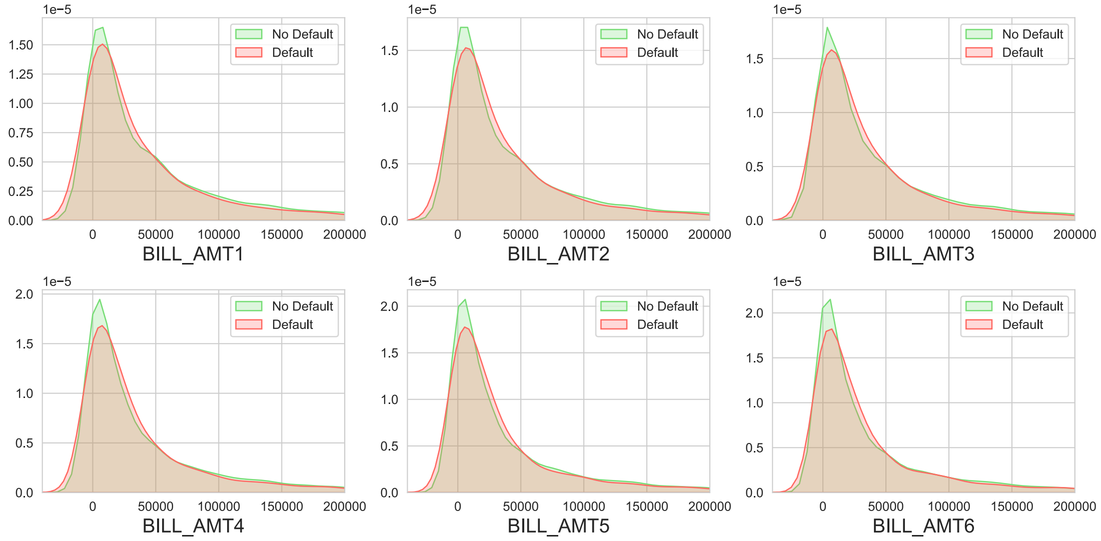

Figure 5: KDE plot of $BILL\_AMTX$

Finally, in Figure 6 there are reported the distributions of PAY_AMTX, from which it is possible to observe that there is an higher chance of default for those that did not paid their depth (so, near to 0).

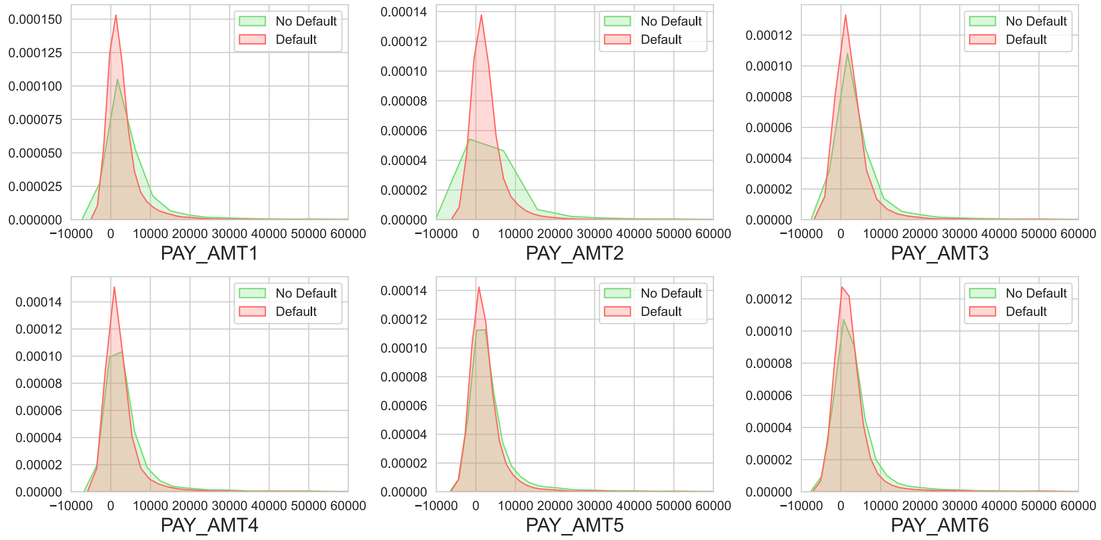

Figure 6: KDE plot of $LIMIT\_BAL$ and AGE

2.2 Correlation

Correlation is a statistical term describing the degree to which two random variables move in coordination with one-another. Intuitively, if they move in the same direction, then those variables are said to have a positive correlation, or vice versa they are said to have a negative correlation. The *Pearson Correlation* (ρ) is one of the most used linear correlation measures and it is defined in Equation 1.

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

Its range is $[-1, 1]$ and it assumes values ± 1 if $Y = aX + b, \quad a \neq 0$ (i.e. there is a linear relationship). Therefore, if two random variables are statistically independent, the correlation is 0, but it is not true the opposite, because they may have a non-linear relationship.

High values of this correlation coefficient with respect to the target is synonym of data redundancy, so it could be helpful to drop those columns. A graphical representation of the Person Correlation is performed with an Heatmap, where each cell (i, j) represents the Person Correlation between the random variables X_i and X_j.

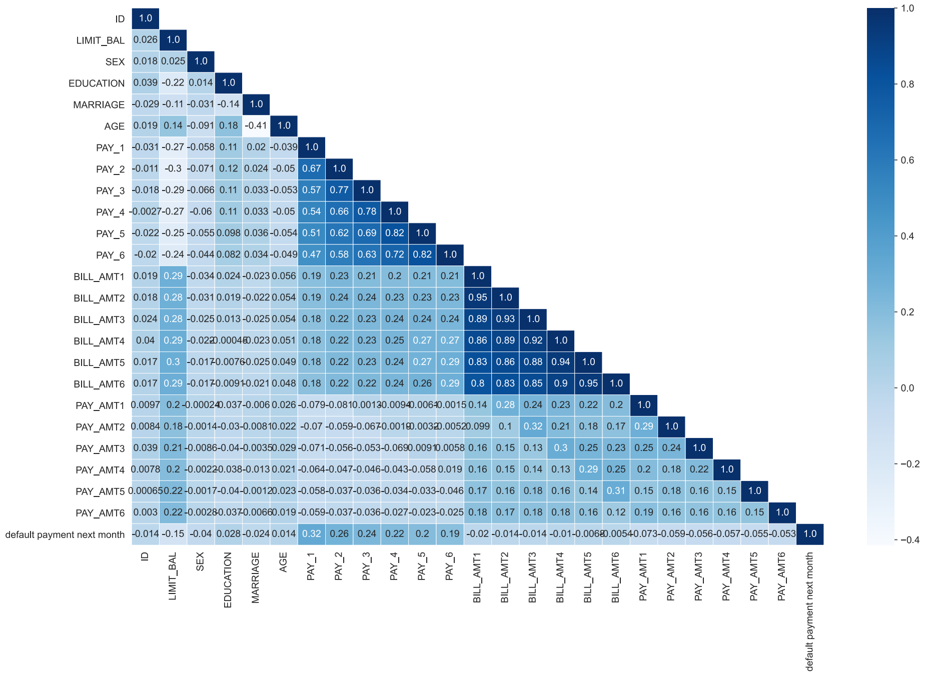

Figure 7: Heatmap

From Figure 7, as we may think, it is possible to observe a strong "internal" correlation among the groups of features such as *BILL*_*AMTX*, *PAY*_*X*. It is also interesting to notice that there is no feature with a strong relationship with the target. In fact, the features with an absolute value of the correlation less than 0.1 are 15 and none of the remaining ones have a correlation greater than 0.32.

3 Data preprocessing

3.1 Feature selection

Feature selection is the process of reducing the number of input variables. It is performed with the aim of reducing the computational cost but it may also improve the performances by leveraging the so called *Curse of dimensionality*, that will be explained in the following sections. The only feature that will be removed is *ID* - the identifier of the customer - because it does not add any meaningful information to the dataset. In order to have a more meaningful subset of features we will then refer to Principal Component Analysis (PCA).

3.2 Data splitting

In order to evaluate the performances of a classification algorithm it is a common practise to divide the dataset into two partitions, called training and test set. The training set will be used to fit the machine learning model, whereas the test set will be used to evaluate the fit machine learning model. In this case, since there is a satisfactory number of samples, 75% of the initial dataset has been used for the training procedure and the remaining 25% for testing, while preserving the initial data distribution (attribute stratify).

In order to tune the hyperparameters, another data splitting is necessary. In particular we call these new partitions train set and validation set. It has been used Stratified K-Fold Cross Validation (with $k = 5$), whose pipeline is as follows:

- Split the dataset into k groups (while maintaining the original distribution)
- For each unique group:
  - Take the group as a hold out or test data set
  - Take the remaining groups as a training data set
  - Fit a model on the training set and evaluate it on the test set

Once obtained the best hyperparameters, train again the model with both train and evaluation set and finally evaluate it on the test set.

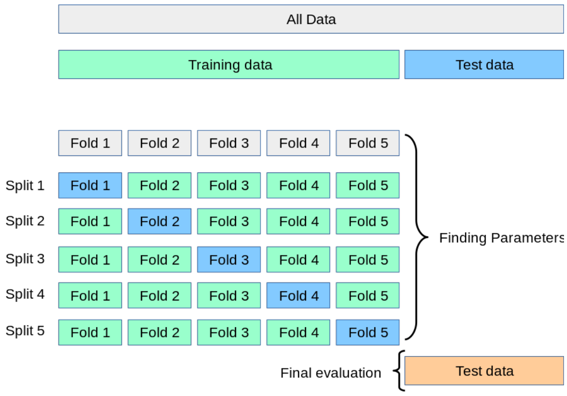

Figure 8: Data Splitting

3.3 Outlier detection

Outliers are extreme values that deviate from other observations on data. Some of the most common causes of outliers are the following:

- an entity may seem different because it belongs to another class,
- there is always the probability (even if it is lower) that we record real values far from the regular patterns,
- some technical or human errors occur.

The presence of a significative amount of outliers in some cases could drastically affect the performances. Therefore, it is a common practise to train the models with and without outliers, in order to see how it actually affect the learning phase. There are several different techniques for removing outliers. It is possible to find them thanks to some graphical representation of the data (e.g. Boxplot), density-based approaches (e.g. Local Outliers Factor) and many more.

3.3.1 Local Outliers Factor

The Local Outlier Factor (LOF) algorithm is an unsupervised method that computes the local density deviation of a given data point with respect to its k-nearest neighbors. It considers as outliers the data points that have a significantly lower density than their neighbors.

As shown in Figure 10, point D has a lower local density than point A. So, if we consider the red circle as the learned decision function, the maximum distance will be the one between the points A and B. Thus point D is out of reachablity and it will be denoted as an outlier.

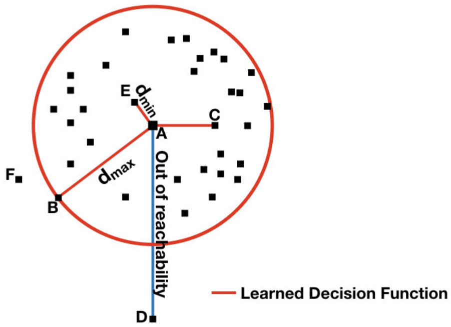

Figure 9: Local Outliers Factor

Let k-$distance(A)$ be the distance of the object A to the k-th nearest neighbor. This distance is used to define the so called reachability distance.

$$rd(A, B) = \max\{k\text{-}distance(B), d(A, B)\} \tag{2}$$

It is the distance among the points and it will be at least as the k-$distance(B)$. It means that all the k nearest points of B will be treated as the same.

The *local reachability density* is the inverse of the average reachability distance of the object A from its neighbors and it is formalized in Equation 3:

$$lrd_k(A) = \frac{|N_k(A)|}{\sum_{B \in N_K(A)} rd_k(A,B)} \tag{3}$$

where $N_k(A)$ is the set of k-nearest neighbors of A. Finally, the local reachability densities are compared with those of the neighbors using Equation 4:

$$LOF_k(A) = \frac{\sum_{B \in N_k(A)} \frac{lrd_k(B)}{lrd_k(A)}}{|N_k(A)|} = \frac{\sum_{B \in N_k(A)} lrd_k(B)}{|N_k(A)| lrd_k(A)} = \tag{4}$$

It is the average local reachability density of the neighbors divided by the object's own local reachability density.

This ratio will assume values:

- ~ 1 if there is similar density as neighbors
- < 1 if there is higher density than neighbors
- > 1 if there is a lower density than neighbors (outlier)

3.4 Feature scaling

Feature scaling is the process through which you change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. In fact, it gives equal weights/importance to each variable, avoiding the fact that some models would give an higher weight to those features with bigger numbers and wider range of values.

There are several scaling techniques, and the *z-normalization* has been applied. It is performed by removing on each data point the mean μ and dividing it for the standard deviation σ, as in Equation 5. Hence, the normalized dataset will have mean equal to zero and standard deviation equal to one.

$$Z = \frac{x - \mu}{\sigma} \tag{5}$$

3.5 Dimensionality reduction

Dimensionality reduction is the transformation of the data from an high dimensional space to a low dimensional space. It has several advantages, one of them is that it helps avoiding the *curse of dimensionality*, a very common phenomenon in Big Data environments. In particular, it states that when the dimensionality increases, the volume of the space drastically increases as well and the data become more and more sparse. As a consequence, as the number of dimensions grows, the amount of data that we need to generalize enough grows exponentially.

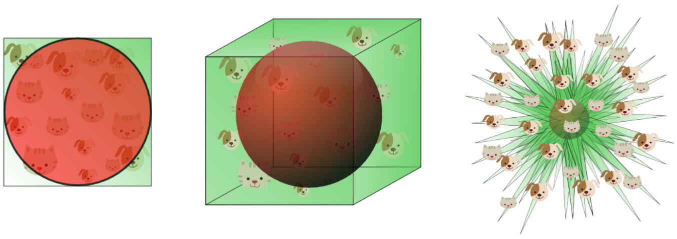

Figure 10: Curse of dimensionality

To better understand this concept, let us consider a unit square that represents the 2D feature space and the points within a unit distance inside a circle. Notice that the sparse points are likely to be on the corners. While the number of dimensions d increase, the volume of a unit hypercube is always 1 whereas the volume of the hypersphere exponentially decreases. In fact, its volume is defined as follows:

$$V(d) = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2} + 1)} 0.5^d \tag{6}$$

Intuitively, the higher will be the number of dimensions, the lower will be its volume. Therefore, most of the data will lie on the corners of the hypercube and the Eucledian distance starts losing their importance because the the distance in between any two points becomes more and more similar. Even though we do not have thousands of features, some distance-based algorithms (e.g. KNN) may benefit from it anyway.

3.5.1 Principal Component Analysis

The most used technique for dimensionality reduction is Principal Component Analysis (PCA). Formarly, given the examples $x_1, ..., x_m \in \mathbb{R}^d$ we want to solve the following optimization problem:

$$\underset{W \in \mathbb{R}^{n,d}, U \in \mathbb{R}^{d,n}}{\text{argmin}} \sum_{i=1}^{m} ||x_i - UWx_i||_2^2 \quad n < d \tag{7}$$

where the compression matrix W encodes x_i in a lower dimension $y_i = Wx_i$ and the recovering matrix U decodes y_i to an higher dimension $x_i = Uy_i$. Therefore, the goal is to find these two matrices that minimize the square of errors. Then, thanks to a lemma on projection [3] we know that the columns of U are orthonormal (i.e. $U^TU = I$) and that $W = U^T$. With some algebraic manipulation it can be rewritten as follows:

$$\underset{U \in \mathbb{R}^{n,d}, U^TU=I}{\text{argmax}} \quad trace(U^T(\sum_{i=1}^{m} x_i x_i^T)U) \tag{8}$$

where $A = \sum_{i=1}^{m} x_i x_i^T$ is the so called Scatter Matrix. Thanks to its symmetry, it can be decomposed through a spectral decomposition $A = VDV^T$ where D is diagonal whose elements are the corresponding eigenvalues and V is orthonormal whose columns are the eigenvectors. We can assume that $D_{1,1} \geq ... \geq D_{d,d} \geq 0$ (D is positive semidefinite). Hence, the solution of this optimization problem is to assign to the matrix U the columns $(u_1, ..., u_n)$, that are the n largest eigenvectors of the scatter matrix A and $W = U^T$.

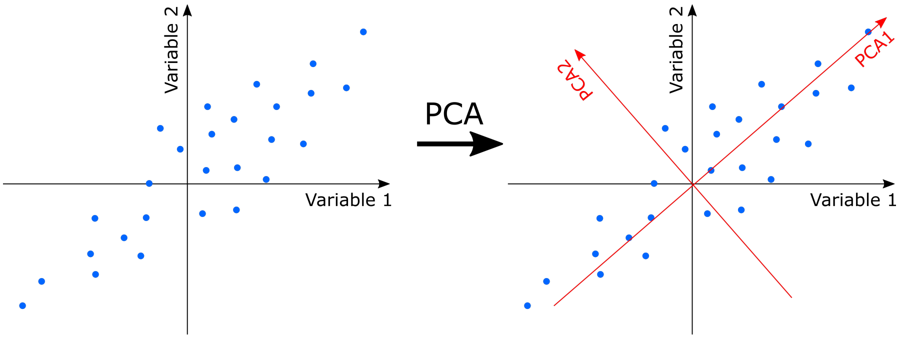

Figure 11: PCA 2D example

A common practise while dealing with PCA is to center the examples before applying it. It is necessary for performing classical PCA to ensure that the first principal component describes the direction of maximum variance. Otherwise, the first principal component might instead correspond to the mean of the data.

PCA can be interpreted from different point of views, in particular it can be also seen as a *variance maximization problem*. In fact, each principal component will explain some variance of the initial dataset through a linear combination of the features. What is important to highlight is that the coefficients of the linear combination of the first principal component will represent the direction along which the data has maximal variance, and so on in a decreasing order.

Formaly, the first principal is the normalized linear combination of the features $(X_1, .., X_p)$ that has the maximal variance.

$$Z_1 = \phi_{11}X_1 + ... + \phi_{p1}X_p \quad s.t. \sum_{j=1}^{m} \phi_{j1}^2 = 1 \text{ (normalized)} \tag{9}$$

where $(\phi_{11}, ..., \phi_{p1})$ are the loadings of the first principal component. Given an $n \times p$ dataset, in order to determine the first principal component we assume that the data has been centered and we look for the linear combination of the sample feature of the form

$$z_{i1} = \phi_{11}x_{i1} + ... + \phi_{p1}x_{ip} \tag{10}$$

that has the largest sample variance, subject to the normalization constraint. This constraint is used because otherwise setting these loadings to be arbitrarily large in absolute value could result in an arbitrarily large variance.

Once the first principal component Z_1 has been determined, we start looking for the second one, having the maximal variance out of all linear correlations uncorrelated with Z_1. With these settings, it results that all the principal components will be orthonormal. Hence, we want to find a new orthonormal basis such that we have the maximal variance explained.

In order to determine the proper number of principal components it is necessary to look for a trade off between the proportion of variance explained and the number of principal components. Of course, the higher will be the number of dimensions, the higher will be the explained variance, but recall that we are still trying to reduce the dimensions. Therefore, a rule of thumb is to choose as many components until you reach a satisfactory variance percentage $(80\% - 90\%)$. As it is possible to observe from the screeplot in Figure 12, it reached 85% of the explained variance with 11 principal components.

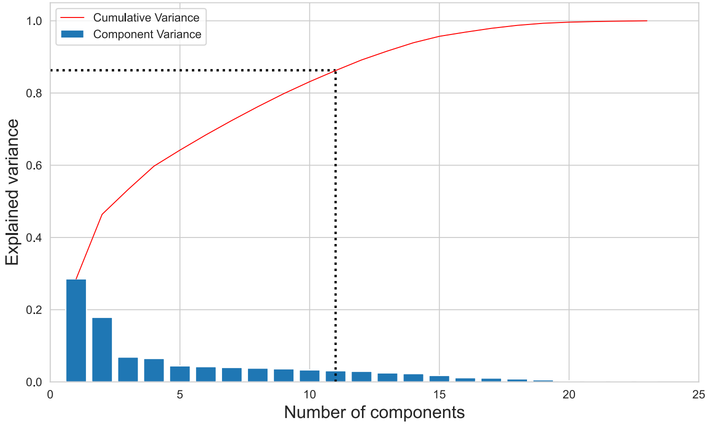

Figure 12: Screeplot of PCA

3.6 Resampling

Whenever a model learn from an imbalanced dataset, it will be harder to make very accurate predictions, because most of the models tend to prefer the majority class. In fact, in such a circumnstances it is a common practise to rebalance (only) the training set. Balancing a dataset can be done through under-sampling the majority class, over-sampling the minority class or even with both of these two techniques.

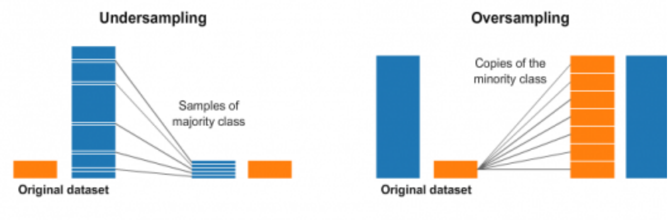

Figure 13: Resampling techniques

3.6.1 Under-sampling

Under-sampling is a method through which some samples from the majority class will be discarded in order to balance it with the minority class. The main disadvantage of this technique is that we can lose some (or even a lot of) meaningful samples. There are several ways for under-sampling, and Near Miss has been applied. It uses some heuristics based on KNN for selecting the samples to remove. There are three version of this algorithm:

- NearMiss-1 selects the positive samples (majority class) for which the average distance to the N closest samples of the negative class (minority class) is the smallest.
- NearMiss-2 selects the positive samples for which the average distance to the N farthest samples of the negative class is the smallest
- NearMiss-3 is a 2-steps algorithm. First, for each negative sample, their M nearest-neighbors will be kept. Then, the positive samples selected are the one for which the average distance to the N nearest-neighbors is the largest.

NearMiss-3 has been applied because it is the one less affected by the noise, thanks to the first step sample selection.

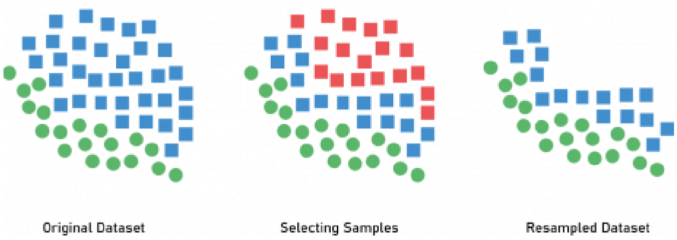

Figure 14: Near Miss

3.6.2 Over-sampling

Over-sampling, on the other hand, generate new samples in the minority classes. The most common technique is SMOTE, acronym of Synthetic Minority Oversampling Technique. It selects examples that are close in the feature space, drawing a line between the examples and drawing a new sample along this line. The main limitation is that these synthetic points are created without considering the majority class, so it may create (or intensify) overlapping distributions.

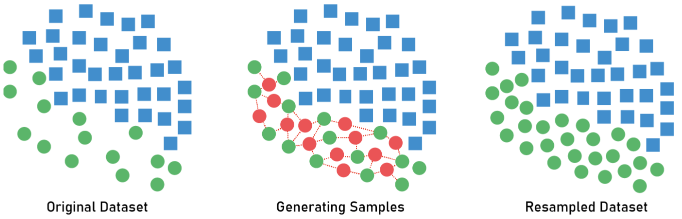

Figure 15: SMOTE

3.6.3 Combination of under- and over-sampling

Alternatively, it is possible to perform both under- and over-sampling. One algorithm that allows to do so is SMOTEENN, a combination of SMOTE and Edited Nearest Neighbours. In particular, it combines SMOTE for generating synthetic examples for the minority class and ENN ability to delete some observations from both classes that are identified as having different class between the observation's class and its K-nearest neighbor majority class.

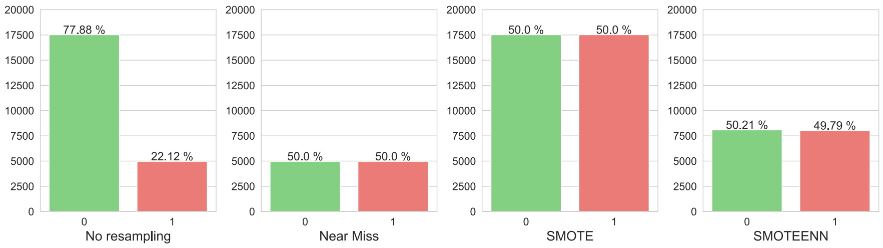

Figure 16: Number of samples for each class and resampling technique

3.6.4 Strategy

For the sake of completeness we have performed tests with all these techniques, in order to analyze how the algorithms that we have selected behave with different rebalancing techniques.

4 Model evaluation

Model evaluation tries to measure how well the model is performing on unseen data. Different metrics can be used depending on the task, the data imbalance and other factors. While dealing with classification tasks, these are some of the most used ones:

- Accuracy : ratio between the number of correctly classified samples and the total number of samples
$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$
- Precision : fraction of correctly positive predicted values over the total predicted positives
$$Prec = \frac{TP}{TP + FP} \tag{12}$$
- Recall : fraction of correctly positive predicted values over the total actual positives
$$Rec = \frac{TP}{TP + FN} \tag{13}$$
- F1 : harmonic mean of precision and recall
$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{14}$$

TP:True Positive, TN:True Negative, FP:False Positive, FN:False Negative.

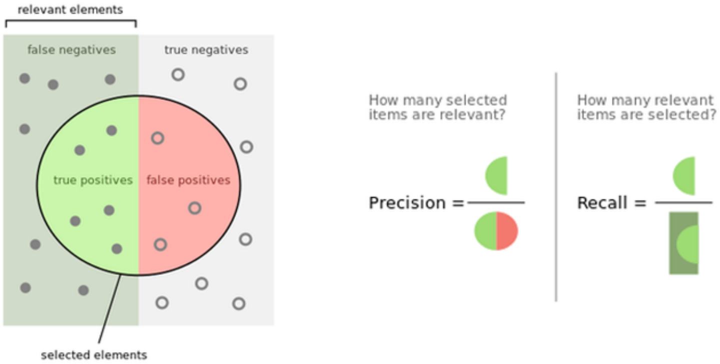

Figure 17: Precision and Recall

Since we are dealing with an heavy imbalanced dataset, maximizing the accuracy is not the best option because we may have an high number of correct classification, mostly from the majority class (that we know will be preferred by the model). In fact, we are strongly interested in the number of False Negatives (i.e. default classified as no default) without forgetting the number of False Positives (i.e. no default classified as default). Hence, we decided to maximize the F1 score on our tests with special attention to the Recall.

5 Model selection

In this section we explored different approaches and different models for solving our problem. In particular we have studied Random Forest, K Nearest Neighbors, Support Vector Machines and and Logistic Regression, following the Empirical Risk Minimization (ERM) paradigm.

In order to explore the ERM paradigm, we need some formalisms. Let us consider a set of observations X, a set of labels Y, a sampling distribution $D \sim X$ and a labelling function $f(\cdot) : X \to Y$. Given a finite training set $S = \{(x_1, y_1), ..., (x_n, y_n)\}$, our goal is to find the predictor rule $h : X \to Y$. Ideally, if $f(\cdot)$ and D are known, the *true error* of the predictor rule can be defined as follows:

$$L_{D,f}(h) = \underset{X \sim D}{\mathrm{P}}[h(x) \neq f(x) = D(\{x : h(x) \neq f(x)\}) \tag{15}$$

However, the learner do not know neither the distribution D nor the labelling function $f(\cdot)$. Hence, this definition cannot be applied. The learner will receive as input only the training set S and as a consequence we may just calculate the *training error*:

$$L_s(h) = \frac{|\{i \in [n] : h(x_i) \neq y_i\}}{n} \tag{16}$$

So, under the ERM paradigm, the goal is to find a predictor h that minimizes the training error $L_S(h)$. Intuitively, one may thing to select the predictor with the lowest training error, but it may fail to generalize. This phenomenon is quite famous in the Machine Learning world and it is called Overfitting.

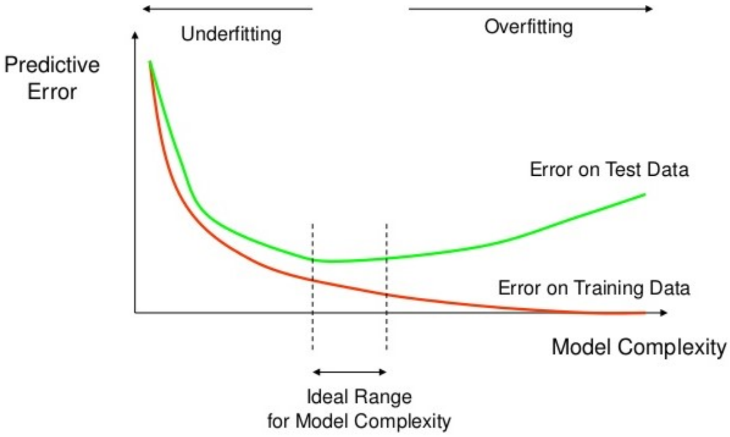

Figure 18: Underfitting and overfitting

In order to prevent it, a common solution is to apply ERM over a restricted set of hypothesis classes H and select the predictor $h \in H$ that achieve the minimum error.

$$ERM_H(S) \in \underset{h \in H}{\mathrm{argmin}} L_S(h) \tag{17}$$

Obviously, this restriction cannot be done blindly. It should be performed by taking into account some prior knowledge about the problem. Therefore, although we may choose a set of predictors H that allows to reduce the probability to overfit, we are somehow introducing some hidden bias. However, this prior knowledge is necessary because we know from the No-Free-Lunch theorem [4] that there is no universal learner. It proved that for every learner there exists a task on which it fail, even though this task can be perfectly achieved by some other learners. In order to select the apparently best hypothesis class H, we have to believe that it includes a predictor that achieves good performances and we also need to to consider that we cannot select the richest class. In order to answer to this fair question we have to mention the well known *bias-complexity trade off*. Therefore, let us decompose the error as approximation error and estimation error.

$$L_D(h_S) = \epsilon_{app} + \epsilon_{est} \qquad \epsilon_{app} = \min_{h \in H} L_D(h) \quad \epsilon_{est} = L_D(h_S) - \epsilon_{app} \tag{18}$$

The approximation error is the minimum risk achievable by a predictor in H, so enlarging H will reduce it. Instead, the estimation error is the difference among the approximation error and the error achieved by ERM and it will depends on the size of both the training set and hypothesis class. Since enlarging H will decrease ϵ_{app} and increase ϵ_{est}, we have to find a good balance among them.

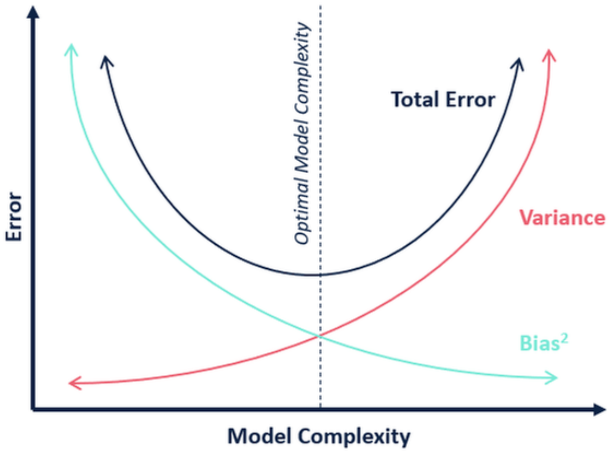

Figure 19: Bias Complexity Trade Off

5.1 Decision Tree and Random Forest

Decision tree is a supervised learning algorithm that predicts the new labels by recursively splitting the predictor space into non overlapping distinct regions. For every observation that falls into any region we make the same prediction, which is the majority class in that region (for classification) or the mean of the response value (for regression).

Since considering every possible partition is computationally infeasible (NP-hard problem), the tree is constructed with greedy top down approach, known as recursive binary splitting. It is *top-down* because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree. Then, it is *greedy* because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

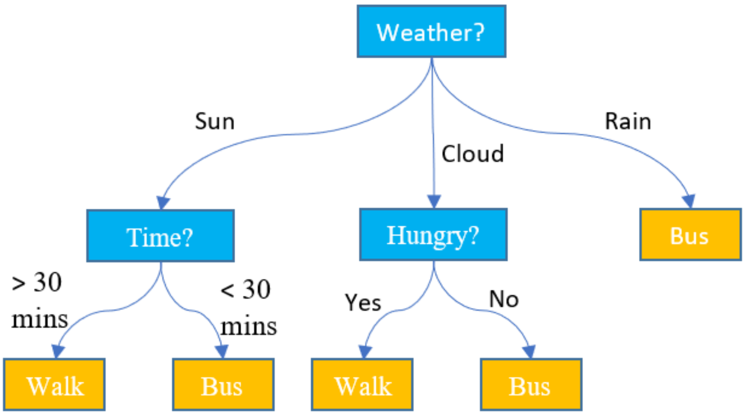

Figure 20: Decision Tree

In order to select the best split to make for classification problems, we consider the split that minimize the so called Gini Index, a measure of the variances across the classes (impurity). Alternatively, another measure that can be taken into account is the Entropy, that measures the so called "information gain", defined as the disorder of the features with the current target. They are formalized as follows:

- $Gini = 1 - \sum_j p_j^2$
- $Entropy = \sum_j p_j \log_2 p_j$

However, even though a decision tree is fairly interpretable, it is typically less accurate and robust compared to more sophisticated algorithms. Therefore, we avoided further tests with it. A more advanced tree-based algorithm is Random Forest, an ensemble method that combines multiple decision trees (that is why it is called "forest") with the bagging technique, obtaining a more accurate and robust model.

The idea of bagging is to make predictions on B bootstrapped copies of the training set (i.e. sampled with replacement). In order to decorellate the trees, feature bagging is the best option. In particular, each time a split in a tree is considered, a fixed number of features (typically $\sqrt{p}$) is randomly chosen. The final prediction will be given by a majority voting scheme of all the estimators (or the average, in regression tasks). This is why this algorithm is fairly robust to noise and outliers and will have much less variance rather than a single decision tree. In fact, if we consider n observations with variance σ^2, the variance of the sample mean will be exactly $\frac{\sigma^2}{n}$.

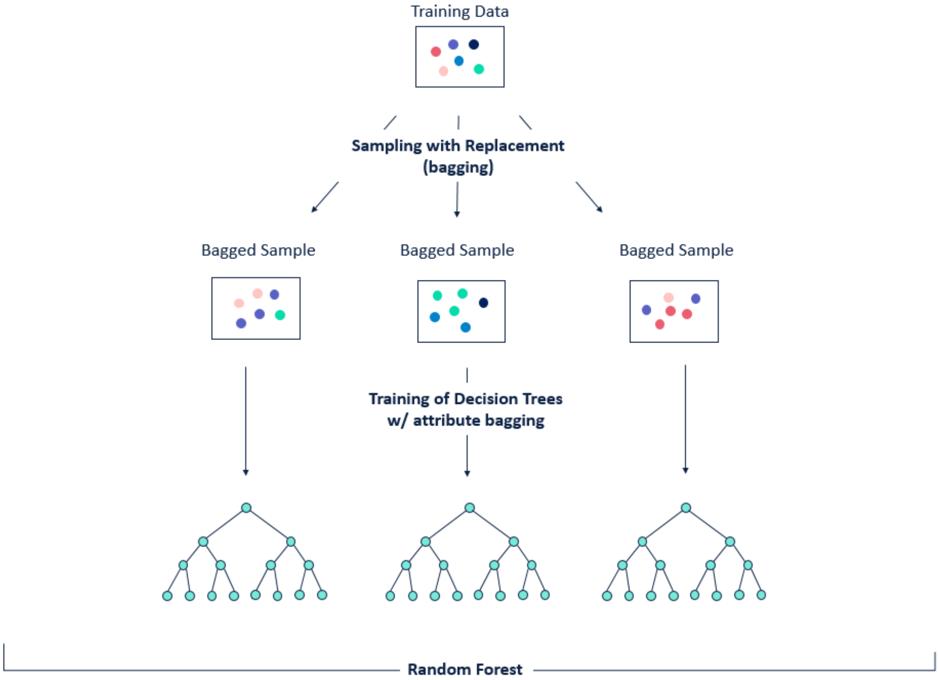
Training Data
Sampling with Replacement (bagging)
Bagged Sample
Bagged Sample
Bagged Sample
Training of Decision Trees w/ attribute bagging
Random Forest

Figure 21: Random Forest

The best F1 score was 0.5, obtained with SMOTE and the following hyperparameters: {criterion': 'entropy', 'max_features': 'sqrt','n_estimators': 150}. Further comments on the results will be discussed in the last section of this document.

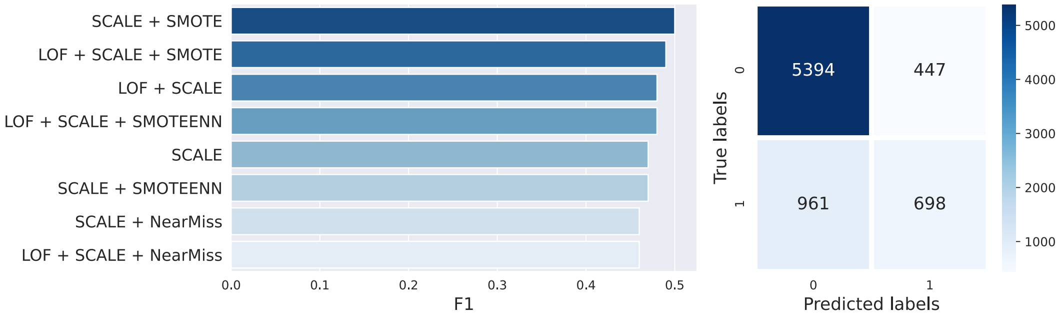

Figure 22: F1 score results and Confusion Matrix of the best configuration

5.2 K Nearest Neighbors

K Nearest Neighbors is a supervised learning algorithm based on the assumption that the points in the neighborhood belong to the same class. Therefore, given a positive integer k and a test observation x_0, KNN identifies the the k closest points to $x_0, (N_0)$ and then, it estimates the conditional probability for class j as the fraction of points in N_0 whose target value is equal to j:

$$Pr(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in N_0} I(y_i = j) \tag{19}$$

Finally, it classifies the test observation x_0 with the class having the highest conditional probability. The most important hyperparameters are the distance metric and k. There are several available distance metrics (Euclidean, Malhanobis and so on) but they are likely to obtain comparable results [5]. What is truly important is the choice of k (number of neighbors). If it is high, the variance decreases but the bias increases, whereas if it is small, the bias will decrease and the variance will increase because it will be more sensitive to the outliers.

In order to choose the value of k it is a common practise to derive a plot between error rate and k denoting values in a defined range. Then, choose k having a minimum error rate, or with a good balance among the computational complexity and the error rate. Since we are tuning every kind of preprocessing and hyperparameters through a GridSearch, we skipped this preliminary step.

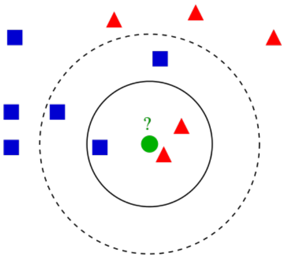

Figure 23: K Nearest Neighbors

All the tests have been performed with PCA in order to avoid the curse of dimensionality and a consequent losing of importance of the distance metrics. The best f1-score was 0.47, obtained with SMOTE the following hyperparameters : {'n_neighbors':50, 'metric':'euclidean'}.

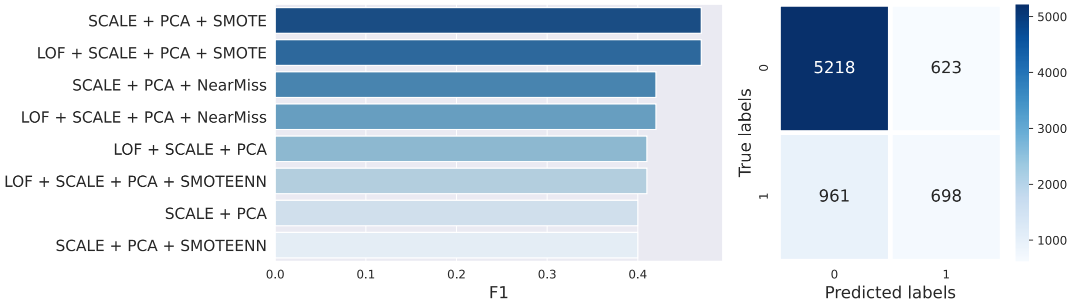

Figure 24: KNN results and Confusion Matrix of the best configuration

5.3 Support Vector Machine

Support Vector Machine is a supervised learning algorithm widely used in Natural Language Processing and Computer Vision tasks.

5.3.1 Hard SVM

Assuming that the data is linearly separable, the goal is to find the *best separating hyperplane*. Since we may find an infinite number on separating hyperplanes, intuitively, the best one will be the one with the maximize the margins, that actually means to maximize the minimum distance among point and plane.

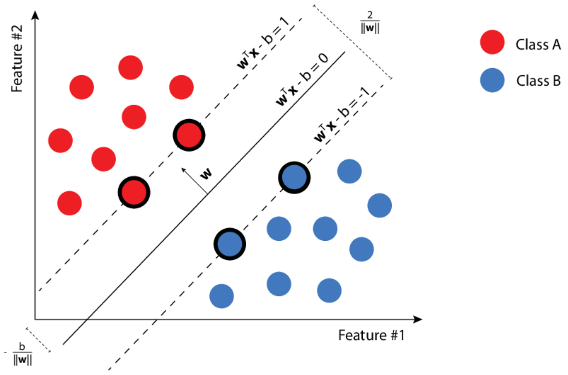

Figure 25: Hard Support Vector Machines

The keyword "hard" is used because we do not let the points lie within the boundaries. The formulation of the Hard SVM is defined as follows:

$$\underset{(w,b):\|w\|=1}{\operatorname{argmax}} \quad \min_{i\in[m]} |\langle w, x_i\rangle + b| \quad s.t. \quad \forall i, y_i(\langle w, x_i\rangle + b) > 0 \tag{20}$$

Notice that the condition on the right is the definition of linearly separability. The idea is that if the argument is negative, y_i is negative and vice versa. So, the product must be greater than 0. Equivalently the same rule for the hard SVM can be rewritten as a quadratic optimization problem:

$$(w_0, b_0) = \underset{(w,x)}{\operatorname{argmin}} \|w\|^2 \quad s.t. \quad y_i(\langle w, x_i\rangle + b) \geq 1 \tag{21}$$

Its solution will be given by $(\hat{w}, \hat{b}) = (\frac{w_0}{||w_0||}, \frac{b_0}{||w_0||})$. Notice that here we forced the margin to be 1, but now the units in which we measure the margin scale with the norm of w.

Finally, the name "Support Vector Machine" relies on the solution itself. In fact, thanks to the Fritz John optimality conditions we can affirm that given w_0 and let $I = \{i : |\langle w_0, x_i\rangle = 1\}$, there exists $\alpha_i, i \in [m]$ such that:

$$w_0 = \sum_{i\in I} \alpha_i x_i \tag{22}$$

It means that the solution is a linear combinations of the examples that are at distance $\frac{1}{||w_0||}$ from the separating hyperplane.

5.3.2 Soft SVM

In real life scenarios, having perfectly linearly separable dataset is quite uncommon, therefore the Hard-SVM condition may be too restrictive. In order to overcome this limitation, we can soften the margins. Hence, we let the points penetrate the margins, formarly:

$$\underset{w,b,\xi}{\operatorname{argmin}}(\lambda||w||^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i) \quad s.t. \quad \forall i, y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i \tag{23}$$

ξ_i is called slack variable and it measures by how much the linearly separability constraint has been violated. Therefore, we want to minimize the norm of w (i.e. the margin) and the average of the violations ($\frac{1}{m}\sum_{i=1}^{m}\xi_i$). Finally, the parameter λ is used for balancing the tradeoff among the two previous terms.

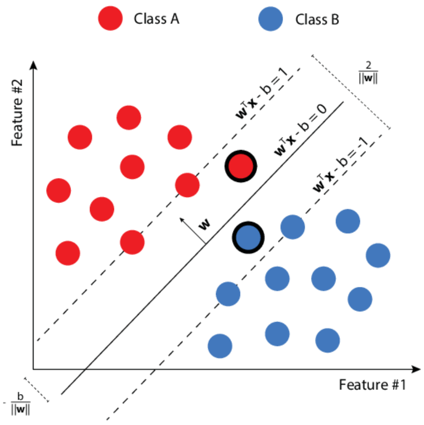

Figure 26: Soft Support Vector Machines

The problem defined in Equation 23 can be rewritten as a regularized loss minimization problem. In fact, let us consider the hinge loss:

$$l^{hinge}((w,b),(x,y)) = \max\{0, 1 - y(\langle w, x\rangle + b)\} \tag{24}$$

Since the slack variable must be non negative, its best value will be 0 and with a simple computation its value will be $1 - y_i(\langle w, x_i\rangle + b)$. Hence, we can conclude that the slack variable $\xi_i = l^{hinge}((w,b),(x_i,y_i))$. So that, considering L_S^{hinge} the average hinge loss over the training set S, the problem can be rewritten as follows:

$$\min_{w,b} \lambda||w||^2 + L_S^{hinge}(w,b) \tag{25}$$

5.3.3 Kernel trick

Soft SVM is able to handle noise and outliers in almost linearly separable conditions. But most of the time, the data we are dealing with, is not linearly separable, therefore, even softening the margin may fail. In these cases it is possible to map the data into an higher dimensional space such that it will be linearly separable. However, if the data is mapped in a very high dimensional space, it will be very costly from a computational point of view.

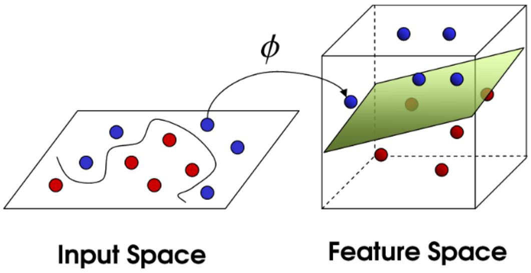

Figure 27: Kernel trick

The idea of kernel has been obtained by considering the dual of Equation 21 for the Hard SVM [6]:

$$\max_{\alpha \in \mathbb{R}^n : \alpha \geq 0} (\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle) \tag{26}$$

It is evident that the dual problem only involves inner products between instances, that is nothing but a linear kernel. There is no restriction of using new kernel functions with the aim of measuring the similarity in higher dimensions. So, given a non linear mapping $\psi : X \to F$, the kernel function is defined as:

$$K(x, x') =< \psi(x), \psi(x') > \tag{27}$$

and the dual problem simply becomes:

$$\max_{\alpha \in \mathbb{R}^n : \alpha \geq 0} (\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \psi(x_i), \psi(x_j) \rangle) \tag{28}$$

Therefore, the kernel trick enables computationally efficient implementation of learning, without explicitly accessing the elements in the feature space. We just need to understand how to calculate the inner products within the new mapped feature space. However, the complexity now depends on the size of the dataset, because for M data points we need to compute $\binom{M}{2}$ inner products. Some of the most common kernel functions are:

- Polynomial : $K(x_i, x_j) = (x_j \cdot x_j + 1)^p$
- Gaussian : $K(x_i, x_j) = \exp\{\frac{-1}{2\sigma^2 (x_i - x_j)^2}\}$
- Radial Basis Function : $K(x_i, x_j) = \exp\{-\gamma(x_i - x_j)^2\}$

Except for these well known kernel functions, it is possible to define new functions but it is easy to end up with highly inefficient ones (in favour of the curse of dimensionality). Thanks to the Mercer's theorem [7] we know that a necessary and sufficient condition for a function to be a kernel function, and so, to implement the inner product is that the Gram matrix (i.e. Kernel Matrix) is positive semidefinite.

All the tests have been performed with PCA in order to reduce the computational complexity and to speed up the training process. The best F1-score was 0.46, obtained with NearMiss. With these settings, the best configuration was {'C':1.2, 'kernel':'rbf'}. Even though the F1-score is lower than the one obtained with Random Forest, here we have a very high recall (0.66), penalized by a low precision (0.36). From the perspective of a bank, it is much better to make more controls on the False Positives instead of missing potential default from their clients (False Negatives).

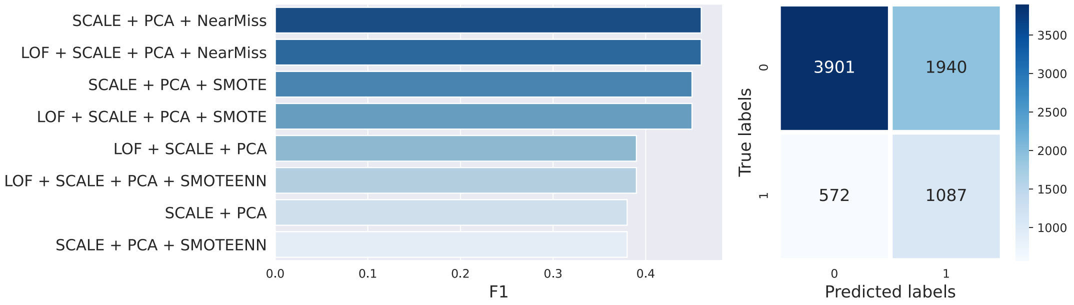

Figure 28: SVM results and Confusion Matrix of the best configuration

5.4 Logistic Regression

Logistic Regression classifies the response variable by exploiting the probability that the given data point belongs to a specific class. Compared to Linear Regression, instead of fitting a straight line to the data, it fits the logistic function defined in Equation 29 (i.e. sigmoid - S-shaped), that squeezes the result between 0 and 1.

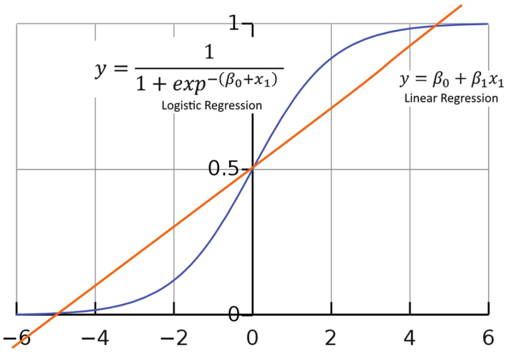

Figure 29: Logistic and Linear Regression

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{29}$$

This equation models the probability of a given class, and in order to make a proper classification we need a threshold (typically 0.5). Therefore, up to this value, the data point will be classified for the given class and vice versa.

Assuming a linear relationship between the predictor variables and the log odds (i.e. probability of event divided by probability of no event) of the response variable means that:

$$\log \frac{p(x)}{1 - p(x)} = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p = \langle w, x_i \rangle + b \tag{30}$$

With some mathematical manipulations we obtain:

$$p(x_i) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + ... + \beta_p x_p)}} = \frac{1}{1 + e^{-(\langle w, x_i \rangle + b)}} \tag{31}$$

Therefore, we want to find the best β_i and we can do that by minimizing a cost function. Since the sigmoid function is non linear, if we use the Mean Squared Error (such as in Linear Regression), we will obtain a non convex function, with many local optimum. So, in order to obtain a convex function starting from the logistic one, we rely on the logarithm operator. Hence, the cost function for logistic regression is defined as:

$$Cost(\sigma(x), y) = \begin{cases} -\log(p(x)) & \text{if } y = 1 \\ -\log(1 - p(x)) & \text{if } y = 0 \end{cases} \tag{32}$$

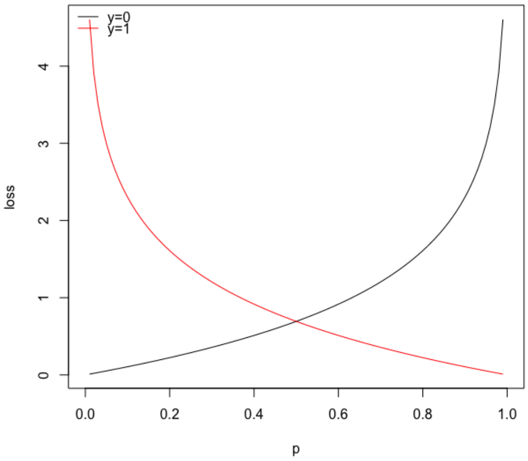

Figure 30: Logistic loss

Considering that y can assume only values 0 or 1, in a simplified way can rewrite it as:

$$Cost(p(x), y) = -y \log(p(x)) - (1 - y) \log(1 - p(x)) \tag{33}$$

It can be averaged over all observations, obtaining the so called Binary Cross Entropy Loss:

$$BCE = -\frac{1}{m} \left[\sum_{i=1}^{m} y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i)) \right] \tag{34}$$

This expression of the cost function can be derived from the Maximum Likelihood Estimation of a bernoulli distribution, to which we flip the sign (since we want to minimize) and scale with $\frac{1}{m}$.

The hyperparameters that we tuned are "C" (inverse of regularization strength) and "penalty" (regularization, l1 or l2). The best f1-score was 0.49, obtained with both SMOTE and NearMiss. The best configuration of hyperparameters was {'C':0.6, 'penalty':'l2'}.

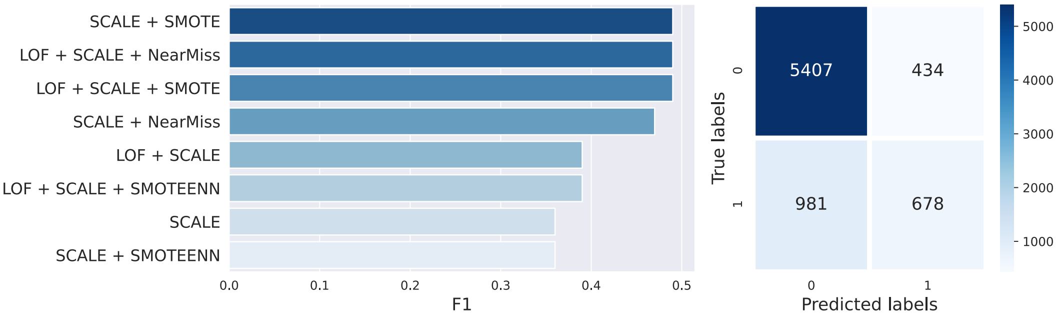

Figure 31: Logistic Regression results and Confusion Matrix of the best configuration

6 Pipeline

To summarize, even though some models have not used one of these steps on their best configurations, the following is the full pipeline of the proposed approach.

Firstly, the whole dataset has been cleaned by removing and grouping the non-representative categorical classes into an "unkown" class. Then, it has been splitted into training and test set, and again, the training set has been splitted into training set and validation set. The outliers have been removed from the training set through Local Outliers Factor.

Scaling and PCA have been fitted only with the training set, and based on the statistics obtained, the training set, validation set and test set have been normalized and projected to a new feature space with 11 principal components (85% of the variance). Next, the training set has been resampled with one between NearMiss, SMOTE and SMOTEENN.

After that, the best configuration of preprocessing and hyperparameters has been tuned with K fold Cross Validation (5 folds), maximizing the F1-score. Finally, the best configuration found so far has been tested with the namesake test set and the results have been reported.

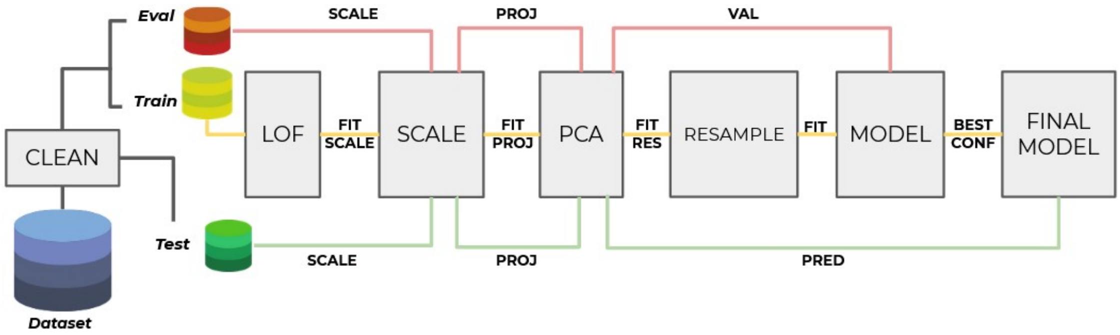

Figure 32: Complete pipeline

7 Conclusions

Different machine learning alorithms have been tested, as well as different preprocessing techniques. Tu sum up, our goal was to maximize the F1-score, the harmonic mean between precision and recall with a particular attention to the Recall. The best F1-score was obtained by Random Forest (0.51) whereas the highest Recall - that may be preferred in such a circumnstances - was achieved by Support Vector Machine (0.67).

By looking more in depth at the results, SMOTE improved the performances more than the remaining resampling techniques. In particular the configurations with SMOTEENN or no resampling were most of the times at the bottom of the bar chart. In fact, Near Miss seems to be always one little step behind SMOTE except with SVM that performed slightly better. Take into account that even though they have somehow comparable results, with Near Miss you have much less data to train, therefore, if the computational cost is a constraint, it may help.

Further considerations can be done through the implementation of Gradient-Boost based models such as Gradient Boosting Classifier or SGD Classifer, both implmented in Scikit-Learn [8; 9]. Finally, other outliers management approaches could be also evaluated, such as Isolation Forest, that isolates the observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature [10].

References

[1] Che-hui Lien I-Cheng Yeh, " "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients"

[2] I-Cheng Yeh , "Default of credit card clients Data Set"

[3] Shai Shalev-Shwartz and Shai Ben-David, "Understanding Machine Learning: From Theory to Algorithms", Lemma 23.1, Pages 279–280

[4] Shai Shalev-Shwartz and Shai Ben-David, "Understanding Machine Learning: From Theory to Algorithms", No-Free-Launch Theorem (5.1), Pages 37–40

[5] Kittipong Chomboon, Pasapitch Chujai et al, "An Empirical Study of Distance Metrics for k-Nearest Neighbor Algorithm"

[6] Shai Shalev-Shwartz and Shai Ben-David, "Understanding Machine Learning: From Theory to Algorithms", Duality, Pages 175–176

[7] Shai Shalev-Shwartz and Shai Ben-David, "Understanding Machine Learning: From Theory to Algorithms", Lemma 16.2, Page 222

[8] Scikit Learn, "Gradient BoostingClassifier"

[9] Scikit Learn, "SGD Classifier“

[10] Scikit Learn, "Isolation Forest“