POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Mathematics in Machine Learning

Default of credit card clients

Professors
prof. F. VACCARINO
prof. M. GASPARINI

Candidate
Matteo Merlo, 287576

A.Y 2021-2022

Contents

Abstract

The aim of this study is to exploit some supervised machine learning algorithms to identify the main factors that determine the probability of a credit card default, underlining the mathematical aspects and the methods used. Credit card defaults may occur when you have become severely broke on credit card payments. In order to increase market share, Taiwan's banks have issued excess cash and credit cards to unskilled applicants. At the same time, most cardholders, regardless of their repayment capability, were using their credit card excessively for consumption and have accumulated huge credits and debts.

The goal is to build an automated model to both identify central drivers and predict credit card default based on customer information and historical transactions. Next, the general concepts of the supervised machine learning paradigm are presented, along with a detailed explanation of all the techniques and algorithms used to build the models. In particular, Logistic Regression, Random Forest and Support Vector Machines algorithms have been applied.

The repository of this paper is available at: https://github.com/MatteoM95/Default-of-Credit-Card-Clients-Dataset-Analisys

1 Introduction

Since 1990, the Taiwanese government has allowed the formation of new banks. In order to increase market share, these banks have issued excess cash and credit cards to unskilled applicants. At the same time, most cardholders, regardless of their repayment ability, have abused their credit card for consumption and piled up heavy credit card debt and cash. Default occurs when a credit card holder is unable to comply with the legal obligation to repay. The crisis has caused a severe blow to confidence in consumer credit and has been a major challenge for both banks and cardholders [1].

In a well-developed financial system, crisis management is downstream and risk prediction is upstream. The primary purpose of risk forecasting is to use financial information, such as corporate financial statements, customer transaction and refund records, etc., to predict individual customer business performance or credit risk and reduce damage and uncertainty.

In this project, the aim is to reliably predict who is at risk of defaulting. In this case, the bank may be able to prevent the loss by providing the customer with alternative options (such as forbearance or debt consolidation, etc.). Then, we build an automated model based on customer information and historical transactions that can identify key factors and predict credit card default.

2 Exploratory Data Analysis

2.1 Dataset Description

The Default of Credit Card Clients dataset contains 30 000 instances of credit card status collected in Taiwan from April 2005 to September 2005. The dataset employs the binary variable `default payment next month` as response variable. It indicates if the credit card holders will be defaulters next month ($\text{Yes} = 1, \text{No} = 0$). 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 [2].

- Client personal information:
  1. `LIMIT_BAL`: Amount of given credit (in *New Taiwan* dollars): it includes both the individual consumer credit and his/her family (supplementary) credit.
  2. `SEX` : 1 = male, 2 = female
  3. `EDUCATION`: 1 = graduate school; 2 = university; 3 = high school; 4 = others.
  4. `MARRIAGE`: Marital status, 1 = married; 2 = single; 3 = others.
  5. `AGE`: Age in years.
- History of past payments from April to September 2005, i.e., the delay of the past payment referred to a specific month:
  6. `PAY_0`: Repayment status in September, 2005.

7. `PAY_2`: Repayment status in August, 2005.
8. `PAY_3`: Repayment status in July, 2005.
9. `PAY_4`: Repayment status in June, 2005.
10. `PAY_5`: Repayment status in May, 2005.
11. `PAY_6`: Repayment status in April, 2005.

The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; ...; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

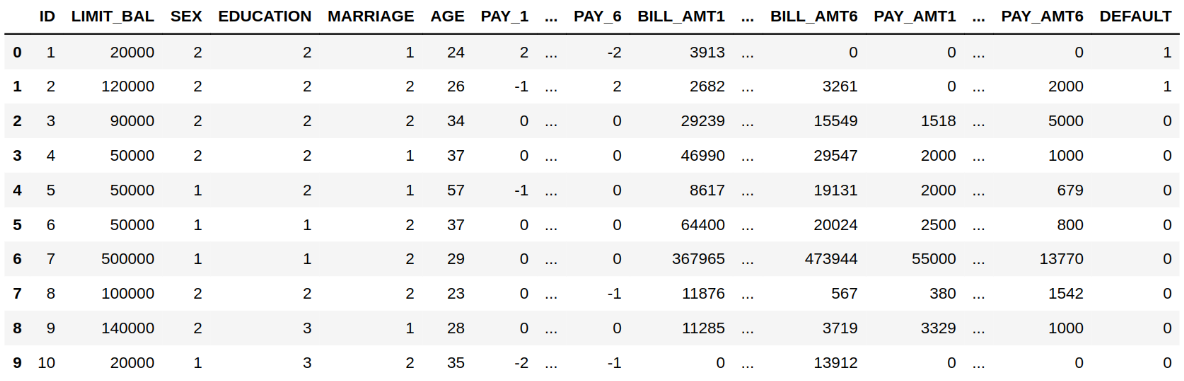
- Amount of bill statement (in *New Taiwan* dollars), i.e. a monthly report that credit card companies issue to credit card holders in a specific month:

12. `BILL_AMT1`: Amount of bill statement in September, 2005.
13. `BILL_AMT2`: Amount of bill statement in August, 2005.
14. `BILL_AMT3`: Amount of bill statement in July, 2005.
15. `BILL_AMT4`: Amount of bill statement in June, 2005.
16. `BILL_AMT5`: Amount of bill statement in May, 2005.
17. `BILL_AMT6`: Amount of bill statement in April, 2005.

- Amount of previous payment (in *New Taiwan* dollars):

18. `PAY_AMT1`: Amount of previous payment in September, 2005.
19. `PAY_AMT2`: Amount of previous payment in August, 2005.
20. `PAY_AMT3`: Amount of previous payment in July, 2005.
21. `PAY_AMT4`: Amount of previous payment in June, 2005.
22. `PAY_AMT5`: Amount of previous payment in May, 2005.
23. `PAY_AMT6`: Amount of previous payment in April, 2005.

In Figure 1 we can understand what the data looks like. The target `default.payment.next.month` is renamed `DEFAULT` to be short, while the PAY_0 column is renamed PAY_1



	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	...	PAY_6	BILL_AMT1	...	BILL_AMT6	PAY_AMT1	...	PAY_AMT6	DEFAULT
0	1	20000	2	2	1	24	2	...	-2	3913	...	0	0	...	0	1
1	2	120000	2	2	2	26	-1	...	2	2682	...	3261	0	...	2000	1
2	3	90000	2	2	2	34	0	...	0	29239	...	15549	1518	...	5000	0
3	4	50000	2	2	1	37	0	...	0	46990	...	29547	2000	...	1000	0
4	5	50000	1	2	1	57	-1	...	0	8617	...	19131	2000	...	679	0
5	6	50000	1	1	2	37	0	...	0	64400	...	20024	2500	...	800	0
6	7	500000	1	1	2	29	0	...	0	367965	...	473944	55000	...	13770	0
7	8	100000	2	2	2	23	0	...	-1	11876	...	567	380	...	1542	0
8	9	140000	2	3	1	28	0	...	0	11285	...	3719	3329	...	1000	0
9	10	20000	1	3	2	35	-2	...	-1	0	...	13912	0	...	0	0

Figure 1: Original dataset from UCI machine learning repository through pandas framework

2.2 Data distribution

From Figure 2 it is possible to see 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. This imbalance problem will make classification models focusing on the majority class overlooking the minority class if not addressed [3].

#	Column	Non-Null Count	Dtype
0	LIMIT_BAL	30000 non-null	int64
1	SEX	30000 non-null	int64
2	EDUCATION	30000 non-null	int64
3	MARRIAGE	30000 non-null	int64
4	AGE	30000 non-null	int64
5	PAY_1	30000 non-null	int64
6	PAY_2	30000 non-null	int64
7	PAY_3	30000 non-null	int64
8	PAY_4	30000 non-null	int64
9	PAY_5	30000 non-null	int64
10	PAY_6	30000 non-null	int64
11	BILL_AMT1	30000 non-null	int64
12	BILL_AMT2	30000 non-null	int64
13	BILL_AMT3	30000 non-null	int64
14	BILL_AMT4	30000 non-null	int64
15	BILL_AMT5	30000 non-null	int64
16	BILL_AMT6	30000 non-null	int64
17	PAY_AMT1	30000 non-null	int64
18	PAY_AMT2	30000 non-null	int64
19	PAY_AMT3	30000 non-null	int64
20	PAY_AMT4	30000 non-null	int64
21	PAY_AMT5	30000 non-null	int64
22	PAY_AMT6	30000 non-null	int64
23	DEFAULT	30000 non-null	int64

Table 1: Info returned by pandas on dataframe cointaining the given dataset

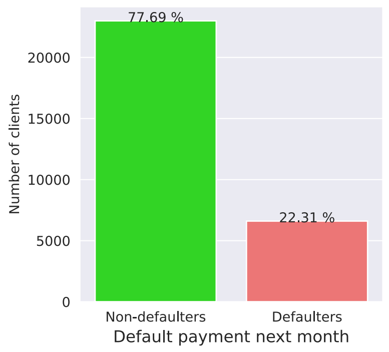


Figure 2: Countplot of `default_payment_next_month`

2.3 Data Structure and cleaning

So looking at the values present in the attributes some changes have to be done:

- Attribute marriage should present only one of those values: 1,2,3; but in the dataset some records have value 0.
- Attribute education should present only one of those values: 1,2,3,4; but in the dataset some records have values 0,5,6.
- Attributes `PAY_N` should present only one of those values: -1,1,2,3,4,5,6,7,8,9; but in the dataset some records have value -2 and 0.

In first two cases since there is an attribute which represent the `Other` class (respectively 3 for marriage and 4 for education), all the attributes not-known are mapped in that category.

attribute	value	count	defaulters	(%)
SEX	Female	17.855	3.744	20,96%
	Male	11.746	2.861	24,35%
EDUCATION	University	14.024	3.329	23,73%
	Graduate school	10.581	2.036	19,24%
	High school	4.873	1.233	25,30%
	Other	123	7	5,70%
MARRIAGE	Single	15.806	3.329	21,06%
	Married	13.477	3.192	23,68%
	Others	318	84	26,4%

Table 2: Value counts for `SEX`, `EDUCATION` and `MARRIAGE` feature

In the last case, in order to use this attributes as a numerical attribute, and not a categorical one, all the values -2 and -1 are mapped in 0. In this way `PAY_N` will indicate for how many months the payment was delayed.

2.4 Categorical features

Regarding the categorical features `SEX`, `EDUCATION` and `MARRIAGE` showed in Figure 3 and counts in Table 2, it is possible to notice that there are much more females than males in the dataset, and, in particular, males have a slightly higher chance to default than females (0.24% vs 0.21%). But in general, whether it is male or female, the proportion of DEFAULTERS and NON-DEFAULTERS per each value, is in line with the categories of the other two features.

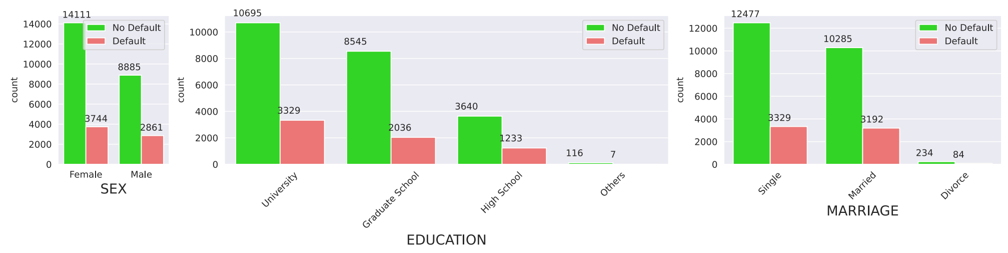


Figure 3: Countplot of `SEX`, `EDUCATION` and `MARRIAGE` grouped by DEFAULT class

We still have to inspect the payment status feature `PAY_N`, boxplots below shown in Figure 4 is very useful. It can be seen that clients who delay payment by one month or less have fewer credit card defaults. I.e. a greater discriminatory power is held the repayment status in September, `PAY_1`, than the repayment status in the other months.

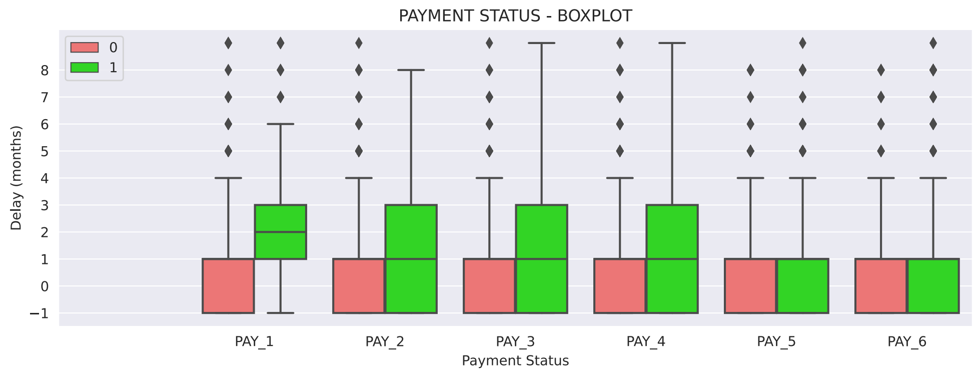


Figure 4: Boxplots of `PAY_N` grouped by DEFAULT class

2.5 Continuous features

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 continuous feature, we explored their KDE plots. Observing Figure 5 it is possible to notice 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 Taiwanese dollars to around \$140000. The customers above this threshold are more likely to repay their debts.

For the feature `AGE`, a similar visual analysis is performed. The probability of non-default of age between approximately 25 and 42 is higher, which indicates that consumers are more capable of repaying credit card loans in this age group. An assumption could be that their work and family tend to be stable without too much pressure.

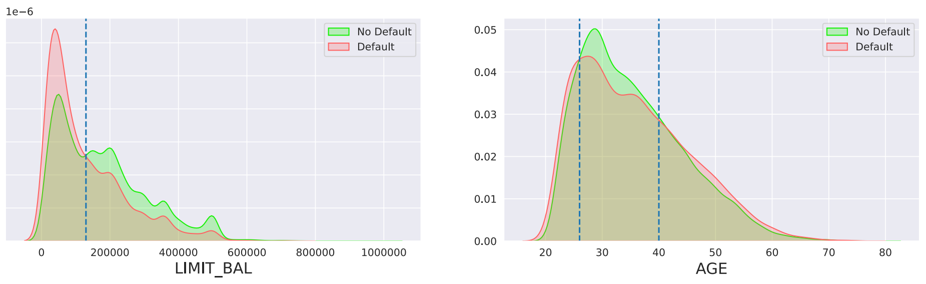


Figure 5: KDE plots of `LIMIT_BAL` and `AGE` grouped by DEFAULT class

2.6 Check for Normality distribution - QQ-plot

Methods we will use later assume that the data should have a known and specific distribution, i.e. Normal distribution. Applying such methods on different data distribution, our final results may be misleading or plain wrong. A way to check whether our data are Normally distributed, we used a graphical method called Quantile-Quantile (QQ) plot that give us a qualitative evaluation. In a QQ-plot, the quantiles of the independent variable are plotted against the expected quantiles of the normal distribution. If the variable is normally distributed, the dots in the QQ-plot should fall along a 45 degree diagonal. The plots show that there is no evidence that numerical features are normally distributed.

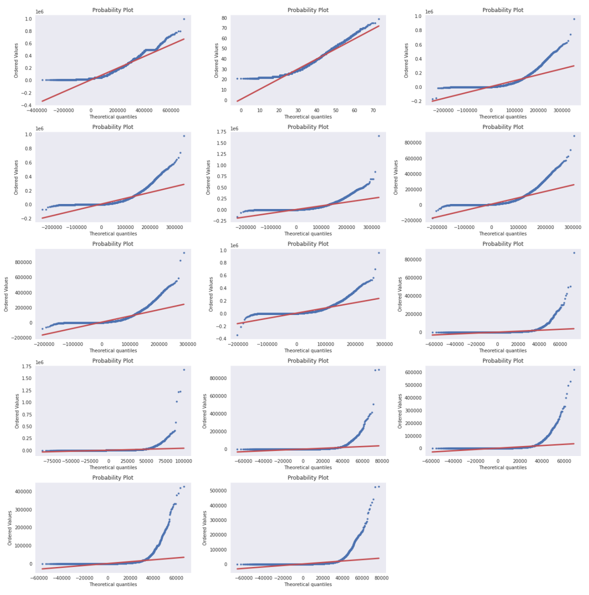


Figure 6: QQ-plot for feature `LIMIT_BAL`, `BILL_ATM`,`AGE` and `PAY_ATM`

2.7 Correlation

Correlation is a statistical term describing the degree to which two random variables move in coordination with one-another. Intuitively, if they are moving in the same direction, then those variables are defined with a positive correlation, or viceversa we define that with a negative correlation. The Pearson Correlation (ρ) is one of the most used linear correlation measures.

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{1}$$

The value of Pearson's Correlation Coefficient range is [-1, 1].

- +1 means that they are strongly correlated.
- 0 means no correlation, the two random variables are statistically independent, but it is not true the opposite, because they may have a non-linear relationship.
- -1 means that there is a negative correlation (inverse proportion).

High values of this correlation coefficient with respect to the target is a synonym of data redundancy, so it could be helpful to drop those columns. In Figure 7 is given a graphical representation of the Person Correlation with a Heatmap, where each cell (i, j) represents the Person Correlation between the random variables X_i and X_j

From Figure 7, as we may think, we can observe an "internal" correlation among the groups of features such as `BILL_ATM`, `PAY_N`. We can also notice that there is no feature with a strong relationship with the target. In fact, there are 15 features with an absolute value of the correlation below than 0.1 and none of the remaining ones have a greater correlation than 0.29.

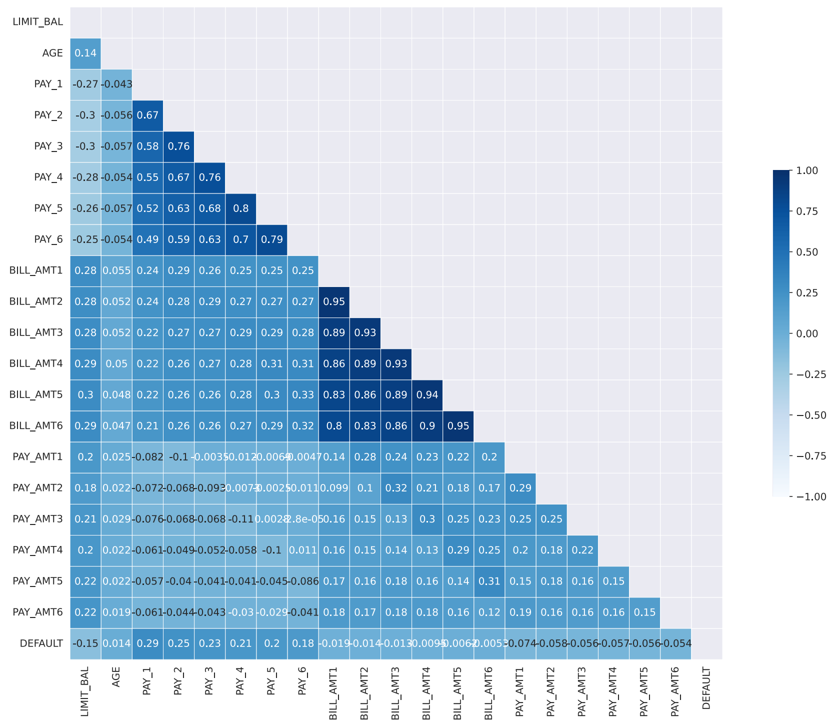


Figure 7: Heatmap correlation

3 Data preprocessing

3.1 Handling Categorical Features

The categorical features `EDUCATION`, `SEX`, and `MARRIAGE` are already encoded with integer numbers and could be fed to a machine learning algorithm. However, these are nominal features, for which it would be sub-optimal to assume an ordering. *One-hot encoding* allows us to remove any ordinal relationship, which would be meaningless between these categorical variables. The idea behind this approach is to create a new dummy feature for each unique value in the nominal feature column. Binary values can then be used to indicate the particular class of an example.

Although Scikit-Learn provides methods to perform one-hot encoding automatically, we decide to do the mapping of the features by hand, since there are few. In this way we mitigate the problem of *multicollinearity*, which occurs when there are highly correlated features. Thus, we create the following boolean columns and drop the old ones, `EDUCATION`, `SEX`, and `MARRIAGE`.

- `MALE`: 1 = male; 0 = female.
- `MARRIED`: 1 = married marital status; 0 = otherwise.
- `GRAD_SCHOOL`: 1 = graduate school level of education; 0 = otherwise.
- `UNIVERSITY`: 1 = university level of education; 0 = otherwise.
- `HIGH_SCHOOL`: 1 = high school level of education; 0 = otherwise.

Using this strategy we do not loose any information. In Table 3 and Table 4 there is an example that shows variable respectively before and after the application of one-hot encoding.

id	`SEX`	`MARRIAGE`	`EDUCATION`
0	1	1	1
1	2	2	2
2	1	3	3
3	2	1	4

Table 3: `SEX`, `MARRIAGE` and `EDUCATION` features before one-hot encoding

id	`MALE`	`MARRIED`	`SINGLE`	`GRAD_SCHOOL`	`UNIVERSITY`	`HIGH_SCHOOL`
0	True	True	False	True	False	False
1	False	False	True	False	True	False
2	True	False	False	False	False	True
3	False	True	False	False	False	False

Table 4: `SEX`, `MARRIAGE` and `EDUCATION` features after one-hot encoding

3.2 Dataset Partition

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

Another data splitting is necessary, in order to tune the hyperparameters. In particular, two new partitions will be considered: train set and validation set. It has been used a Stratified K-Fold Cross Validation (with k = 5). Once obtained the best hyperparameters, train again the model with both train and evaluation set and finally evaluate it on the test set.

- Training set:
  - shape: (22200,25)

  - defaulters: 4955
  - non-defaulters: 17245
  - class proportion: 22,30%

- Test set:
  - shape: (7400,25)
  - defaulters: 1650
  - non-defaulters: 5750
  - class proportion: 22,30%

3.3 Outliers and Anomaly detection

An outlier or an anomaly detection is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism. Some of the most common causes of outliers are:

- 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 significant amount of outliers in some cases could drastically affect the performances. Therefore, it is a common practice to train the model with and without them in order to catch their contribution. 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).

Figure 8 shows the boxplots for `BILL_ATM` and `PAY_ATM` variables, and are each followed by a description of how outliers were determined for the variables. The `BILL_ATM` variables were depicted the amount of bill statement during the respective months. Since these can actually be considered as repeated observations of the same variable (per cardholder), then for the purpose of outlier identification, they were analyzed using the same boxplot. In addition, and to minimize the loss of information due to elimination of outliers, a cut-off value was set for these variables based on the general trend observed in the six variables. Consequently, it was assumed that for all `BILL_ATM` variables, any amount exceeding 1,000,000 and any amount going below *min*(`BILL_ATM4`), which was the value 170,000, was considered to be an outlier.

For the `PAY_ATM` variables, the same approach was takes for outlier detection as the one taken for `BILL_ATM` variables. The general trend of the six variables set the upper cut-off point at *max*(`PAY_ATM4`) which was the value 621,000. Any observation exceeding this amount was considered an outlier. On the lower side, no observation had a value lower than 0, hence there were no outliers based on this (taking 0 as the minimum). After the exclusion of outlier values was done, a sample size of 29,993 remained (7 observations were dropped), which was the data used for training and validating the model [4].

We do not eliminate any sample as an outlier because the literature on the dataset does not provide information on this [5], and we lack knowledge about the domain.

3.4 Features Scaling

The majority of machine learning and optimization algorithms behave much better if numerical features are on the same scale. Decision trees and random forests are two of the very few machine learning algorithms where there is no need to worry about feature scaling, as they are scale invariant [6].

There are two common approaches to bringing different features onto the same scale:

- *Normalization* refers to the rescaling of the features to a range of [0, 1], in this case we use min-max scaling:
$$X_{Norm} = \frac{X - X_{Min}}{X_{Max} - X_{Min}} \tag{2}$$

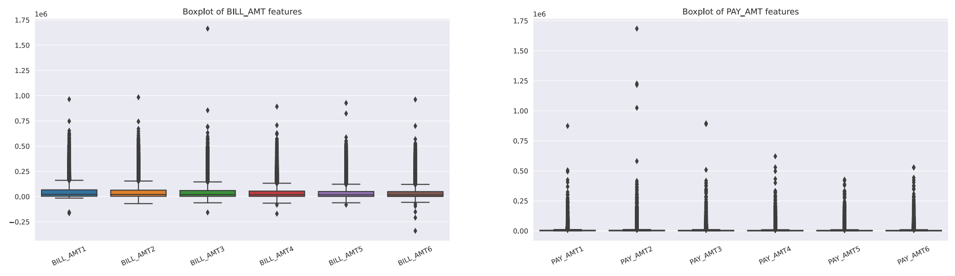


Figure 8: Box-plot for `PAY_ATM` and `BILL_ATM` Variables

where X is a particular example, X_{Min} is the smallest value in a feature column, and X_{Max} is the largest value.

- *Standardization or Z-Score Normalization*: center the feature columns at mean $\mu = 0$, with standard deviation $\sigma = 1$, so that the feature columns have the same parameters as a standard normal distribution, which makes it easier to learn the weights:

$$Z = \frac{X - \mu}{\sigma} \tag{3}$$

While normalization suppress the effect of outliers, standardization maintains information about them. Since the technique we will apply shortly is sensitive to the presence of outliers, we decide to continue our study with the normalized data.

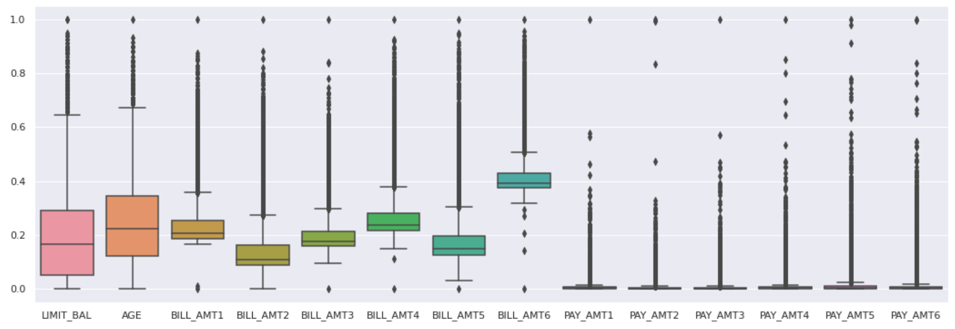


Figure 9: Box-plot scaled features

3.5 Dimensionality reduction

As we mentioned previously, many algorithms would benefit from the removal of strongly correlated features, and in general from having a lower dimensionality of the data. The presence of many features in the dataset leads to excessively complex models, which fit the parameters too closely with regard to the particular observations in the training set, but do not generalize well to new data. We say that these models have a high variance and are overfitting the training set.

In fact, high dimensional data can overfit the training set. The problem of overfitting become more serious i.e. in Nearest Neighbor classifiers, where the course of dimensionality problem is faced. The curse of dimensionality depicts the phenomenon where the feature space becomes progressively sparse for an increasing number of dimensions of a fixed-size training dataset. We can think of even the nearest neighbors as being excessively far away in a high-dimensional space to give a good estimate.

The dimensionality reduction can be carried out by means of:

- *feature selection*: we select a subset of the original features

- *feature extraction*: we construct a new feature subspace deriving information from the original feature set.

As previously noticed, `BILL_ATM` categories are highly correlated between them, so a manual discard could be done. However, we decide to keep them to perform *Principal Component Analysis*, a feature extraction technique.

3.5.1 PCA - Principal component analysis

In PCA, both the compression and the recovery, are performed by linear transformations. Let $x_1, ...x_m$, be m vectors in R_d. We would like to reduce the dimensionality of these vectors using a linear transformation.

The idea is to find a matrix $W \in R^{n,d}$, where $n < d$, that induces a mapping

$$x \mapsto W_x, \quad x \in R^d, W_x \in R^n; \quad (n < d) \tag{4}$$

and a second matrix $U \in R^{n,d}$, that can be used to recover each original vector x from its compressed version. That is, for a compressed vector $y = Wx$, where y is in the low dimensional space R^n, we can construct $\tilde{x} = Uy$, so that $\tilde{x}$ is the recovered version of x and resides in the original high dimensional space R^d.

The objective of PCA is to find the compression matrix W and the recovering matrix U so that the total squared distance between original and recovered vector is minimal in the least squared sense:

$$\underset{W \in R^{n,d}, U \in R^{d,n}}{argmin} \sum_{i=1}^{m} \|\mathbf{x}_i - UW\mathbf{x}_i\|_2^2 \tag{5}$$

The optimal solution (U, W) of this problem takes the form:

- the columns of U are orthonormal: $U^\top U = I_n \in R$;
- $W = W^\top$

Hence, we can write:

$$\underset{U \in R^{d,n}: U^\top U = I_n}{argmin} \sum_{i=1}^{m} \left\|\mathbf{x}_i - U^\top U\mathbf{x}_i\right\|_2^2 \tag{6}$$

It can be proven that:

$$\left\|\mathbf{x} - U^\top U\mathbf{x}\right\|^2 = \|\mathbf{x}\|^2 - tr(U^\top \mathbf{x}\mathbf{x}^\top U) \tag{7}$$

where the trace tr of a matrix is the sum of its diagonal entries, i.e., a linear operator. This allows us to rewrite Equation (6) as the following maximization problem:

$$\underset{U \in R^{d,n}: U^\top U = I_n}{argmax} \quad tr(U^\top (\sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}^\top) U) \tag{8}$$

Now, we can define the scatter matrix $A = \sum_{i=0}^{m} x_i x_i^\top$, since this matrix is symmetric, it can be written using its spectral decomposition $A = VDV^\top$, where D is diagonal and $V^\top V = VV^\top = I$. Here, the elements on the diagonal of D are the eigenvalues of A, while the columns of V are the corresponding eigenvectors. In particular, we can safely assume that the diagonal elements of D are sorted by the largest, and are all positive because A is semi-definite positive:

$$D_{1,1} \geq D_{2,2} \geq ... \geq D_{d,d} \geq 0 \tag{9}$$

From these premises, we can claim that the solution of the optimization problem (6) is the matrix U, whose columns $\mathbf{u}_1, ..., \mathbf{u}_n$ are the n eigenvectors of the matrix A corresponding to the largest n eigenvalues, while $W = U^\top$.

So to sum up, PCA helps us to identify patterns in the data based on the correlation between features, finding the directions of maximum variance in high-dimensional data and projecting it onto a new subspace with equal or fewer dimensions than the original one.

The orthogonal axes (PC = principal components) of the new subspace can be interpreted as the directions of maximum variance given the constraint that the new feature axes are orthogonal to each other, as illustrated in the Figure 10. Here, x_1 and x_2 are the original feature axes, and PC_1 and PC_2 are the principal components [7].

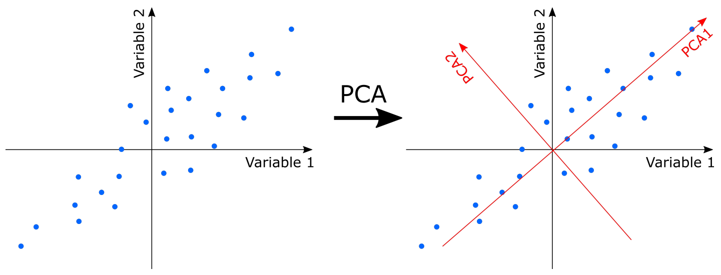


Figure 10: PCA graphical explanation

Since a reduction of the dimensionality of our dataset can be done by compressing it onto a new feature subspace, the number of principal components chosen will be that preserve at the most the information carried by the data. A decision can be taken looking at the explained variance ratio, that will be how much variance does contain each principal component out of the d components we obtain applying PCA to our training dataset.

Results depicted in Figure 11 show that the first 6 principal components capture more than 90% of the total variance. However, considering the first 12 principal components the variance is capable to explain 99% of the total variance, even if the number of features is halved.

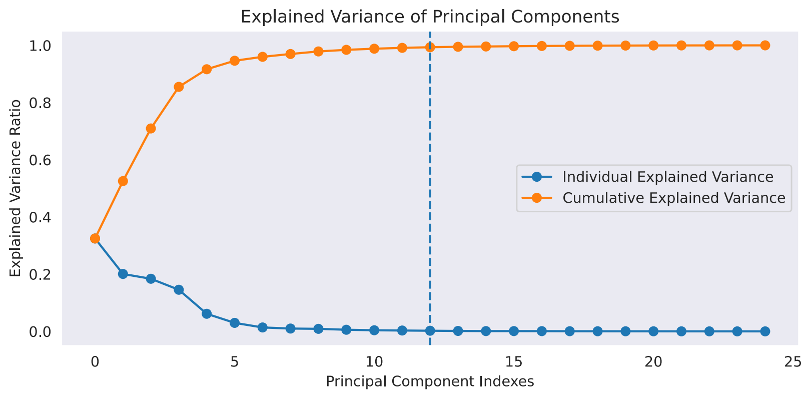


Figure 11: Cumulative and individual explained variance plotted against principal components

3.6 Class imbalance - Resampling

Class imbalance is quite a common problem when working with real-world data, it consists of having much more samples that refers to one class than other classes in the dataset. We showed in Section 2.2 and reported in Figure 2 that the dataset we are dealing with is unbalanced, as the non-defaulter examples are over-represented.

In our case, we could achieve almost 80% accuracy by just predicting the majority class (non-defaulters) for all examples, without the help of a supervised machine learning algorithm. Thus, when we will fit classifiers on our datasets, it would make sense to focus on other metrics than accuracy when comparing different models.

Aside from evaluating machine learning models, class imbalance influences a learning algorithm to model fitting itself. Since machine learning algorithms typically optimize cost function that is computed as a sum over the training examples that it sees during fitting. The decision rule is likely going to be biased toward the majority class.

The option of collecting more data is excluded a priori. There are other option to tackle this problem:

- At training time assign larger penalty to wrong predictions on the minority class.
- Upsampling the minority class
- Downsampling the majority class

Unfortunately, there is not a generalized best solution or technique that works best across different problem domains. Thus, in practice, it is recommended to try out different strategies on a given problem, evaluate the results, and choose the technique that seems most appropriate.

We decide to exclude the naïve methods of oversampling (undersampling), since they randomly duplicate (delete) data from the minority (majority) class, until the desired level is obtained. On the one hand, random undersampling does not allow controlling which information is discarded, on the other hand random oversampling causes overfitting because the model is trained on many identical data. We decide to explore two technique which mitigate exposed issues that are the Cluster Centroid (Undersampling) method and the Synthetic Minority Oversampling Technique (SMOTE).

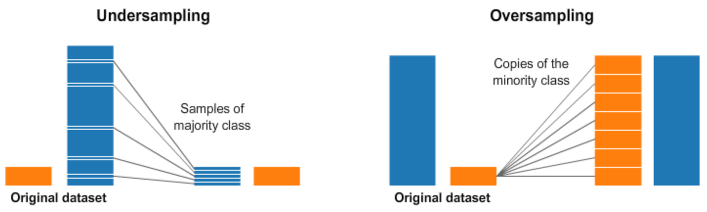


Figure 12: Resampling techniques

3.6.1 Cluster Centroid Undersampling

Minority class samples in our dataset are enough to allow us to perform undersampling. However, one major problem of using undersampling is that important information may be lost from the majority class, which can cause overly general rules. This cannot be afforded to develop the credit card default prediction model, especially for default samples. Hence, to overcome this problem, the Cluster Centroids method has been introduced in [8].

The idea of the Cluster Centroids method is to replace clusters of majority samples with the respective cluster centroids. A K-means algorithm is fitted to the data, and the number of clusters is set equal to the number of samples of the minority class. Then, the majority of samples from the clusters are entirely substituted by the sets of cluster centroids from K-means [9]. Cluster Centroids contain the most representative variations of the majority class in which features values would be visualized at the center.

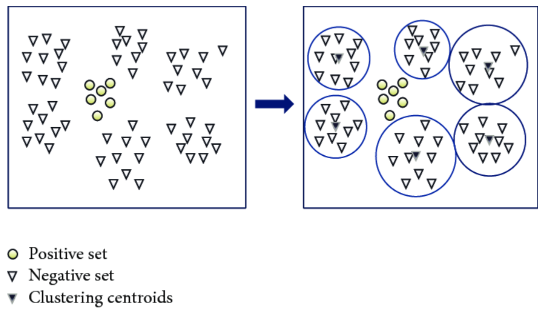


Figure 13: Cluster centroids undersampling [10]

3.6.2 SMOTE - Synthetic Minority Oversampling Technique

The Synthetic Minority Oversampling Technique (SMOTE) was proposed in [11] to avoid the risk of overfitting faced by random oversampling. Instead of simply replicating existing observations, the technique generates artificial samples. As shown in Figure 14, this is achieved by linearly interpolating a randomly selected minority observation and one of its neighboring minority observations. More precisely, SMOTE executes three steps to generate a synthetic sample:

- Firstly, it chooses a random minority observation $\vec{a}$
- Among the nearest minority of $\vec{a}$ neighbors select the instance $\vec{b}$
- Create a new sample $\vec{x}$ by linearly interpolating $\vec{a}$ and $\vec{b}$ (with ω random)

$$\vec{x} = \vec{a} + \omega \times (\vec{b} - \vec{a}), \quad \omega \in [0, 1] \tag{10}$$

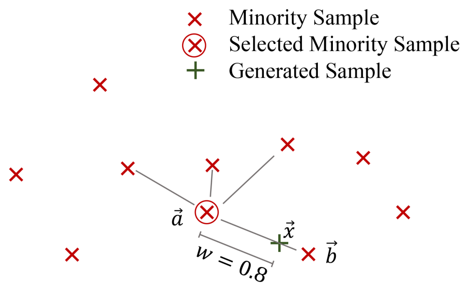


Figure 14: SMOTE linearly interpolates a randomly selected minority sample and one of its k = 4 nearest neighbors [12]

Unfortunately, SMOTE, as other random oversampling techniques, suffers from weakness due to within-class imbalance and noise. In fact, SMOTE choose a random sample from the minority class to start, but if distribution of the minority class is not uniformed this can cause densely populated areas to be further inflated with artificial data. Moreover, SMOTE does not recognize noisy minority samples which are located among majority class instances and interpolate them with their minority neighboring, may be creating new noisy entries.

Finally, it has been proven that classification algorithms could benefit from samples that are closer to class boundaries, and SMOTE does not specifically work in this sense, as shown in Figure 15.

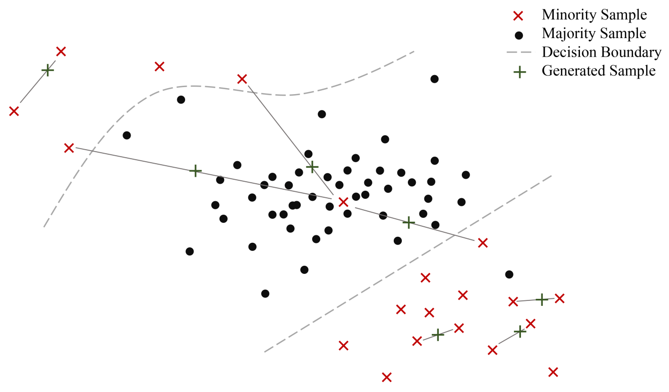


Figure 15: SMOTE behavior in presence of noise and with-in class imbalance [12]

3.6.3 K-means SMOTE

As showed in [12] the method employs the simpler and popular K-means clustering algorithm in conjunction with SMOTE oversampling in order to rebalance datasets. It manages to avoid the generation of noise by oversampling only in safe areas (i.e., areas made up of at least 50% of minority samples). Moreover, its focus is placed on both between-class imbalance and within-class imbalance, fighting the small disjuncts' problem by inflating sparse minority areas. K-means SMOTE executes three steps as show in Figure 16: clustering, filtering, and oversampling.

- Clustering: the input space is clustered into k groups using K-means clustering.
- Filtering: selects clusters for oversampling, retaining those with a high proportion of minority class samples. It then distributes the number of synthetic samples to generate, assigning more samples to clusters where minority samples are sparsely distributed.
- Oversampling: SMOTE is applied in each selected cluster to achieve the target ratio of minority and majority instances.

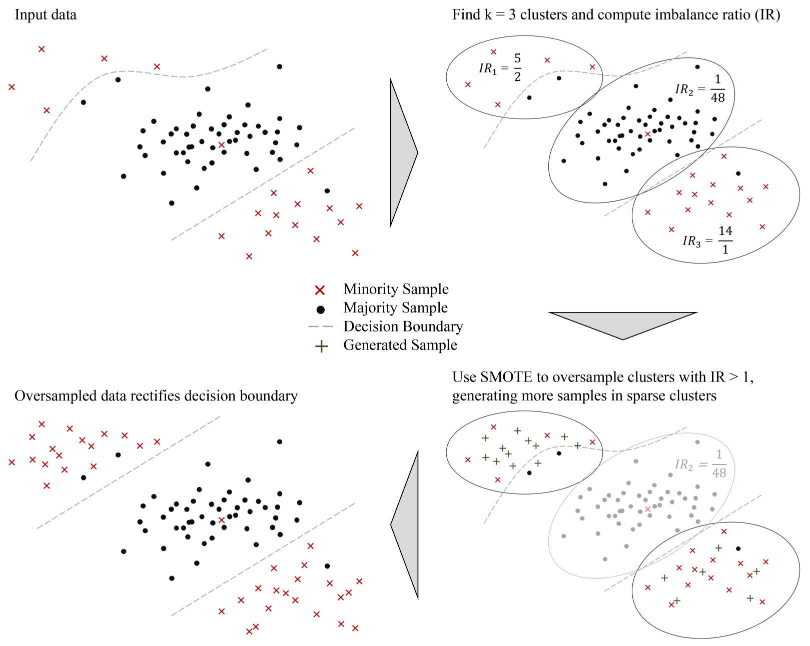

Input data
Find k = 3 clusters and compute imbalance ratio (IR)
$IR_1 = \frac{5}{2}$
$IR_2 = \frac{1}{48}$
$IR_3 = \frac{14}{1}$
× Minority Sample
● Majority Sample
Decision Boundary
+ Generated Sample
Use SMOTE to oversample clusters with IR > 1, generating more samples in sparse clusters
Oversampled data rectifies decision boundary

Figure 16: K-means SMOTE over samples safe areas and combats within-class imbalance

4 Model evaluation

One of the main steps in building a machine learning model is to estimate its performance on data that the model has not seen before. For this reason, in 3.2 initial dataset is splitted into separate training and test datasets. The former will be used for model training, and the latter to estimate its generalization performance. This approach is commonly known as holdout method.

However, in typical machine learning applications, it is interested in tuning and comparing different parameter settings to further improve the performance for making predictions on unseen data. But, if we reuse the same test dataset over and over again during this process, it will become part of our training data and thus the model will be more likely to overfit. A validation set could be held out of the training set, to evaluate on it the performance of the model. However, this is not recommended because performance estimation may be sensitive to how we partition the training set.

4.1 K-fold cross validation

Instead of holdout, a more robust technique for model tuning is k-fold cross-validation, where we repeat the holdout method *k* times on *k* subsets of the training data. In practice, we randomly split the training dataset into *k* folds without replacement, where *k-1* folds are used for the model training, and one fold is used for performance evaluation. We then calculate the average performance of the models based on the different independent test folds to obtain a performance estimate that is less sensitive to the sub-partitioning of the training data compared to the holdout method. Once we have found satisfactory hyperparameter values, we can retrain the model on the complete training dataset and obtain a final performance estimate using the independent test dataset. The rationale behind fitting a model to the whole training dataset after k-fold cross-validation is that providing more training examples to a learning algorithm usually results in a more accurate and robust model.

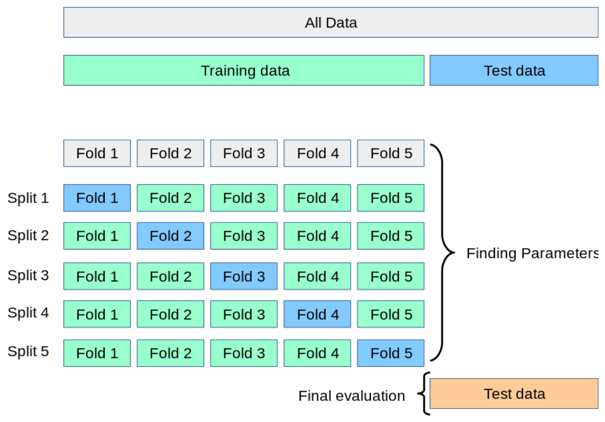


Figure 17: K-fold cross validation (i.e.: k = 5)

Since k-fold cross validation is a resampling technique without replacement, its advantage is that each sample of the training set will be used exactly once for validation purpose, and this yields to a lower-variance estimate of the model performances than the holdout method. A common value for k in k-fold cross validation is 10 [13]. However, we are dealing with a large dataset and choosing such a value would be too costly in computational sense, for this reason we decide to use a smaller value for k, i.e. $k = 5$, still taking advantage of the k-fold method.

4.2 Performance evaluation metrics

Classification evaluation metrics compare the predicted class from the classifier with the actual class of the samples. At this scope it is useful to build a confusion matrix where the *(j, k)-th* element counts the number of times that the actual class is j whereas the predicted class is k. In the binary case (such as our case) the confusion matrix become simpler to build and to interpret. If the two classes to be predicted are True and False the confusion matrix is the one depicted in Table 5.

		Actual Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Table 5: Binary classification confusion matrix

The following terminology is used when referring to the counts tabulated in a confusion matrix:

- *True positive (TP)*, which corresponds to the number of positive examples correctly predicted by the classification model.
- *False negative (FN)*, which corresponds to the number of positive examples wrongly predicted as negative by the classification model.
- *False positive (FP)*, which corresponds to the number of negative examples wrongly predicted as positive by the classification model.
- *True negative (TN)*, which corresponds to the number of negative examples correctly predicted by the classification model.

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:

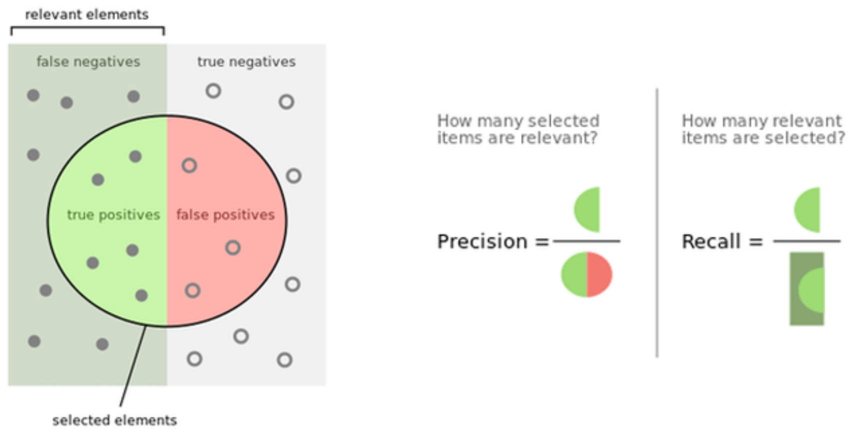


Figure 18: Precision and Recall

- *Accuracy*: is the performance measure generally associated with machine learning algorithms. It is the ratio of correct predictions over the total number of data points classified.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{11}$$

- *Precision*: (also called *positive predictive* value): Indicates how many of a j-object (in binary classification, commonly True class is considered) predictions are correct. It is defined as the ratio of correct positive predictions over all the positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

- *Recall*: (also called *sensitivity*): Indicates how many of the j-object (in binary classification, commonly True class is considered) samples are correctly classified. It is defined as the fraction of j-object predictions over the total number of j-object samples.

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

- *F1*: Accuracy measure treats every class as equally important, for this reason it may be not suitable for imbalanced datasets, where the rare class is considered more interesting than the majority class [14].

  Hence, Precision and Recall, that are class-specific metrics, are widely employed in applications in which successful detection of the rare class is more significant than detection of the other.

  The challenge is to build a model that is capable of maximize both Precision and Recall. Hence, the two metrics are usually summarized in a new metric that is F1-score. In practice, F1-score represents the harmonic mean between precision and recall, so, a high value ensures that both are reasonably high.

$$F1_{score} = \frac{2}{\frac{1}{r} + \frac{1}{p}} = \frac{2rp}{r + p}, \quad where\ r = Recall, p = Precision \tag{14}$$

5 Classification models

In this section, we present different supervised learning algorithms with their mathematical details, and we use them on our dataset to build a classification model that is able to predict credit card defaults in the next month. In particular we will dive into Support Vector Machine, Logistic Regression and some tree based methods, all following the Empirical Risk Minimization paradigm.

5.1 Logistic Regression

Logistic Regression models are predictors of the family of *Generalized Linear Models*(GLM). GLM are a broad class of models that provide a unifying framework for many commonly used statistical techniques, such as linear regression.

GLM are characterized by three components:

- the *random component* which identifies the response variable Y and assumes a probability distribution for it, treating the m *observations* on Y , denoted: $(y_1, ..., y_m)$, as independent;
- the linear predictor that specifies the explanatory variables through a prediction equation that has linear form, such as:
$$\alpha + \beta_1 x_1 + ... + \beta_p x_p = \mathbf{x}^\top \beta; \tag{15}$$
- the link function which specifies a function g that relates $E[Y]$ to the linear predictor such as [15]:
$$g(E[Y]) = \alpha + \beta_1 x_1 + ... + \beta_p x_p \tag{16}$$

Hence, in a GLM the expected response for a given feature vector $\mathbf{x} = [x_1, ..., x_n]$ is of the form:

$$E[Y|X = \mathbf{x}] = h(\mathbf{x}^\top \beta) \tag{17}$$

with h, called *activation function*, being the inverse of the link function g [16].

Rather than directly modeling the distribution of Y, the logistic regression models the probability that Y belongs to a particular class using the logistic function as activation function h:

$$P(Y_i = 1|X = x_i) = h(\mathbf{x}^\top \beta) = \frac{e^{x_i^\top \beta}}{1 + e^{x_i^\top \beta}} \tag{18}$$

indeed, with $x_i^\top \beta$ large we will have a high probability for Y to be 1, and for small $x_i^\top \beta$ we will have a high probability for Y to be 0. The logistic function will always produce an S-shaped curve between 0 and 1, as shown in Figure 19.

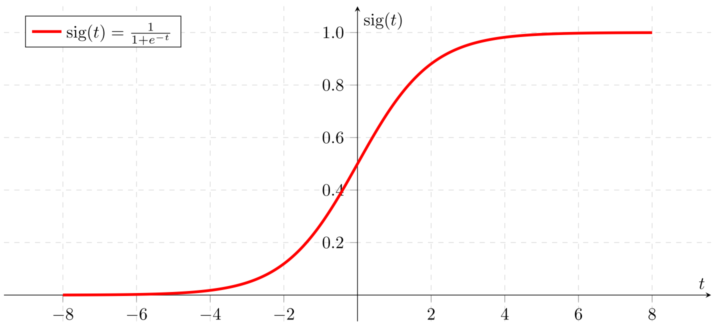


Figure 19: Sigmoid activation function

To estimate the coefficients vector β through the available training data we use the Maximum Likelihood method, which finds $\hat{\beta}$ that is the maximum likelihood estimate of β, this is formalized in such a way:

$$\mathcal{L}(\beta) = \prod_{i=1}^{m} [h(\mathbf{x}_i^\top \beta)]^{y_i} [1 - h(\mathbf{x}_i^\top \beta)]^{1-y_i} \tag{19}$$

where $\mathcal{L}(\beta)$ is the log-likelihood, which maximized with respect to β gives the maximum likelihood estimator of β. In a supervised learning environment, that maximization is equivalent to minimizing the function:

$$-\frac{1}{m} \log \mathcal{L}(\beta) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log h(\mathbf{x}_i^\top \beta) + (1 - y_i) \log 1 - h(\mathbf{x}_i^\top \beta)] \tag{20}$$

We can interpret this function as the *binary cross-entropy* training loss associated with comparing a true conditional probability density function (pdf) with an approximated pdf. In this study, the Logistic Regression model provided by the `SciKit-learn` python library, which by default applies l_2 regularization. Applying a regularization term is useful in reducing the generalization error but not its training error, preventing overfitting on the training set.

The regularization term is added to the objective function, and in practice it penalizes large weights values during the training, in other words regularization term force the weights to go toward 0, and particularly the l_2 regularization does not make them to be 0 (while l_1 does) and it is defined as:

$$l_2(\beta, \lambda) = \frac{\lambda}{2} \sum_{i=1}^{m} \beta_i^2 \tag{21}$$

where λ called *regularization parameter* help us to tune the regularization strength.

5.2 Decision Tree and Random Forest

Tree-based methods provide a simple, intuitive, and powerful mechanism for both regression and classification. The main idea is to stratify a (potentially complicated) feature space into smaller regions and fit a simple prediction function to each region [17]. In order to classify a given observation, we typically use the mode response value for the training observations in the region to which it belongs. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision tree* methods [17].

5.2.1 Decision Tree

Decision tree classifiers are attractive models if we care about interpretability. They involve creating a set of *binary splits* on the predictor variables in order to create a tree that can be used to classify new observations into one of two groups [18].

Using the decision algorithm, we start at the tree root and split the data on the feature that results in the largest information gain (IG):

- if the predictor is continuous, we choose a cut-point that maximizes purity for the two groups created;
- if the predictor variable is categorical, we combine the categories to obtain two groups with maximum purity.

In an iterative process, we can then repeat this splitting procedure at each child node until the leaves are pure. This means that the training examples at each node all belong to the same class. Unfortunately, this process tends to produce a tree that is too large and suffers from overfitting. Thus, we typically prune the tree by setting a limit for the maximal depth of the tree.

Gini impurity (I_G) and entropy (I_H) are the most commonly used splitting criteria in binary decision trees. Definying as $p(i|t)$ the proportion of the examples that belong to class i for a particular node t, we can write the entropy as:

- Gini Impurity: $I_G(t) = \sum_{i=1}^{c} p(i|t)(1 - p(i|t))$
- Entropy: $-\sum_{i=1}^{c} p(i|t) \log_2 p(i|t)$

However, even though a decision tree is fairly interpretable, it is typically less accurate and robust compared to more sophisticated algorithms.

A more advanced tree based algorithm is *Random Forest*.

5.2.2 Random Forest

Random Forest is an ensemble method which reduces the variance of a single model by combining multiple decision trees with the *bagging* technique. The idea behind the bagging or *bootstrap aggregating* technique is to generate different *bootstrapped* training sets taking sample with repetition form the dataset. An illustration in Figure 20 skim the concept.

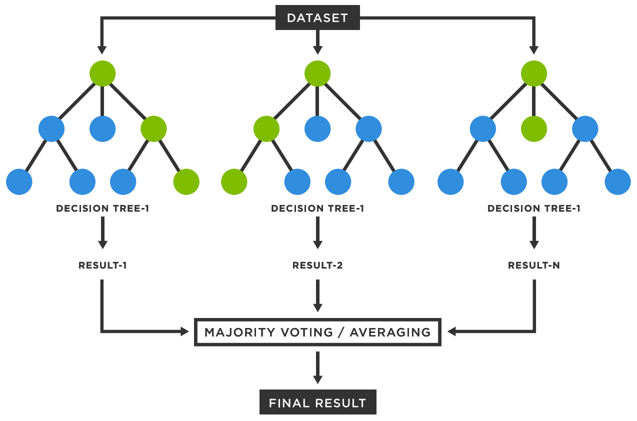


Figure 20: Feature importance obtained training Random Forest on the raw training dataset

Random Forest uses the bagging also at each split for the feature selection to *decorrelate* the trees: to generate each splitting rule it is considered a randomly selected subset of features of fixed size.

This is why this algorithm is fairly robust to noise and outliers and will have much less variance rather than a single decision tree. Random forest models are not interpretable as Decision trees, but they allow us to measure the importance of each feature. As we will see in the results and in Figure 21 from our analysis on the raw data, it emerges that repayment status, age and bill amount are very important in the decision process.

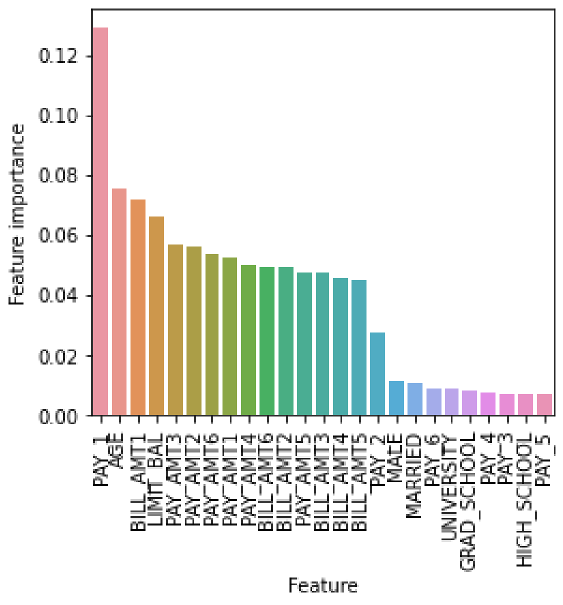


Figure 21: Feature importance obtained training Random Forest on the raw training dataset

5.3 Support Vector Machine

Support Vector Machines (SVMs) are considered among the most effective classification algorithms in modern machine learning [19]. When used for classification tasks, SVMs are supervised learning methods that construct an hyperplane that maximizes the margin between two classes in the feature space.

5.3.1 Hard margin SVM

A hyperplane in a space $\mathcal{H}$ endowed with a dot product $\langle \cdot, \cdot \rangle$ is described by the set:

$$\{\mathbf{x} \in \mathcal{H} |\, \langle \mathbf{w},\mathbf{x} \rangle + b = 0\} \tag{22}$$

where $\mathbf{w} \in \mathcal{H}$ and $b \in R$

Such a hyperplane naturally divides $\mathcal{H}$ into two half-spaces and hence can be used as the decision boundary of a binary classifier: $\{\mathbf{x} \in \mathcal{H} |\, \langle \mathbf{w},\mathbf{x} \rangle + b \geq 0\}$ and $\{\mathbf{x} \in \mathcal{H} |\, \langle \mathbf{w},\mathbf{x} \rangle + b \leq 0\}$

Given a set $\mathcal{X} = [\mathbf{x}_1, ..., \mathbf{x}_m]$, the margin is the distance of the closest point in $\mathcal{X}$ to the hyperplane:

$$\underset{i=1,...,m}{min} \frac{|\, \langle \mathbf{w}, \mathbf{x}_i \rangle \,|}{\|\mathbf{w}\|} \tag{23}$$

Since the parametrization of the hyperplane is not unique, we set

$$\underset{i=1,...,m}{min} |\, \langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1 \tag{24}$$

and the margin simply becomes $\frac{1}{\|\mathbf{w}\|}$

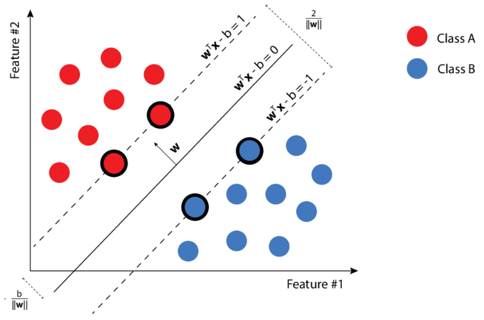

$\frac{2}{\|\mathbf{w}\|}$
Class A
Class B
Feature #2
$\mathbf{w}^T\mathbf{x}-b=1$
$\mathbf{w}^T\mathbf{x}-b=0$
$\mathbf{w}^T\mathbf{x}-b=-1$
$\mathbf{w}$
Feature #1
$\frac{b}{\|\mathbf{w}\|}$

Figure 22: Hard Support Vector Machines

Let $S = [(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)]$ be a training set of examples, where each $\mathbf{x}_i \in \mathcal{H}$ and $y_i \in \{\pm 1\}$. Our aim is to find a linear decision boundary parameterized by $(\mathbf{w}, b)$ such that $\langle \mathbf{w}, \mathbf{x}_i \rangle + b \geq 0$ whenever $y_i = +1$ and $\langle \mathbf{w}, \mathbf{x}_i \rangle + b < 0$ whenever $y_i = -1$. The SVM solution is the separating hyperplane with the maximum geometric margin, as it is the safest choice. The problem of maximizing the margin can be written as:

$$\begin{aligned} &\underset{\mathbf{w},b,\xi}{min} \;\; \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad &y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \quad \forall i \end{aligned} \tag{25}$$

5.3.2 Soft margin SVM

In the hard margin formulation, as mentioned, to find a solution to the maximization problem we need the strong assumption that data are linearly separable. In the case in which this assumption does not hold, the Soft SVM formulation is useful. That formulation allows the violation of the constraint for some samples in the training set. Soft margin SVM is formulated introducing non-negative slack variables $\xi_1, ..., \xi_m$ which measure how much the constraint $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ is being violated. Now the problem is to jointly minimize the norm of $\mathbf{w}$ (the margin) and the average of all ξ_i (the average violations to the constraint):

$$\begin{aligned} &\underset{\mathbf{w},b,\xi}{min} \;\; \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{m}\sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad &y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \\ &\xi_i \geq 0, \forall i \end{aligned} \tag{26}$$

where C is a penalty parameter, typically determined via k-fold cross validation. A better looking at the problem reveals that [20]:

- $\xi_i = 0$ whenever $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$
- $\xi_i = 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ whenever $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \leq 1$

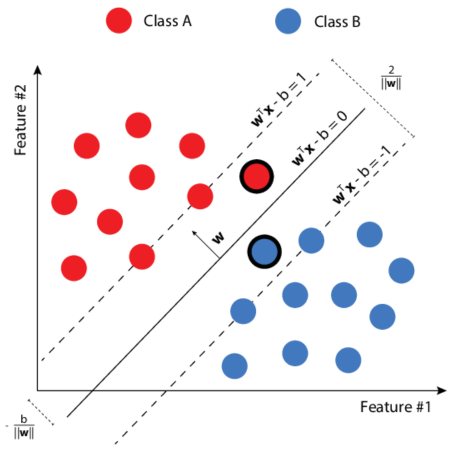

Class A
Class B
Feature #2
Feature #1
$\mathbf{w}^T\mathbf{x}-b=1$
$\mathbf{w}^T\mathbf{x}-b=0$
$\mathbf{w}^T\mathbf{x}-b=-1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Figure 23: Soft Support Vector Machines

Let the hinge loss in the context of half spaces learning be:

$$l^{hinge}(\mathbf{w}, (x, y) = max\{0, 1 - y\langle \mathbf{w}, x \rangle\} \tag{27}$$

we have that $\xi_i = l^{hinge}((\mathbf{w}, b), (x_i, y_i))$ and so we can reformulate the Problem in (18) as the unconstrained problem:

$$\underset{\mathbf{w},b}{min} \; \frac{1}{2}\|\mathbf{w}\|^2 + L_S^{hinge}(\mathbf{w}, b) \tag{28}$$

5.3.3 Kernel SVM - kernal 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 not be enough. In these cases, it is possible to map the data into a higher dimensional space such that it will be linearly separable.

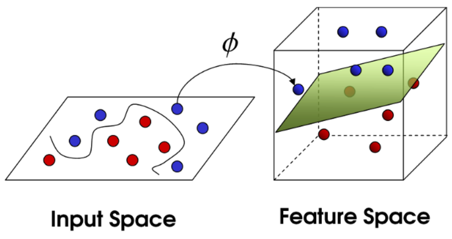


Figure 24: Kernel trick

Let us consider now the dual problem of (26) formulated as:

$$\underset{\alpha \in R; \alpha \geq 0}{max} (\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle) \tag{29}$$

It is evident that the dual problem only involves inner products between instances, that is nothing but a linear kernel, so there is no restriction of using new kernel functions with the aim of measuring the similarity in higher dimensions.

Given a non-linear mapping $\psi : X \mapsto F$ the kernel function is defined as:

$$K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle \tag{30}$$

the dual problem in (29) becomes:

$$\max_{\alpha \in R; \alpha \geq 0} (\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle) \tag{31}$$

the kernel enables an implicit non-linear mapping of the input points to a high-dimensional space where large-margin separation is sought.

The complexity now depends on the size of the dataset, because for M data points we need to compute $\binom{M}{2}$ inner products. Kernels explored in our analysis are:

- polynomial kernel: $K(\mathbf{x}, \mathbf{x}') = (\gamma \langle \mathbf{x}, \mathbf{x}' \rangle)^d$
- RBF kernel: $K(\mathbf{x}, \mathbf{x}') = e^{-(\gamma ||\mathbf{x} - \mathbf{x}'||^2)}$

6 Results

In the following pages, an overview on the results is given for each classifier. Different preprocessing combinations were tested: applying dimensionality reduction techniques (PCA) or not, using or not different resampling techniques. The metric we choose to adopt is F1-score. Precision-recall curve and Confusion Matrix of the best model have been provided for each model.

6.1 Logistic regression

The *Logistic Regression* class in *Scikit-learn* implements the parameter C as the inverse of the regularization parameter λ. In the Table 6 are shown combinations of different values of C with different choices about the preprocessing phase have been explored through a Cross-Validated search.

Preprocessing	**Parameters**	**F1-score**
None (Raw data)	C : 50, penalty: l2	0,3090
PCA	C : 50, penalty: l2	0,3028
PCA + SMOTE	C : 35, penalty: l2	0,4504
PCA + KMeans SMOTE	C : 35, penalty: l2	0,3887
PCA + ClusterCentroids	C : 20, penalty: l2	0,4213

Table 6: Results obtained with Logistic Regression model and different preprocessing strategies

According to Figure 25, it's easy to notice how the best performances have been obtained using PCA and SMOTE oversampling technique. Moreover, the confusion matrix has been provided.

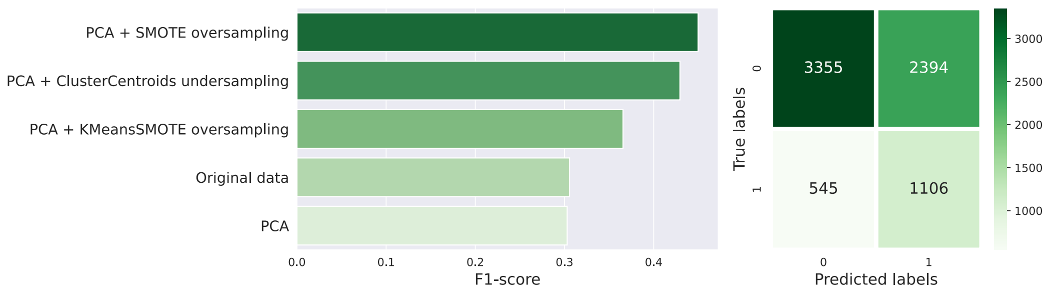


Figure 25: Results obtained for Logistic regression Classifier using different settings

6.2 Decision Tree

According to Figure 26, it's easy to notice how the best performances have been obtained using original data without any manipulation technique. Moreover, the confusion matrix has been provided.

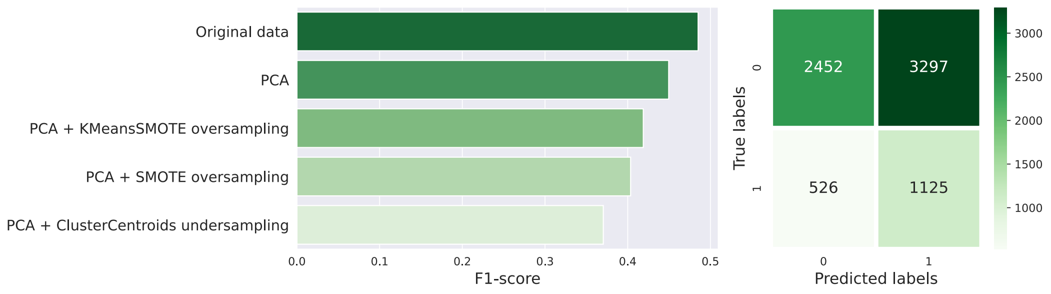


Figure 26: Results obtained for Decision Tree Classifier using different settings

6.3 Random forest

In the RandomForestClassifier implementation in SciKit-Learn, the size of the bootstrap sample is chosen to be equal to the number of training samples in the original training dataset, which usually provides a good bias-variance tradeoff [6]. Therefore we are interested in tuning the number of trees that form the forest (*n_estimators*) and the maximum number of features to consider in each split (*max_features*). Again we perform a Cross-Validated Grid Search to tune these parameters, and results found are reported in Table 7.

According to Figure 27, it's easy to notice how the best performances have been obtained using PCA and SMOTE oversampling technique. Moreover, the confusion matrix has been provided.

Preprocessing	**Parameters**	**F1-score**
None (Raw data)	max_features:None, n_estimators: 200	0,4972
PCA	max_features:None, n_estimators: 200	0,4609
PCA + SMOTE	max_features:sqrt, n_estimators: 200	0,4919
PCA + KMeans SMOTE	max_features:None, n_estimators: 50	0,4607
PCA + ClusterCentroids	max_features:sqrt, n_estimators: 100	0,4153

Table 7: Results obtained with Random Forest model and different preprocessing strategies

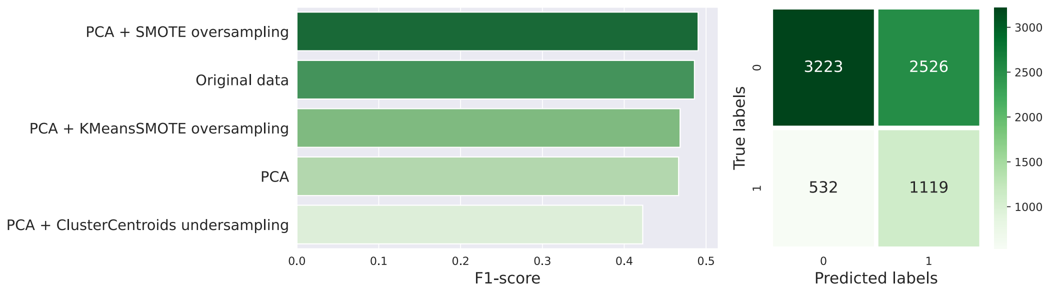


Figure 27: Results obtained for Random forest using different settings

6.4 Support vector machine

Some combinations of parameters value and kernels have been explored through a CrossValidated grid search which involved also different choices about the preprocessing phase. Results obtained are reported in Table 8 and Figure 16. According to Figure 28, it's easy to notice how the best performances have been obtained using PCA and SMOTE oversampling technique. Moreover, the confusion matrix has been provided.

Preprocessing	Parameters	F1-score
None (Raw data)	kernel: RBF, C : 100, λ : 0.1	0,4681
PCA	kernel: RBF, C : 100, λ : 0.1	0,4665
PCA + SMOTE	kernel: RBF, C : 100, λ :scale	0,5247
PCA + KMeans SMOTE	kernel: RBF, C : 100, λ :scale	0,4639
PCA + ClusterCentroids	kernel: RBF, C : 100, λ :scale	0,4341

Table 8: Results obtained with SVM model and different preprocessing strategies

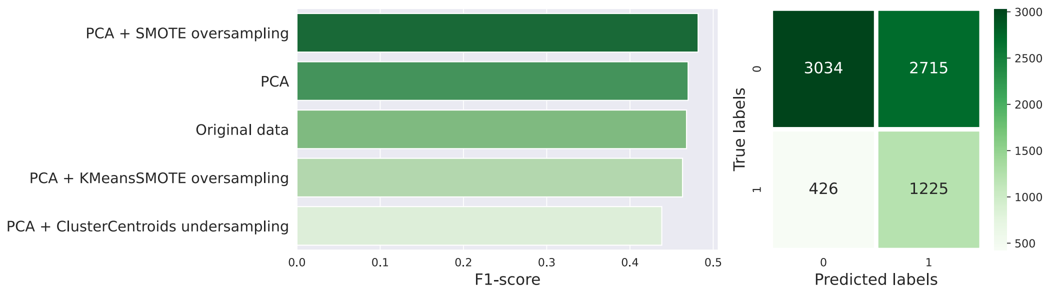


Figure 28: Results obtained for SVM Classifier using different settings

6.5 Overall overview

In the following Table 9 the final result obtained with all technique for a confrontation.

		Accuracy	Recall	Precision	F1-score	AUC
	Original data	0.805946	0.191399	0.757794	0.305609	0.564799
	PCA	0.805405	0.189582	0.754217	0.303001	0.562305
Logistic Reg	**PCA + SMOTE**	0.632703	0.672925	0.337793	0.449798	0.541846
	PCA + KMeansSMOTE	0.720000	0.361599	0.369659	0.365585	0.436845
	PCA + ClusterCentroids	0.602838	0.669897	0.316	0.429431	0.529773
	Original data	0.825135	0.34464	0.728553	0.467928	0.609704
	PCA	0.82527	0.347062	0.727157	0.469865	0.609948
SVM	**PCA + SMOTE**	0.689054	0.648092	0.383513	0.481873	0.555059
	PCA + KMeansSMOTE	0.748108	0.486978	0.441516	0.463134	0.521476
	PCA + ClusterCentroids	0.575541	0.741975	0.310914	0.438204	0.555228
	Original data	0.824189	0.37129	0.699772	0.48516	0.605666
	PCA	0.803108	0.360388	0.59739	0.449566	0.55024
Decision Tree	**PCA + SMOTE**	0.687973	0.473047	0.351802	0.403513	0.471208
	PCA + KMeansSMOTE	0.733378	0.430648	0.407683	0.418851	0.482679
	PCA + ClusterCentroids	0.483378	0.681405	0.25441	0.370492	0.503448
	Original data	0.819189	0.382798	0.664564	0.48578	0.592532
	PCA	0.811081	0.370079	0.630547	0.466412	0.570583
Random Forest	**PCA + SMOTE**	0.762027	0.513022	0.469512	0.490304	0.545592
	PCA + KMeansSMOTE	0.784865	0.424591	0.521966	0.46827	0.537468
	PCA + ClusterCentroids	0.586757	0.677771	0.306996	0.422583	0.528329

Table 9: Final result obtained with all technique

7 Conclusion

In this study, different supervised learning algorithms have been inspected and presented with their mathematical details, and finally used on the UCI dataset to build a classification model that is able to predict if a credit card clients will default in the next month. Data preprocessing makes algorithms perform slightly better than when trained with original data: in particular, PCA results are approximately the same, but the computational cost has been lowered. Oversampling and undersampling techniques has been combined with PCA to assess the dataset imbalance problem. Oversampling as mentioned performed slightly better w.r.t. the undersampling, this is likely because the model is trained on a large amount of data. However, all the models implemented achieved comparable results in terms of accuracy.

However, our result are quite low and other methods may be explored trying to get better performances. It would be interesting to implement some Gradient Boost based models such as Gradient Boosting Classifier or SGD Classifier, and also some outliers' management approaches such as Local Outliers Factor or Isolation Forest could help to improve our results.

References

[1] I.-C. Yeh and C. hui Lien, “The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients,” Expert Systems with Applications, vol. 36, no. 2, pp. 2473–2480, Mar. 2009. [Online]. Available: :https://doi.org/10.1016/j.eswa.2007.12.020.

[2] D. Dua and C. Graff, “Uci machine learning repository: default of credit card clients data set,” 2019. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients.

[3] T. M. Alam, K. Shaukat, I. A. Hameed, S. Luo, M. U. Sarwar, S. Shabbir, J. Li, and M. Khushi, “An investigation of credit card default prediction in the imbalanced datasets,” IEEE Access, vol. 8, pp. 201 173–201 198, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.3033784.

[4] Muchiri, L. M. (2020). A Model for predicting credit card loan defaulting using cardholder characteristics and account transaction activities [Thesis, Strathmore University]. http://hdl.handle.net/11071/12127.

[5] Ying Chen, Ruirui Zhang, ”Research on Credit Card Default Prediction Based on k-Means SMOTE and BP Neural Network”, Complexity, Volume 2021, Article ID 6618841, 13 pages, 2021. https://doi.org/10.1155/2021/6618841.

[6] S. Raschka, Python Machine Learning. Packt Publishing, 2015.

[7] S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning. Cambridge University Press, 2009. [Online]. Available: https://doi.org/10.1017/cbo9781107298019.

[8] S.-J. Yen and Y.-S. Lee, “Cluster-based under-sampling approaches for imbalanced data distributions,” Expert Systems with Applications, vol. 36, no. 3, pp. 5718–5727, apr 2009. [Online]. Available: https://doi.org/10.1016/j.eswa.2008.06.108.

[9] G. Lemaitre, F. Nogueira, and C. K. Aridas, “Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning,” Sep 2016. [Online]. Available: https://arxiv.org/abs/1609.06570.

[10] Q. Zou, Z. Wang, X. Guan, B. Liu, Y. Wu, and Z. Lin, “An approach for identifying cytokines based on a novel ensemble classifier,” BioMed Research International, vol. 2013, pp. 1–11, 2013. [Online]. Available: https://doi.org/10.1155/2013/686090.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: Synthetic minority over-sampling technique,” 2011. [Online]. Available: https://arxiv.org/abs/1106.1813.

[12] F. Last, G. Douzas, and F. Bacao, “Oversampling for imbalanced learning based on k-means and smote,” Nov 2017. [Online]. Available: https://arxiv.org/abs/1711.00837.

[13] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, ser. IJCAI’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 1137–1143.

[14] P.-N. Tan, Introduction to Data Mining, 2nd ed. Pearson Education Limited, 2019.

[15] A. Agresti, An Introduction to Categorical Data Analysis, 3rd ed., ser. Wiley Series in Probability and Statistics. John Wiley and Sons, 2019. [Online]. Available: http://gen.lib.rus.ec/book/index.php?md5=ec387de4af731cc168b9f0506700f5cc.

[16] D. P. K. Z. I. B. T. T. R. Vaisman, Data Science and Machine Learning: Mathematical and Statistical Methods, ser. Chapman and Hall/CRC Machine Leraning and Pattern Recognition. CRC Press, 2020.

[17] T. H. R. T. Gareth James, Daniela Witten, An Introduction to Statistical Learning with Applications in R, ser. Springer Texts in Statistics, Vol. 103. Springer, 2013.

[18] Robert I. Kabacoff, "R in Action", 2nd ed., Manning Publications, 2015. http://www.cs.uni.edu/ jacobson/4772/week11/R$_i$$n_A$$ction.pdf$.

[19] E. a. M. Mohri, Foundations of machine learning. MIT press, 2018.

[20] A. Smola, S.V.N. Vishwanathan, "Introduction to Machine Learning", 2008. https://alex.smola.org/drafts/thebook.pdf.