Politecnico di Torino

Mathematics for Machine Learning Project

MSc Data Science and Engineering

Online Shoppers Intention

Andrea Rubeis s290216

Professor Francesco Vaccarino
Professor Mauro Gasparini

Academic Year 2021-2022

Contents

1 | Introduction

Before presenting the project an introduction and a bit of context are needed to explain why predicting online shoppers intention is a crucial aspect for companies nowadays.
The rise of Internet in 80's had the power to connect people from all around the world. It would not be surprising notice that, once that Internet reality was well consolidated and public available to everyone, retailers started to see the potential of this new technology. Internet indeed has allowed shopkeepers to exponentially increase their sales by selling to people worldwide and not only to the locals anymore. Customers started shopping online more frequently and in larger numbers, particularly during the COVID-19 pandemic and this behaviour is predicted to persist even after the pandemic (Figure 1.1).

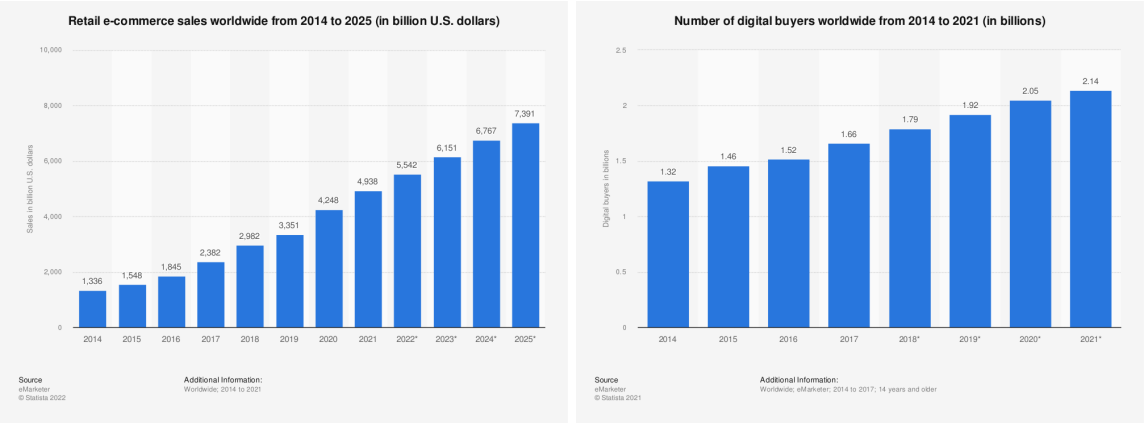


Figure 1.1: Retail e-commerce sales worldwide from 2014 to 2025 (left)
Global number of digital buyers from 2014 to 2021 (right)

One of the most crucial aspects an e-commerce platform would like to know is whether the user is simply browsing or making a purchase. This is crucial since different consumers require different treatments, and the platform should always make the most of its limited resources to attract and retain the finest clients. If the platform has higher confidence that a certain segment of users is more likely to make a purchase, it can take proactive measures such as providing time-limited offers or even a one-month free trial, as Amazon does, to encourage users to complete their orders. In the end, this will help sellers in identifying the suitable clients, enhancing client retention, and ultimately boosting sales and earnings.

2 | Dataset and Data Exploaration

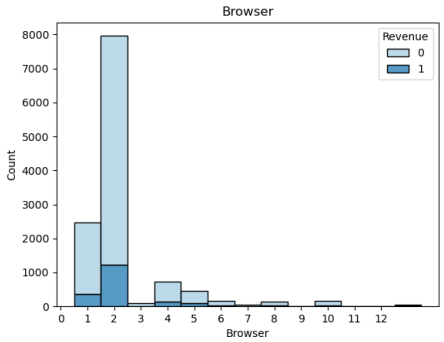
To simulate different experiments, the public avaiable UCI dataset has been used [1]. The collection is made up of 18 feature vectors (10 numerical and 8 categorical) and 12,330 different sessions. Each of them would be owned by a different user throughout the course of a year to prevent any inclination to a certain cause, holiday, or user period, or profile.

2.1 | Categorical Features

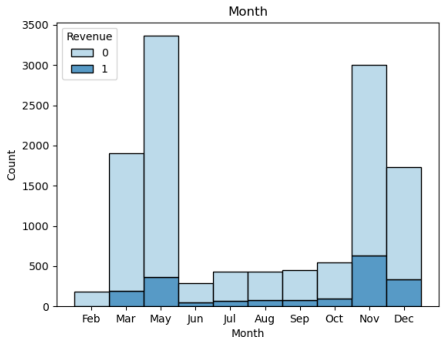
Feature Name	Description	NaNs	Distinct Values	Distinct Count
Browser	Browser of the visitor	0	[1 2 3 4 5 6 7 10 8 9 12 13 11]	13
Month	Month value of the visit date	0	['Feb' 'Mar' 'May' 'Oct' 'June' 'Jul' 'Aug' 'Nov' 'Sep' 'Dec']	10
OperatingSystems	Operating system of the visitor	0	[1 2 4 3 7 6 8 5]	8
Region	Geographic region from which the session has been started by the visitor	0	[1 9 2 3 4 5 6 7 8]	9
Revenue	Class label indicating whether the visit has been finalized with a transaction	0	[False True]	2
TrafficType	Traffic source by which the visitor has arrived at the Web site (e.g., banner, SMS, direct)	0	[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 18 19 16 17 20]	20
VisitorType	Visitor type as "New Visitor," "Returning Visitor," and "Other"	0	['Returning_Visitor' 'New_Visitor' 'Other']	3
Weekend	Boolean value indicating whether the date of the visit is weekend	0	[False True]	2

Figure 2.1: Summary description of categorical features.

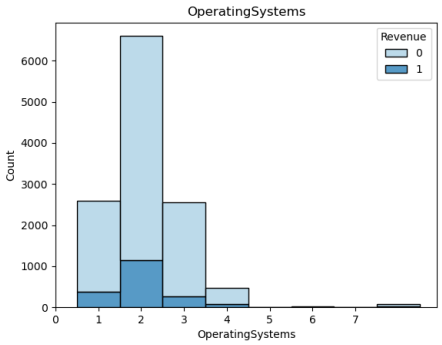
Data Exploration Categorical Features



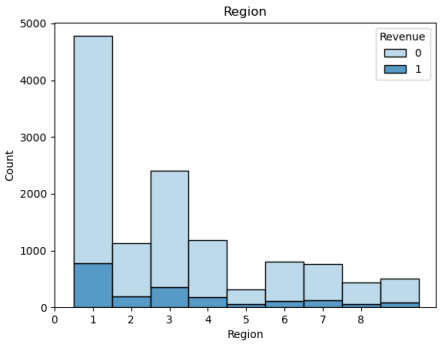
(a) Browser on Revenue



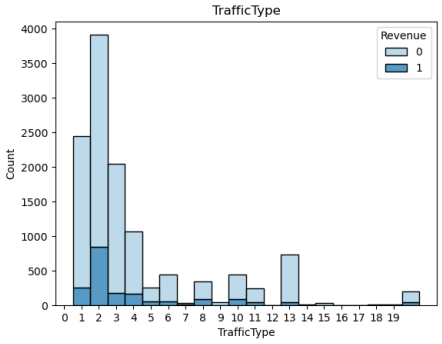
(b) Month on Revenue



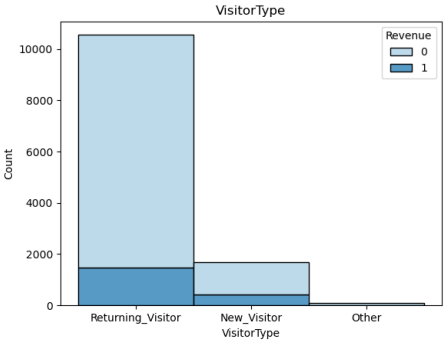
(c) Operating Systems on Revenue



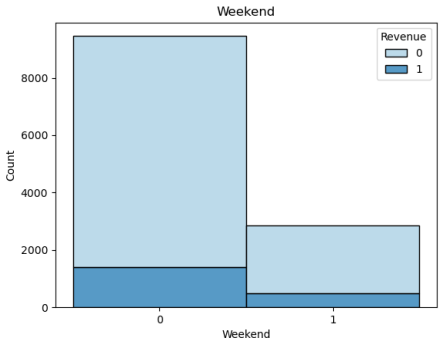
(d) Region on Revenue



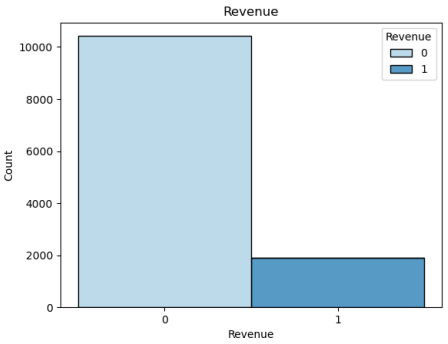
(e) Traffic Type on Revenue



(f) Month on Revenue



(g) Weekend on Revenue



(h) Revenue class balance

Figure 2.2: Categorical features count values grouped by Revenue.

2.2 | Comments Categorical Features

Counting and ranking the number of samples (i.e. the users) belonging to each category help us to desume on which categories investments of products' advertisments may be result more effective.
Figure 2.2a shows that the most 3 popular browsers used are 2 followed by 1 and 4, so advertisments of products through browser 2 may result more effective. Same reasoning can be applied for all the other figures in Figure 2.2 and corresponding rankings can be summarized in Table 2.1.

Table 2.1: Ranking of categorical values.

Ranking	Browser	Month	OS	Region	Traffic Type	Visitor Type	Weekend
1	**2**	**May**	**2**	**1**	**2**	**Returning**	**Not**
2	1	Nov	1	3	1	New	Yes
3	4	Mar	3	4	3	Other	-
4	5	Dec	4	2	4	-	-

Table 2.1 suggests that the company should advertise its products in May by traffic 2 on workweek days, address ads to returning visitors who use browser 2, operating system 2, located in region 1 to reach the most vast audit among its customers.
However, it should be stressed out that reaching a vast audit of customers through advertisements does not necessarily implies that they will buy for sure the product advertised.

2.3 | Imbalanced Dataset

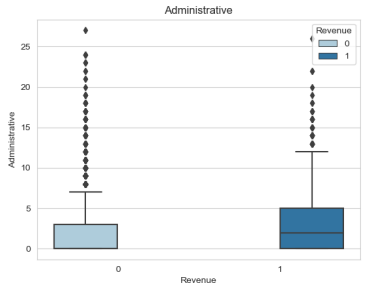
In addiction to this analysis in every figure can be noticed that, in proportion, visitors who did not conclude the transaction are much more than visitors who did it. This is demonstrated by Figure 2.2h that clearly shows the class label's imbalance, where $10,422$ not completed transactions against $1,908$ successful ones are present. This may affect the predictions of the models that will be used later so proper resampling techniques will be applied to guarantee good results.

2.4 | Numerical Features

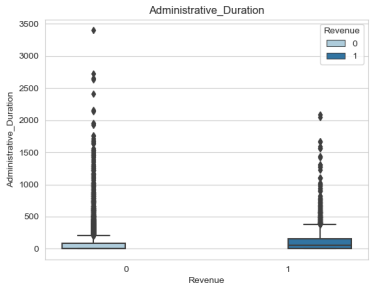
Feature Name	Description	NaNs	Min	Max	Mean	Std
Administrative	Number of pages visited by the visitor about account management.	0	0.000000	27.000000	2.315166	3.321784
Administrative_Duration	Total amount of time (in seconds) spent by the visitor on account management related pages	0	0.000000	3398.750000	80.818611	176.779107
BounceRates	Average bounce rate value of the pages visited by the visitor. BR is the proportion of users that arrive on a page of the website and leave without performing any further actions	0	0.000000	0.200000	0.022191	0.048488
ExitRates	Average exit rate value of the pages visited by the visitor. ER is the percentage of website pageviews actually end on that particular page	0	0.000000	0.200000	0.043073	0.048597
Informational	Number of pages visited by the visitor about Web site, communication and address information of the shopping site	0	0.000000	24.000000	0.503569	1.270156
Informational_Duration	Total amount of time (in seconds) spent by the visitor on informational pages	0	0.000000	2549.375000	34.472398	140.749294
PageValues	Average page value of the pages visited by the visitor. PV is the average page value over the value of the target page and/or the successful completion of an online purchase	0	0.000000	361.763742	5.889258	18.568437
ProductRelated	Number of pages visited by visitor about product related pages	0	0.000000	705.000000	31.731468	44.475503
ProductRelated_Duration	Total amount of time (in seconds) spent by the visitor on product related pages	0	0.000000	63973.522230	1194.746220	1913.669288
SpecialDay	Closeness of the site visiting time to a special day	0	0.000000	1.000000	0.061427	0.198917

Figure 2.3: Summary description of numerical features.

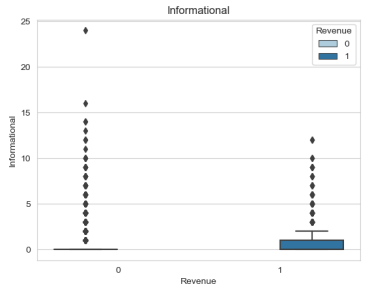
Data Exploration Numerical Features



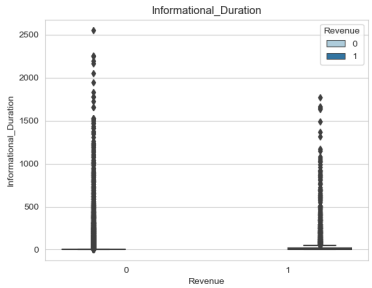
(a) Administrative on Revenue



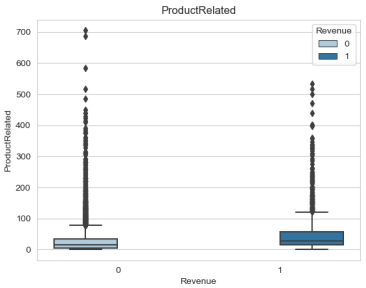
(b) Administrative Duration on Revenue



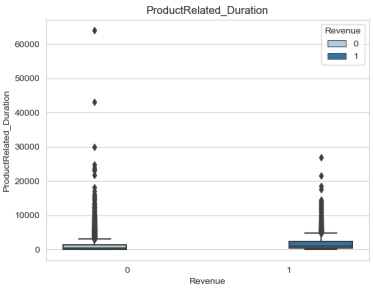
(c) Informational on Revenue



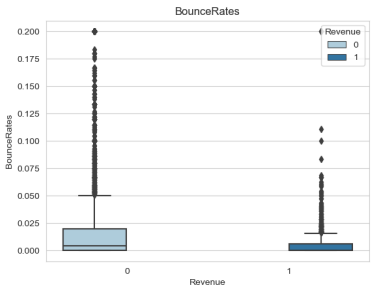
(d) Informational Duration on Revenue



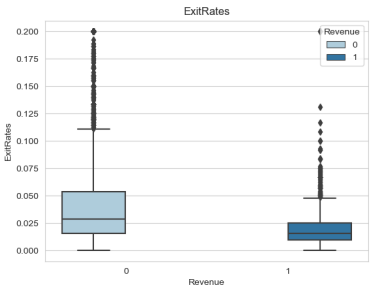
(e) Product Related on Revenue



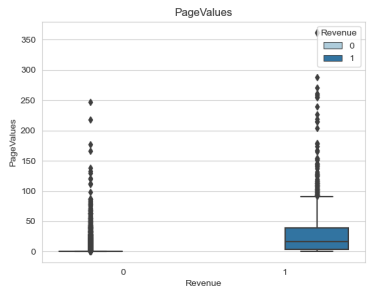
(f) Product Related Duration on Revenue



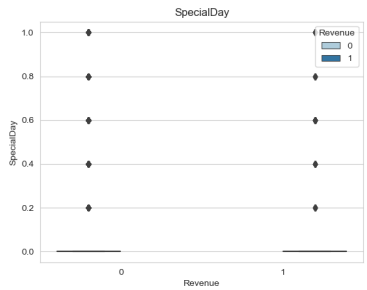
(g) Bounce Rates on Revenue



(h) Exit Rates on Revenue



(i) Page Values on Revenue



(j) Special Day on Revenue

Figure 2.4: Numerical features boxplots grouped by Revenue

2.5 | Comments Numerical Features

Boxplots are useful tools to provide summary statistics of numerical features by representing minimum, median, maximum, interquantile ranges and outliers of the numerical features' distributions. In this case, looking at Figure 2.4 every numerical feature in the dataset presents an abundant amount of outliers, meaning that every numerical feature assumes a really skewed distribution.
Since boxplots provide just a summary statistics of numerical features, the analysis on them is deepen by computing the correlation matrix to discover the existence of possible linear correlations.

2.6 | Feature Correlation

Pearson Correlation

$$\rho = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{2.1}$$

The numerator is the covariance between 2 random variables while the denominator is given by the product of their standard deviations.

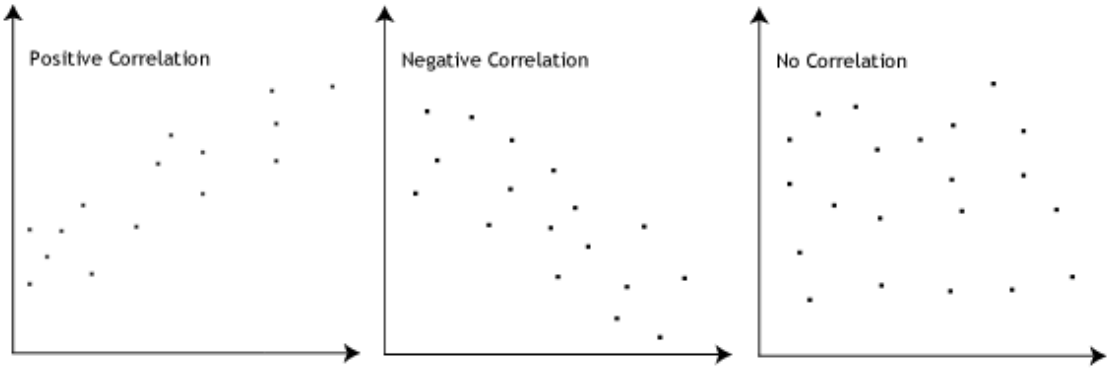


Figure 2.5: Examples of possible correlations between 2 variables.

This coefficient can assume values in the range $[-1, 1]$, a negative value means that the two random variables are negatively correlated (if one increase the other decrease or viceversa), 0 indicates no correlation while positive value means they are positively correlated (both increase or decrease). Positive correlation scenario means that both features bring the same amount of information so one can be dropped since it does not bring any further information.

The correlation matrix in Figure 2.6 reported below represents the Pearson's Correlation Coefficients (2.1) computed between all the possible pairs of numerical features in the dataset.

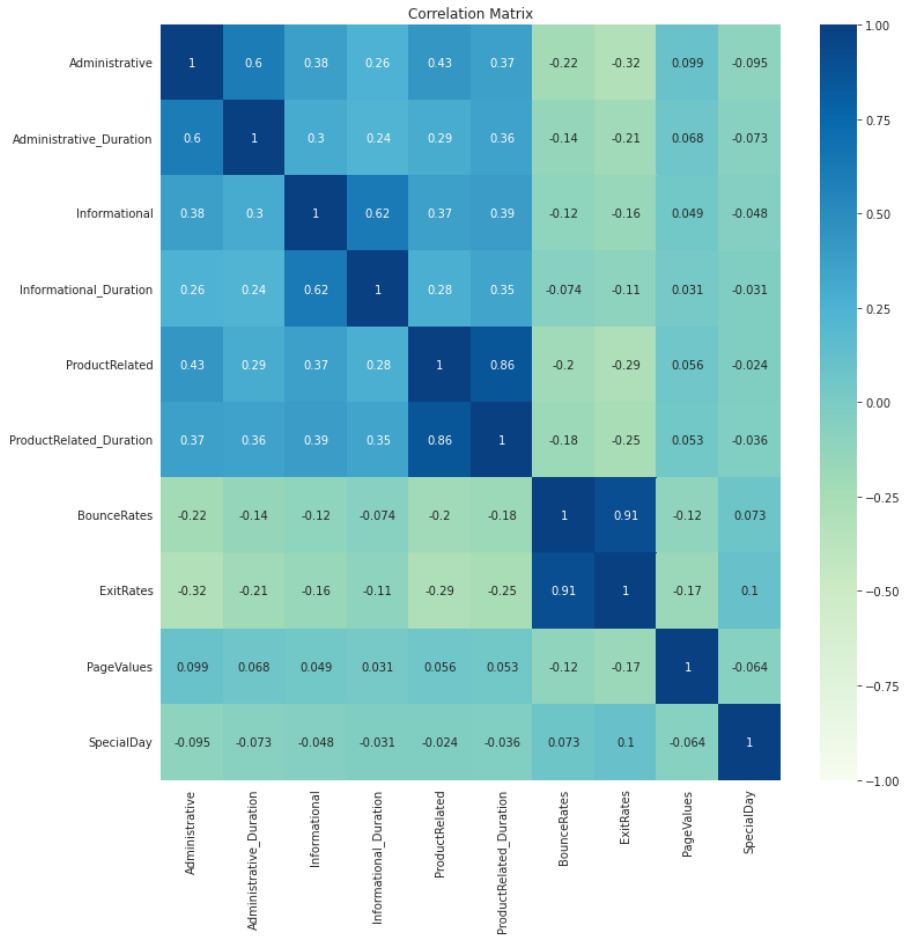


Figure 2.6: Correlation Matrix

From Figure 2.6, the most positively correlated features are *ExitRates* and *BounceRates*, followed by the couples: *Administrative* - *Administrative_Duration*, *Informational* - *Informational_Duration* and *ProductRelated* - *ProductRelated_Duration*.

For this reason the features *BounceRates*, *Administrative_Duration*, $Informational\_Duration$, *ProductRelated_Duration* are dropped, while *ExitRates*, *Administrative*, $Informational$ and *ProductRelated* are kept.

3 | Data Preprocessing

First of all boolean features as *Weekend* and *Revenue* are converted in integer values (*True* and *False* become 1 and 0 respectively).
Next, there is not necessity to apply any kind of strategy to handle missing data since missing values are absent (Figures 2.1, 2.3). Outliers are kept since the majority of numerical features' distributions are skewed and filtering outliers may imply the loss of too much information.
The aim of data preprocessing is to achieve a final numerical dataset on which classifiers can be trained on. To obtain it, *one hot encoding* and *standardization* techniques are applied to categorical and numerical features respectively.

3.1 | One Hot Encoding

Machine learning models are not able to deal with text, so converting categorical features in numerical ones through *one hot encoding* technique (Figure 3.1) is needed.

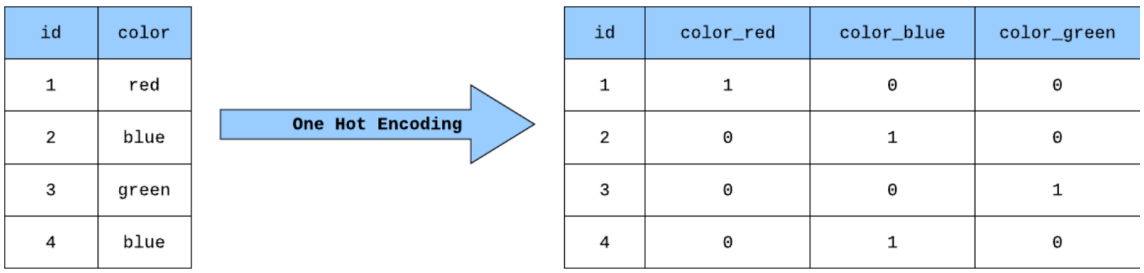


Figure 3.1: Example of one hot encoding technique application.

One hot encoding converts the categorical features by creating n additional columns, where n is the number of unique values assumed by the categorical column to convert.
One of the main drawbacks of one hot encoding technique is the *multicollinearity problem*, consisting in high correlation among independent predictors. To avoid it, it is necessary to drop one of the dummy variables created, so instead of having n columns more only *n-1* are present.

3.2 | Standardization

After cleaning the data, standardization step for numerical features is performed, since the majority of numerical features present skewed distributions defined on different ranges. Standardization aims to make them comparable one each other. In this case the function *StandardScaler()* from *sklearn* Python library is used, in which every numerical feature is standardized as a distribution with mean μ equals to 0 and unitary standard deviation σ according to the following formula:

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

3.3 | Principal Component Analysis

PCA is probably the most used dimensionality reduction technique among data scientists. It is an unsupervised machine learning algorithm, that from data in high dimensional space, derives a reduced hierarchical coordinate system by capturing the highest variance of the original data.

To understand the necessity of having as much variance as possible, one can thinks to an exaggerate case in which the dataset is reduced to just a single dimension. In this case, since the variance represents the spread of the data, the feature that will be selected should be the one that accounts for the most possible variability, so, the one that preserves the most information coming from the original dataset.

PCA steps

The training dataset $X \in \mathbb{R}^{n,d}$ can be thought as n vectors composed by d features each.

1. Compute the mean row vector μ and the average matrix $\bar{X}$:

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j \quad \bar{X} = \vec{1}^T \mu \tag{3.2}$$

2. Compute the mean centered matrix B:

$$B = X - \bar{X} \tag{3.3}$$

   This passage is equivalent to centering the distribution of original data around the origin.

3. Compute the sample covariance matrix C of B:

$$C = \frac{1}{n-1} B^T B \tag{3.4}$$

4. Compute the eigenvalue decomposition of C:

$$C = Q \Lambda Q^T \tag{3.5}$$

   where the columns of Q represent the eigenvectors which correspond to the *principal components* (PCs), while Λ represents a diagonal matrix with the corresponding eigenvalues.

5. Order the eigenvalues of Λ in decreasing order, the higher the eigenvalue is, the more important the corresponding PC is.

6. Take the first s PCs that best represent the original dataset in terms of variance to project data onto the new lower s dimensional space:

$$X_s = X Q_s \tag{3.6}$$

In this case eigenvalue decompostition is used to solve PCA, however it can be solved with SVD decompostition as well as done by *sklearn* library.

The B matrix can be decomposed according to SVD:

$$B = U \Sigma V^T \tag{3.7}$$

So 3.4 can be rewritten as:

$$C = \frac{B^T B}{n-1} = \frac{V \Sigma U^T U \Sigma V^T}{n-1} = V \frac{\Sigma^2}{n-1} V^T \tag{3.8}$$

Looking at 3.5, the relationship between Σ and Λ can be easily recognized:

$$\Lambda = \frac{\Sigma^2}{n-1} \tag{3.9}$$

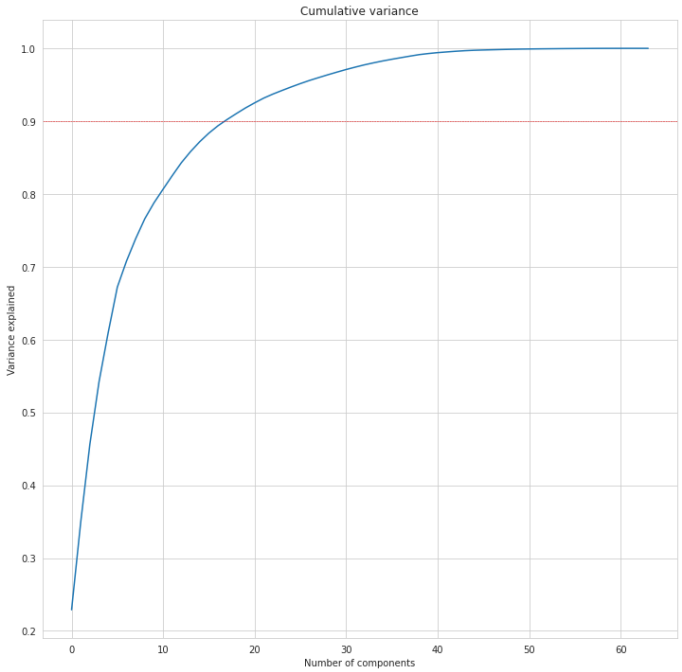


Figure 3.2: Cumulative variance explained.

Notice that PCA should be applied after the *train-test split* on train dataset only and then the test dataset should be projected on PCs found by training dataset. If indeed PCA would be applied to the whole dataset before the train-test split would be problematic since the PCs were found by contribution of test dataset as well leading to biased predictions.
For this project after application of PCA the 64 total features of training dataset reduced to 17, accounting for the 90% of the cumulative variance explained (Figure 3.2).

3.4 | Data Balance

As already said, from Figure 2.2h is evident that the class label is heavily imbalanced. So, after PCA, and before training any kind of classifier, a rebalancing technique should be applied in order to obtain a balanced dataset and avoid possible biases in final predictions.
To rebalnce a dataset there exist two main approaches: *undersampling* and *oversampling*.

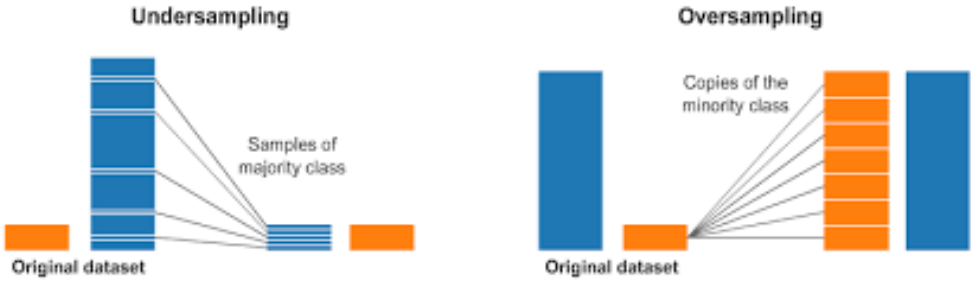


Figure 3.3: Undersampling and oversampling techniques.

In this project, 4 different scenarios are implemented in order to see which rebalancing technique is more effective.

- **Original**: no resampling technique is applied.
- **Undersampling**: Near Miss algorithm.
- **Oversampling**: SMOTE algorithm.
- **Both (oversampling + undersampling)**: SMOTE + TomekLinks algorithms.

In the next sections a further detailed explanation for each resampling technique is reported.

3.5 | Undersampling

In order to better balance the class distribution, undersampling approaches eliminate examples from the training dataset that belong to the majority class.
The most trivial undersampling technique is *random undersampling* in which instances from the majority class are randomly chosen and removed from the training dataset. Although straightforward and efficient, this technique has a drawback: examples are discarded regardless how valuable can be in establishing the decision boundary between the 2 classes. Extensions of this approach like *NearMiss* and *TomekLinks* are based on an heuristic algorithm aiming to detect redundant examples to delete or valuable examples to keep.

3.5.1 | Near Miss UnderSampling

The term *near miss* refers to a group of undersampling techniques based on KNN approach [2] that choose majority class samples to keep depending on the distance between majority and minority class examples. There actually exist 3 different versions of this algorithm where the difference is given by the way in which majority examples to keep are selected.

- NearMiss-1: Majority examples with minimum average distance to three closest minority examples.
- NearMiss-2: Majority examples with minimum average distance to three furthest minority examples.
- NearMiss-3: Majority examples with minimum distance to each minority example.

In this project version 3 (Figure 3.4) is adopted, which basically consists in keeping the majority examples along the decision boundary.

Figure 3.4: NearMiss-3 undersampling technique.

This however can lead to adversely affect the final classifier's predictions since existing samples are eliminated.

3.5.2 | Undersampling - TomekLinks

Another common undersampling algorithm is *TomekLinks*. Denoting with A the set of majority class samples, B the set of minority class samples, this technique identifies pairs of samples (a, b), one from each class, and removes the sample belonging to the majority class (a) in each pair. These pairs are detected in such a way that they have the shortest Euclidean distance in the feature space, they are called *Tomek Links* and need to satisfy the following conditions:

- the nearest neighbor of instance a is b.
- the nearest neighbor of instance b is a.
- instances a and b belong to different classes.

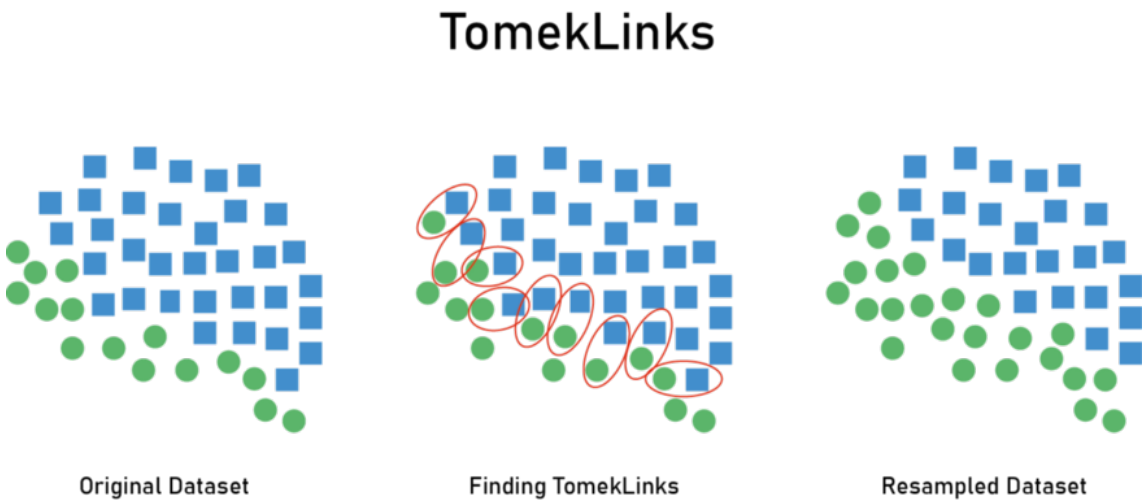


Figure 3.5: TomekLinks undersampling technique.

According to these conditions, TomekLinks usually represent boundary instances or noisy ones, since only this type of instances will have nearest neighbors that belong to a different class.
After TomekLinks application, it is not expected to have a balanced dataset, indeed the goal of TomekLinks procedure is to obtain a less ambiguous dataset along decision boundaries, not to have a final balanced dataset.
For this reason it usually used in conjunction with SMOTE oversampling technique discussed in the following section.

3.6 | Oversampling

In contrast to undersampling, oversampling aims to resample examples belonging to the minority class in order to obtain a proportioned final dataset.
In this case to get reliable predictions by a classifier, the oversampling technique should be only applied on the training set, *after* the train-test split, otherwise if it would be applied before to the whole dataset *overfitting* may occur.

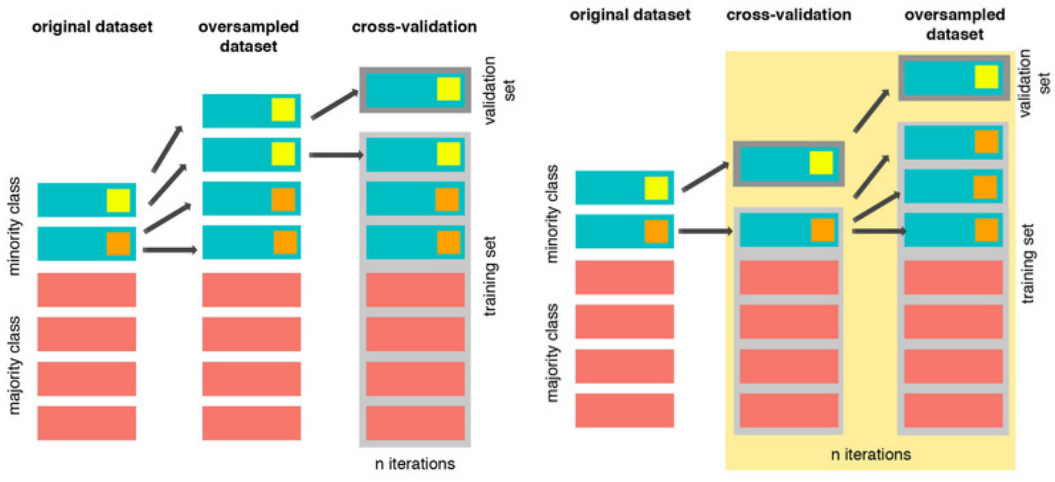


Figure 3.6: Training with oversampled dataset: wrong approach (left), correct approach (right),

As indeed Figure 3.6 (left) shows, if the oversampling technique is applied before the train-test split to the whole dataset, during cross validation it may be possible to have training and validation sets containing the same samples. Instead, if oversampling is applied after the train-test split only to training data, there is no way to have same data for training and testing at the same time (Figure 3.6 (right)).

3.6.1 | Oversampling - SMOTE

One of the most common used oversampling technique is SMOTE (Synthetic Minority Oversampling Technique). It randomly selects a minority class example and detects its k nearest neighbors. Then, one of the k neighbors is randomly picked and a new synthetic example is created along the line connecting the 2 samples in the feature space (Figure 3.7).

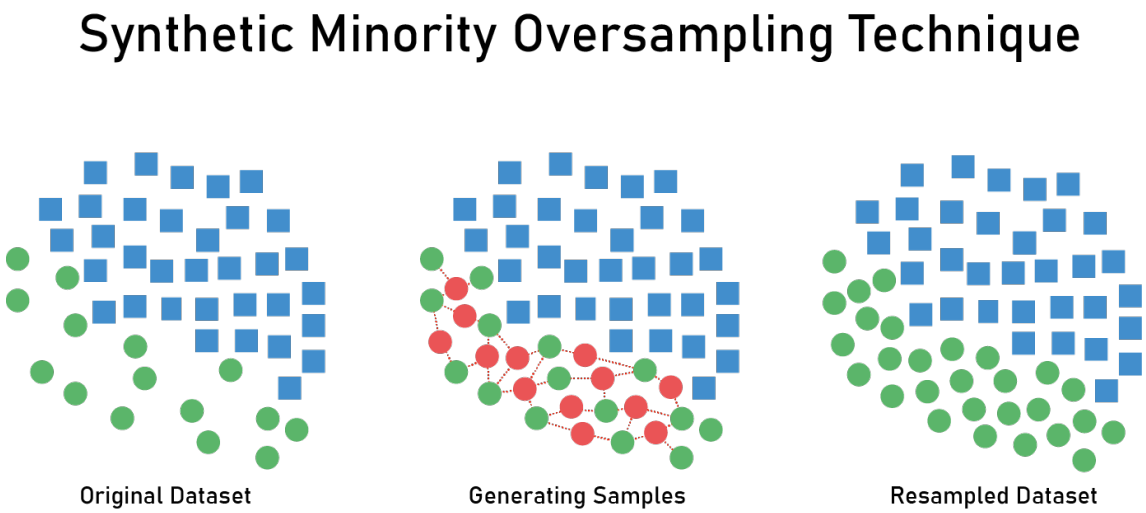


Figure 3.7: SMOTE technique

This strategy turns out to be effective since the synthetic minority class examples created are close to existing minority class ones. However in case of strong overlap between majority and minority class examples, the synthetic ones may result ambiguous and this approach may does not work so well.

4 | Training

Once that the dataset has been properly pre-processed, the training of different classifiers can start.
First of all, the dataset is split in a stratifed manner to have equal proportion of classes in training and test sets. The dimensions of the former X_{train} and the latter X_{test} are 80% and 20% of the original dataset respectively.
Secondly, to train the models stratified k-cross validation technique with $k = 10$ is used.
Stratified k-cross validation is a particular type of cross validation which guarantees that the proportion of classes reflects the one in the original data. It is a method to evaluate how well statistical analysis results generalize to an indipendent dataset, so that a model' scores do not depend on how the training and testing dataset are chosen. This is done by dividing the training dataset X_{train} in k folds, then the model is trained on X_{train_train} made by $k-1$ folds and tested on the remaining one X_{val_train} until each fold has been used as a test set at least once.

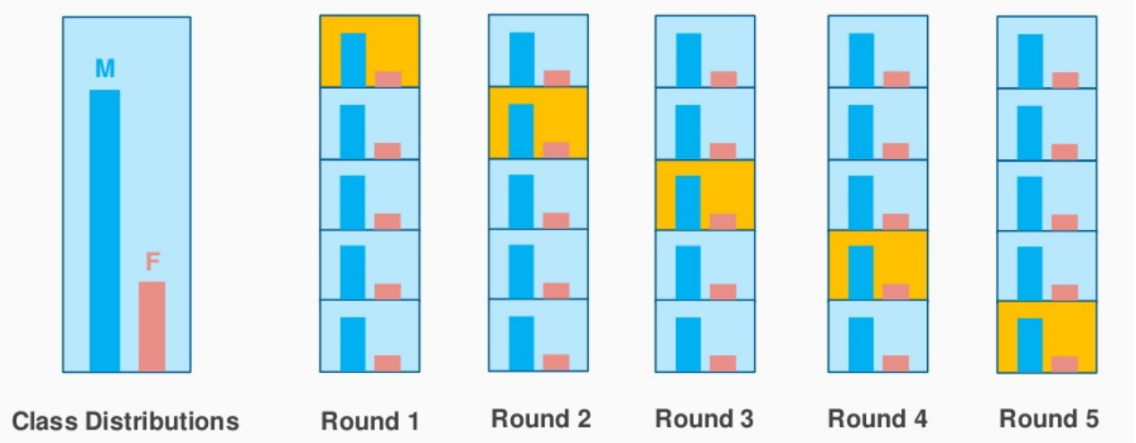


Figure 4.1: Example of stratified k cross validation (original scenario).

For each group of 10 iterations the classifier is trained with different hyperparameters in a *grid search* to find out which set of them may lead the classifier to make the best predictions.
To do this, an evaluation metric score m_s is stored for each iteration, at the end of the loop, at the 10^{th} iteration, the average of scores stored $\bar{m}_s$ is computed. So each different combination of hyperparameters adopted by the classifier will have its own $\bar{m}_s$ and the set of hyperparameters used to compute the final test on X_{test} will be the one with the highest $\bar{m}_s$.

For this project, given the imbalance nature of the problem, m_s concides with the weighted average of the $F1 - score$.

$$F1 - score = 2 \frac{Precision \times Recall}{Precision + Recall}, \quad Precision = \frac{TP}{TP + FP,} \quad Recall = \frac{TP}{TP + FN} \tag{4.1}$$

In the equation above TP, FP, TN and FN stand for true positives, false positives, true negatives and false negatives respectively.

5 | Test

5.1 | Logistic Regression

5.1.1 | Theory Background

Logistic Regression classifier belongs to a family of linear predictors which represent one of the most useful families of hypothesis classes due to its efficiency and easy interpretation. The family of linear predictors is made by affine functions defined as:

$$L_d = \{h_{w,b} : w \in \mathbb{R}^d, b \in \mathbb{R}\}, \qquad h_{w,b}(x) = \langle w, x \rangle + b = (\sum_{i=1}^{d} w_i x_i) + b \tag{5.1}$$

or equivalently,

$$L_d = \{x \to \langle w, x \rangle + b : w \in \mathbb{R}^d, b \in \mathbb{R}\} \tag{5.2}$$

L_d is a set of functions parametrized by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ which receives as input a vector x and outputs the scalar $\langle w, x \rangle + b$.
Logistic Regression is an hypothesis class of linear predictors obtained by the composition of the following function ϕ_{sig} on L_d.

$$H_{sig} = \phi_{sig} \circ L_d = \{x \to \phi_{sig}(\langle w, x \rangle + b) : w \in \mathbb{R}^d, b \in \mathbb{R}\} \tag{5.3}$$

$$z = \langle w, x \rangle + b, \qquad \phi_{sig} : \mathbb{R}^d \to [0, 1] \qquad \phi_{sig}(z) = \frac{1}{1 + exp(-z)} \tag{5.4}$$

The pedix *sig* of ϕ_{sig} stands for *sigmoid* (= "S-shaped") and the function $h(x)$, with $h \in H_{sig}$, can be thought as the probability to assign label 1 to x (Figure 5.1).

$$p = \mathbb{P}(y = 1|x) = \phi_{sig}(z) = \frac{1}{1 + \exp(-z)} = \frac{1}{1 + \exp(-(w_0 + w_1 x_1 + ... + w_n x_n))}, \quad w_0 = b \tag{5.5}$$

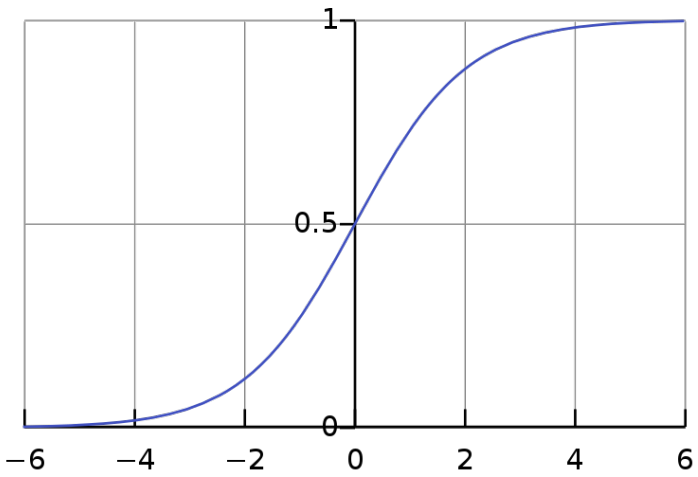


Figure 5.1: Sigmoid function

To estimate the regression coefficients Maximum Likelihood Estimation (MLE) is used. Among an infinite sets of regression coefficients, MLE selects the set for which the probability of getting the data observed is maximum. Since logisitic regression is used in binary classification tasks, Y_i can be seen as an indipendent Bernoulli random variable with $i = 1, ..., n$ and probability p: $Y_i \sim Bernoulli(p)$. Then the maximization of the likelihood function is:

$$L(y; w^*) = \arg\max_{w^*} \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1 - y_i} \tag{5.6}$$

To simplify futher calculations, logarithm can be applied since it is a monotonically increasing function and will not change the maximum.

$$L(y; w^*) = \arg\max_{w^*} \sum_{i=1}^{n} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \tag{5.7}$$

It can be proven that both loglikelihood function and likelihood function are concave and have the same unique solution. To find it iterative methods such as *newton*, *SGD* and others can be used.
It can be noticed that if a minus sign is added in front of the final loglikelihood, the logloss used by the logistic regression classifier is obtained:

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} (y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) \tag{5.8}$$

To avoid to incur in overfitting problem a *regularization term* can be added to the loglikelihood function.

$$J = L(y; w) + \lambda R(w) \tag{5.9}$$

An example of regularization is the so called L_2 which is defined as:

$$L_2 = \frac{1}{2}(w_0^2 + w_1^2 + ... + w_n^2) = \frac{1}{2} \sum_{i=0}^{n} w_i^2 \tag{5.10}$$

In *sklearn* this term is weighted by a parameter C whose higher values weight more the training data while lower values weigth more the penalty term.

5.1.2 | Hyperparameters

	LR Hyperparameters
C	100, 10, 1, 0.1, 0.01
Solvers	newton-cg, lbfgs, liblinear
Penalty	L_2

Table 5.1: LR hyperparameters used during cross-validation.

	Best LR Configurations
With/Without PCA	
Original	C = 0.1, s = liblinear, p = L_2
Undersampling	C = 0.01, s = newton-cg, p = L_2
Oversampling	C = 0.01, s = liblinear, p = L_2
Both	C = 0.01, s = liblinear, p = L_2

Table 5.2: Best LR hyperparameters found during cross-validation with and without PCA.

5.2 | K-Nearest Neighbors

5.2.1 | Theory Background

K-Nearest Neighbors classifier is a supervised machine learning algoithm whose idea is to memorize the training set and classify any new instance with the label of its closest neighbors in the feature space.
In a classification setting the KNN first memorizes all the training set $S = (x_1, y_1), ..., (x_m, y_m)$, then once an unseen sample q (*query*) to predict arrives, it computes all the distances $d(x_i, q), \forall x_i \in S$. Next it finds the nearest neighbors of q by ranking points by increasing distance. Finally it predicts q with the label of majority of samples in its k nearest negighbors. Example of k-NN rule with $k = 3$ and $k = 5$ is shown in Figure 5.2.

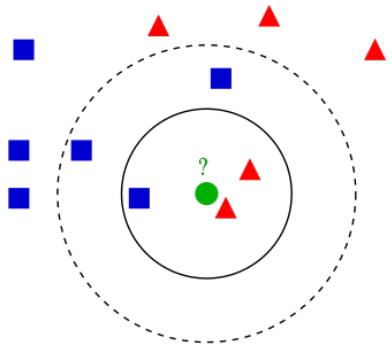


Figure 5.2: KNN classification example.

In the feature space different distances such as *euclidean, manhattan* and *malhanobis* can be defined. Beyond this, the prediction step can be computed by making use of a weight function which can be *uniform* (all neighbors are weighted equally) or *inverse distance* (closer neighbors of a query point will have a greater influence than neighbors which are further away).

5.2.2 | Hyperparameters

	KNN Hyperparameters
n_neighbors	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
weights	uniform, distance
metrics	euclidean, manhattan, minkowski

Table 5.3: KNN hyperparameters used during cross-validation.

	Best KNN Configurations
With/Without PCA	
Original	n.n. = 8, w = uniform, m = manhattan
Undersampling	n.n. = 2, w = uniform, m = manhattan
Oversampling	n.n. = 2, w = uniform, m = manhattan
Both	n.n. = 2, w = uniform, m = manhattan

Table 5.4: Best KNN hyperparameters found during cross-validation with and without PCA.

5.3 | Support Vector Machine

5.3.1 | Theory Background

Support Vector Machine classifier (SVM) is a supervised machine learning algortithm usually used in binary classification tasks that relies on hyperplanes.
Recalling formula 5.1, the family of halfspaces class can be obtained as:

$$H = sign \circ L_d = \{x \to sign(h_{w,b}(x)) : h_{w,b} \in L_d\} \tag{5.11}$$

which means that every point x is classified with label -1 or 1 depending on the value of $h_{w,b}(x)$. w represents the orthogonal vector to the hyperplane while b is the intercept term.
Given a training set $S = (x_1, y_1), ..., (x_n, y_n)$ with $y_i \in \{-1, 1\}$, SVM classifier seeks for the hyperplane that best separates the two classes of samples by maximizing the distance with respect to the *margin*, which is defined as the distance between the hyperplane and the closest samples belonging to the classes. Since there are 2 classes, there are 2 corresponding margins and the best hyperplane lies down in the middle between the two.

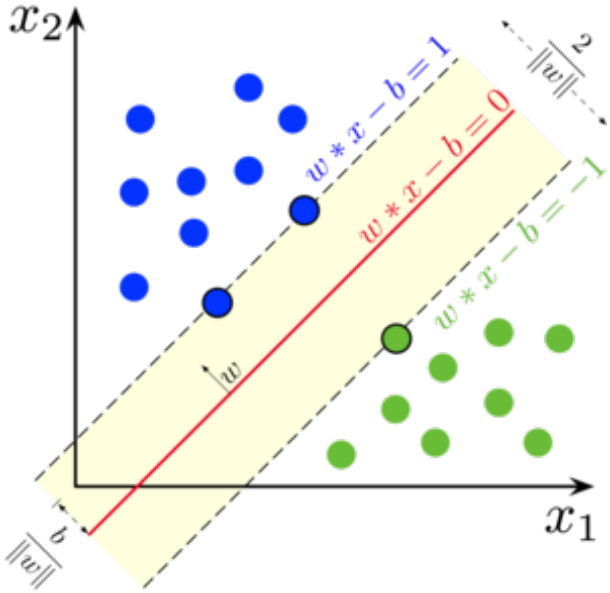


Figure 5.3: SVM classifier.

As Figure 5.3 illustrates, the best hyperplane in this case is represented by the red line since distance between it and the margins of the 2 classes is maximal. Samples lying along the margins are called *support vectors*.
Two formalizations of SVM can be defined: **Hard SVM** and **Soft SVM**.

Hard SVM In this case it is assumed that data are linearly separable.
Suppose to have an unseen x that needs to be classified, to know if it falls into the left or right side of the hyperplane. One way to discover it is to compute the dot product between x and w, which means the projection of x over w. Three cases can occur:

- $w \cdot x = b$, the sample lies on the decision boundary.
- $w \cdot x \geq b$, the sample is classified as positive sample.
- $w \cdot x \leq b$, the sample is classified as negative sample.

Then, the following constraints are added:

- $w \cdot x_+ + b \geq 1$, the positive samples are forced to fall outside the positive margin.
- $w \cdot x_- + b \leq -1$, the negative samples are forced to fall outside the negative margin.

Assuming that $y = -1$ identifies negative samples and $y = 1$ identifies positive ones the 2 constraints above can be rewritten as a single constraint:

$$y_i(\langle w, x \rangle + b) \geq 1 \tag{5.12}$$

The distance between the 2 margins concides with the distance between two support vectors x_+ and x_-:

$$(x_+ - x_-) \cdot \frac{w}{||w||} = \frac{x_+ \cdot w - x_- \cdot w}{||w||} \tag{5.13}$$

For the positive support vector x_+ $y = 1$:

$$1(w \cdot x_+ + b) = 1 \quad \rightarrow \quad w \cdot x_+ = 1 - b \tag{5.14}$$

For the negative support vector x_- $y = -1$:

$$-1(w \cdot x_- + b) = 1 \quad \rightarrow \quad w \cdot x_- = -1 - b \tag{5.15}$$

Replacing 5.14 and 5.15 in 5.13:

$$\frac{(1-b)-(-1-b)}{||w||} = \frac{1-b+1+b}{||w||} = \frac{2}{||w||} \tag{5.16}$$

So we get that the distance between the two support vectors is $\frac{2}{||w||}$. Since SVM aims to find an hyperplane maximizing this distance, the final objective function to maximize is:

$$\arg\max_{w,b} \frac{2}{||w||} \quad s.t. \quad y_i(\langle w, x\rangle + b) \geq 1 \tag{5.17}$$

However, in Hard SVM the training dataset is assumed to be perfectly linearly separable which is a quite rare event in real applications, for this reason in this section is presented an approximated version of SVM called Soft Margin SVM.

Soft SVM In this case since the dataset is not perfectly linearly separable some misclassifications are allowed, so, Soft SVM can be seen as a relaxation of the Hard SVM rule that can be applied even if data are not linearly separable. In this case 5.17 can be rewritten as:

$$\arg\min_{w} \frac{||w||^2}{2} + C\sum_{i=1}^{n} \zeta_i \qquad s.t. \quad y_i(\langle w, x\rangle + b) \geq 1 - \zeta_i, \quad \zeta_i \geq 0 \quad \forall i \in [1, ..., n] \tag{5.18}$$

The relaxation consists in violating the hard constraint $y_i(\langle w, x\rangle + b) \geq 1$ for some training examples by introducing a non-negative slack variable $\zeta = \zeta_1, ..., \zeta_n$. The measure of how much a training sample x_i violates its hard margin is represented by the variable ζ_i, which is weighted by the hyperparameter C. The smaller C is, the lower the penalization for misclassified examples will be, on the other hand higher values of C imply more severe penalization for misclassified examples (strict margins).

Kernels in Support Vector Machine If data are not linear, SVM can still be applied by exploiting the so called *kernel trick* which maps data from a lower dimensional space to a higher dimensional space through kernel functions, able to find decision boundaries able to clearly separate the data in the new space.

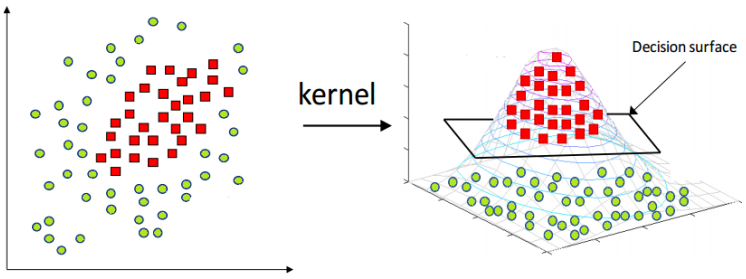


Figure 5.4: Kernel trick.

Polynomial Kernel: denoting with x_1 and x_2 two different observations in the dataset, with r and d the coefficient and the degree of the polynomial respectively, the following equation can be written:

$$k_p = (x_1 x_2 + r)^d \tag{5.19}$$

To better understand how it works, suppose to have $r = \frac{1}{2}$ and $d = 2$, so we are mapping the data from one to two dimensional space.

$$\begin{aligned}(x_1x_2 + \frac{1}{2})^2 &= (x_1x_2 + \frac{1}{2})(x_1x_2 + \frac{1}{2}) \\ &= x_1x_2 + x_1^2x_2^2 + \frac{1}{4} \\ &= (x_1, x_1^2, \frac{1}{2}) \cdot (x_2, x_2^2, \frac{1}{2})\end{aligned} \tag{5.20}$$

The first and second tuple refers to the (x, y, z) coordinates of x_1 and x_2 respectively. Since $z_1 = z_2$, then x_1 and x_2 live in a 2D plane and have (x_1, x_1^2) and (x_2, x_2^2) coordinates.

Radial Kernel: it finds support vector classifiers in infinite dimensions. Radial Kernel reveals how much influence each observation in the training set has on the classification of the new observation. Denoting again with x_1 and x_2 two different observations in the dataset and γ as a weighting factor for the distance between them, the kernel is defined as:

$$k_r = e^{-\gamma(x_1 - x_2)^2} \tag{5.21}$$

This kernel returns the high-dimensional relationships between observations where the amount of influence that one observation has on the other is a function of squared distance weighted by parameter γ. So, Radial Kernel can be seen as a weighted Nearest Neighbors model where the nearest neighbors have a lot of influence on the classification of the new observation, while the farthest contribute less.

Sigmoid Kernel: the last kernel function presented maps the data into a new space by making use of an hyperbolic tangent:

$$k_s = tanh(cx_1^T x_2 + r) \tag{5.22}$$

c is a scaling parameter of input data while r can be seen as a shifting parameter that controls the threshold of mapping.

5.3.2 | Hyperparameters

In order to find the most suitable SVM classifier to each scenario (*original, undersampling, oversampling, both*) the following hyperparameters were tested during the cross validation.

	SVM Hyperparameters
Kernels	poly, rbf, sigmoid
C	50, 10, 1, 0.1, 0.01
γ	scale

Table 5.5: SVM hyperparameters used during cross-validation.

The follwing table reports the best configurations found for each scenario:

	Best SVM Configurations
With/Without PCA	
Original	K = rbf, C = 0.01, γ = *scale*
Undersampling	K = poly, C = 0.01, γ = *scale*
Oversampling	K = poly, C = 0.01, γ = *scale*
Both	K = poly, C = 0.01, γ = *scale*

Table 5.6: Best SVM hyperparameters found during cross-validation with and without PCA.

5.4 | Random Forest

5.4.1 | Theory Background

Random Forest classifier is an ensemble learning method in which a group of decision trees are combined to obtain a more efficient predictive model.

A **decision tree** recursively splits the training set by some decision rules defined on features until reaching the leaf nodes (labels). Then, to classify an unseen sample, the decision tree is traversed starting from the root and moving on along the tree by satisfying the conditions inside each node until reaching a leaf node. The best split is found by maximizing the *purity* of each node by computing either the Gini index or the entropy.
Decision trees are highly sensitive to training data which implies high variance so may struggle to generalize. To alleviate this problem, random forest classifiers come into play.

A **random forest** classifier is created by building n new datasets from the original one. To generate the new datasets the rows of the original dataset are randomly selected with replacement, meaning that the same sample can appear multiple times within the new created dataset, and each new dataset will have the same number of rows of the original one. These new generated datasets are called *bootstrapped datasets*.

Furthermore, on each bootstrapped dataset a decision tree will be trained on just a random subset of features. Everytime a new unseen sample has to be classified, it will be passed to each decision tree that will output its prediction, once the sample passed all n decision trees, the final prediction for classification will concide with the majority voting.
The process of merging results coming from different models is called *aggregation*.

These two steps are often reffered with the term *bagging* (bootstrap + aggregating) (Figure 5.5).

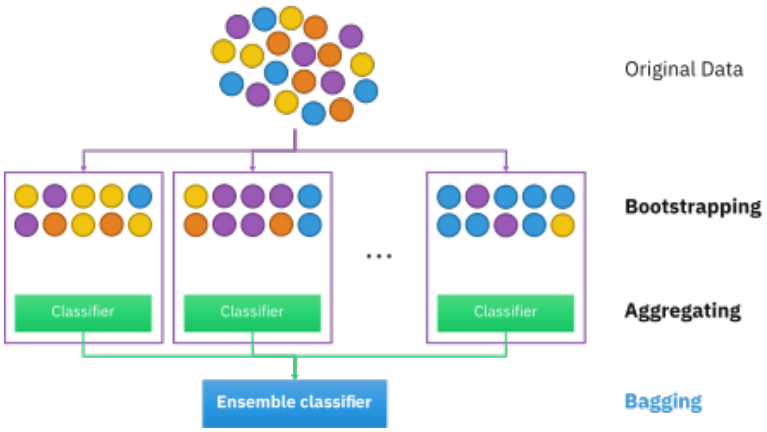


Figure 5.5: Bootstrap and Aggregation in Random Forest classifiers.

The word ”random” in random forest states for the fact that two random processes were performed: bootstrapping and feature selection.
Random bootstrapping avoids to use the same data for each tree reducing the sensitivity to original data, while random feature selection helps to decrease the correlation between trees. Indeed using same features leads to have same decision nodes in most of the trees, acting really similarly. It has been demonstrated that selecting the log or the square root of the total number of features allow to reach good performances.

5.4.2 | Hyperparams

	RF Hyperparameters
n. estimators	10, 100, 1000
max_features	sqrt, $\log_2$

Table 5.7: RF hyperparameters used during cross-validation.

	Best RF Configurations
Without PCA	
Original	n.e. = 1000 , m.f. = $\log_2$
Undersampling	n.e. = 1000 , m.f. = sqrt
Oversampling	n.e. = 100, m.f. = sqrt
Both	n.e. = 1000 , m.f. = sqrt
With PCA	
Original	n.e.= 1000, m.f. = log2
Undersampling	n.e. = 1000 , m.f. = sqrt
Oversampling	n.e. = 1000, m.f. = sqrt
Both	n.e. = 1000, m.f. = sqrt

Table 5.8: Best RF hyperparameters found during cross-validation with and without PCA.

6 | Results

In this section results obtained with the best combination of hyperparameters for each classifier and for each scenario (*original, undersampling, oversampling, both*) are reported. Since the dataset was imbalanced, the metric adopted to evaluate the models is the auROC, which is the area below the ROC curve computed at test phase. Below tables and graphs are shown to illustrate the precise results and the trend respectively.

Model	Original	Undersampling	Oversampling	Both
LR	0.657	0.758	0.803	0.801
KNN	0.612	0.638	0.705	0.704
SVM	0.589	0.557	0.700	0.700
RF	0.742	0.817	0.823	0.823

Table 6.1: auROC obtained without PCA

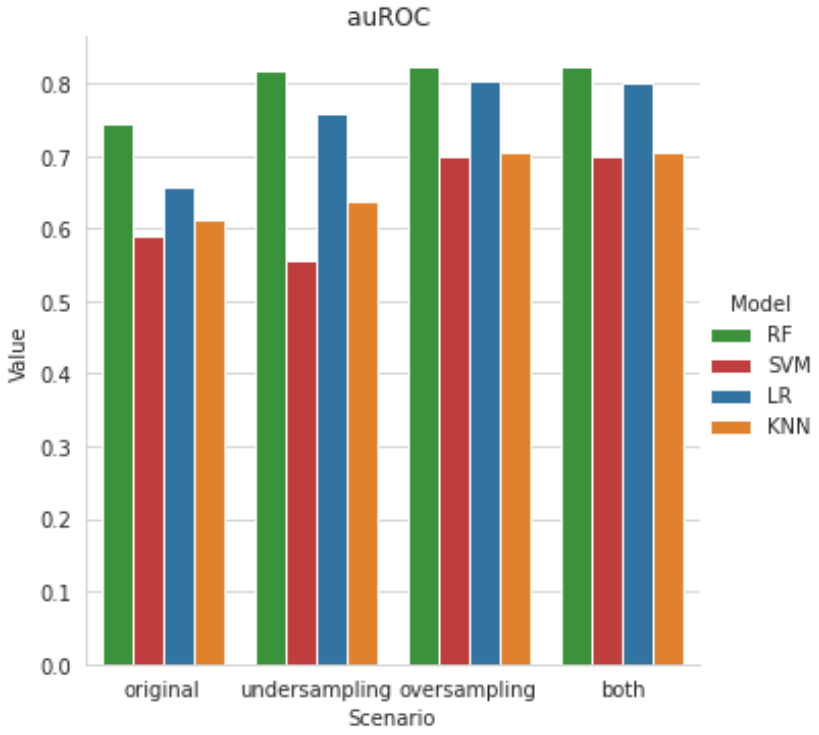


Figure 6.1: auROC computed for each classifier and scenario without PCA.

Model	Original	Undersampling	Oversampling	Both
LR	0.659	0.753	0.821	0.821
KNN	0.654	0.621	0.699	0.700
SVM	0.618	0.561	0.699	0.699
RF	0.734	0.762	0.798	0.800

Table 6.2: auROC obtained with PCA

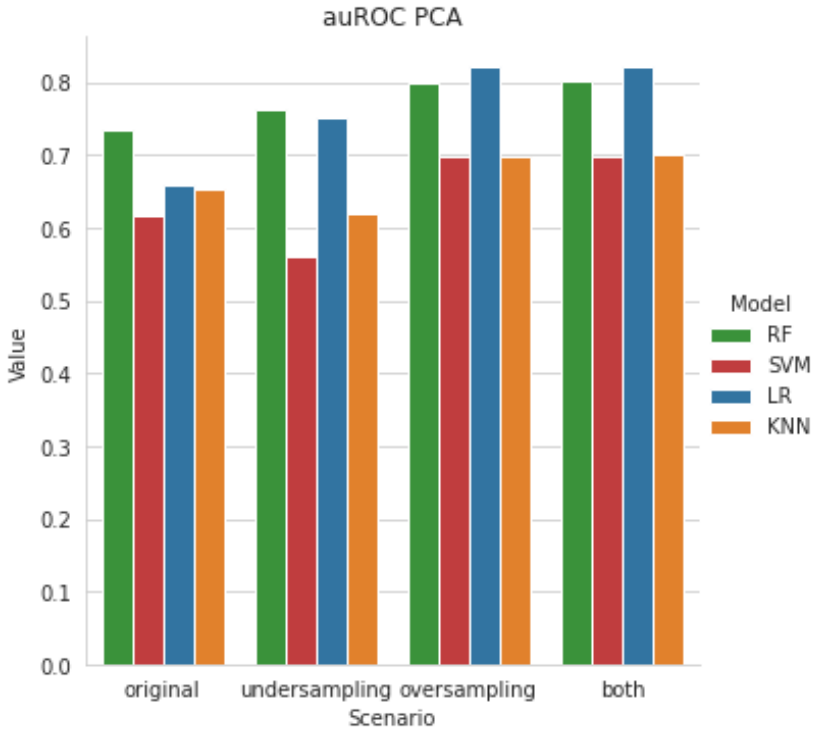


Figure 6.2: auROC computed for each classifier and scenario with PCA.

By looking at both graphs, as expected, the worst performances are achieved in the *original* and *undersampling* scenarios. Indeed in the former no measure to rebalance data has been taken into account, while in the latter possible relevant samples have been discarded.
The auROC reaches its best results with SMOTE and SMOTETomekLinks. However, the addiction of TomekLinks to SMOTE does not seem to improve so much the results already obtained with SMOTE only.
Beyond this, it can be noticed that oversampling is preferrable compared to undersampling since both SMOTE and SMOTETomekLinks always outperform NearMiss.
To conclude, it can be said that the best models for this task are the logistic regression and the random forest classifiers, since they are the best 2 classifiers in every scenario.

7 | References

[1] C Okan Sakar, S Olcay Polat, Mete Katircioglu, and Yomi Kastro. Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and lstm recurrent neural networks. *Neural Computing and Applications*, 31(10):6893–6908, 2019.

[2] Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126, pages 1–7. ICML, 2003.