Term deposits analysis

Gabriele Gioetto
Matricola: s285501

Politecnico di Torino

Mathematics in Machine Learning
A.Y. 2020/2021

Contents

1 Introduction

A term deposit is a fixed-term investment that includes the deposit of money into an account at a financial institution. The investor can withdraw their funds only after the term ends. In some cases, the account holder may allow the investor early termination or withdrawal if they give several days notification, and he will probably be given a penalty fee.
It's important for a bank to know if a person is going to subscribe for a term deposit, because it usually uses money from the term deposits to lend to people/companies in the form of loans such as personal loans, home loans, car loans, etc. It can be useful for a bank to have an algorithm that says if a new client will hold a term deposit up to a certain degree of certainty, in order to do beforehand the investment plans for the future using realistic data about the capitals the bank can invest.
The data we will use is related with direct marketing campaigns of a Portuguese banking institution and it's available on this link
The code used is avaible on Github or Colab

2 Data

The problem is a binary classification problem, in which the model we create will give us as output *yes* if the new client we are analyzing is likely to open a term deposit, otherwise *no*.
The database is composed by 45211 entries, in which there aren't *Null* data for any of the features. Excluding the target variable y, there are six categorical features, three binary features and seven numeric features

3 Data exploration

As already said before there are no null data, so there is no need to remove any record for now.
Looking at the distribution of the target variable y we can see that we have an imbalanced dataset, which we will see later how to handle.
To give an idea how the data are distributed among the different features, we have plotted barplots (Figure 2) for the categorical features and violinplots (Figure 1) for numerical features. In particular in the violin plots we have added the line of the 99.9-percentile and of the 0.01-percentile to try to notice outliers. The barplots are normalized with respect to the variable y to compensate for the class imbalance. If we didn't normalize the data the counts plots of the variable y positive would have been difficult to visualize.
Thanks to these plots we can notice that:

- The majority of people's age is in the range of 30-50 years old
- An higher level of education corresponds to more likelihood to open a term deposit

Feature's name	Description	Type
Age	Age of the client	Numeric
Job	Type of job of the client (categorical: "admin", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")	Categorical
Marital	Marital status of the client (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)	Categorical
Education	Level of education of the client (categorical: "unknown", "secondary", "primary", "tertiary")	Categorical
Default	If the client has credit in default	Binary
Balance	Average yearly balance of the client, in euros	Numeric
Housing	If the client has housing loan	Binary
Loan	If the client has personal loan	Binary
Contact	Contact communication type of the client (categorical: "unknown", "telephone", "cellular")	Categorical
Day	Last contact day of the month	Numeric
Month	Last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")	Categorical
Duration	Last contact duration, in seconds	Numeric
Campaign	Number of contacts performed during this campaign and for this client (numeric, includes last contact)	Numeric
P_days	Number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)	Numeric
Previous	Number of contacts performed before this campaign and for this client	Numeric
P_outcome	Outcome of the previous marketing campaign (categorical: "unknown", "other", "failure", "success")	Categorical
y	If the client has the client subscribed a term deposit	Binary

Table 1: All features present it the original dataset

- People who have credit in default are extremely rare
- Who has an housing loan isn't probably going to have also a deposit term.

The same can be said also for personal loans, but with a lower discrepancy

- If the last marketing campaign for the client has been successful it is almost certain that the client has opened a term deposit

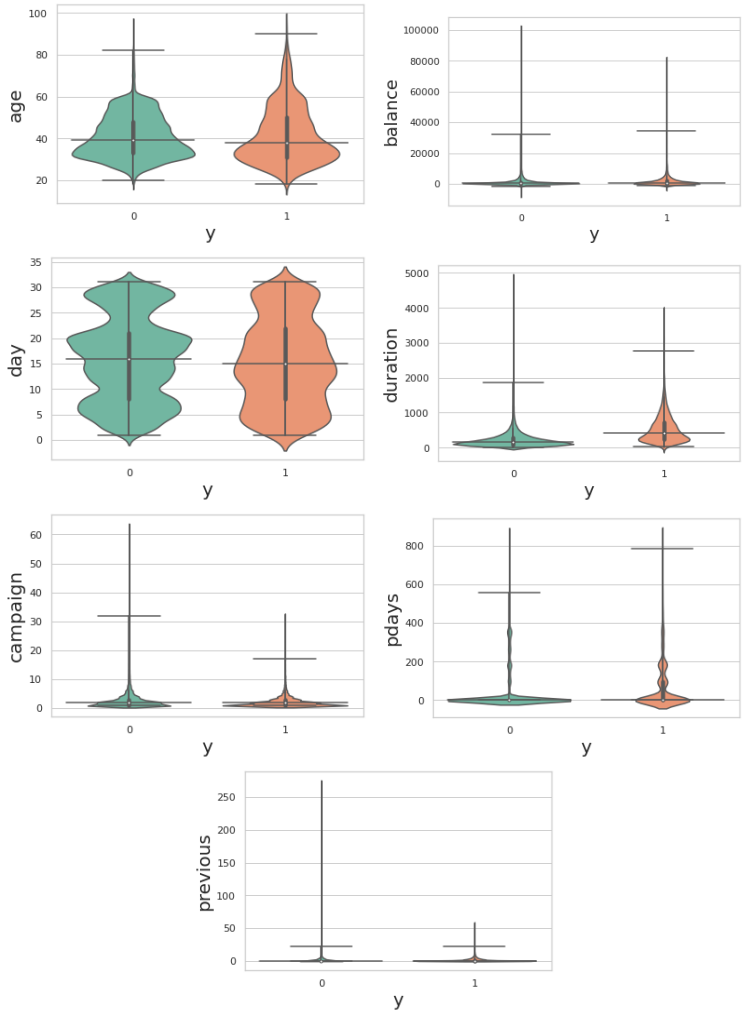


Figure 1: Boxplots and 99.9-percentiles and 0.01-percentiles of all the numerical features.

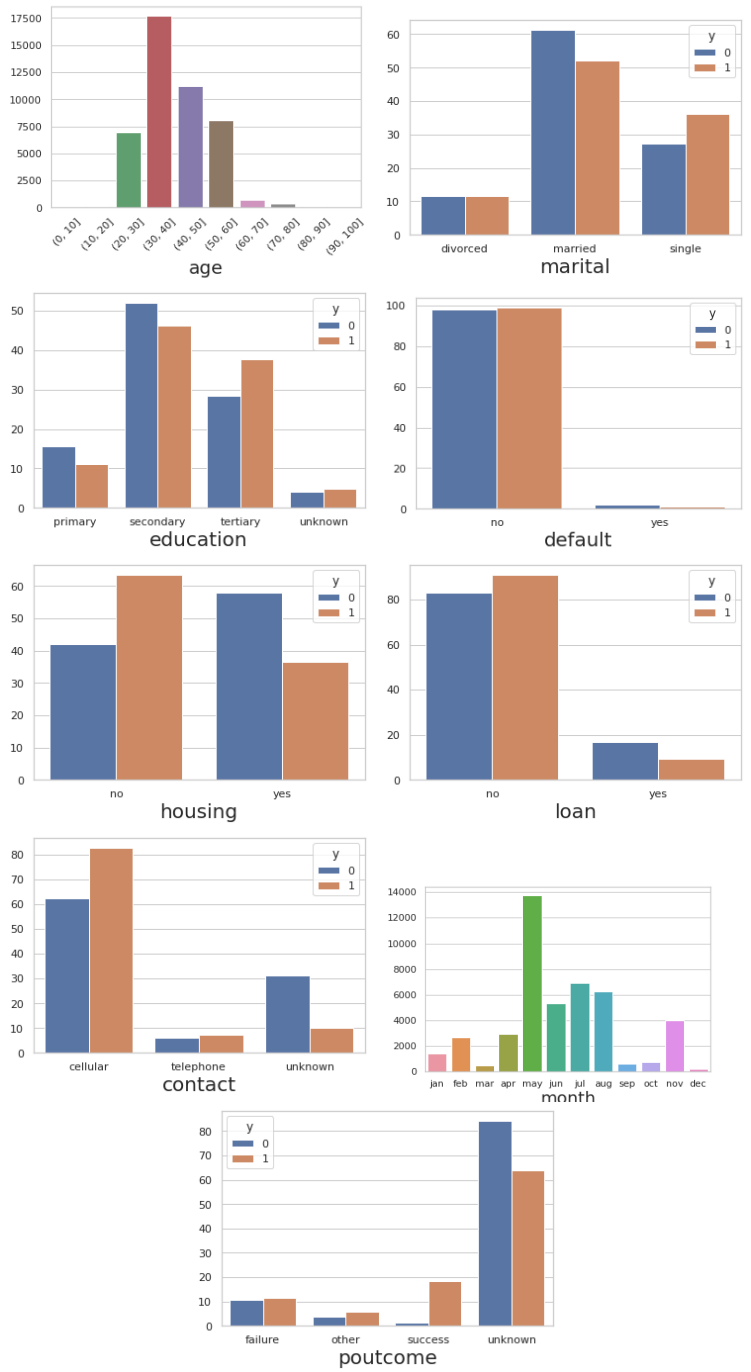


Figure 2: Barplots of all the categorical features, divided by y and normalized.

4 Pre-processing

4.1 Training test split

To build an effective model, we need a set of data that is considered new for our model. Te reason why is that if we trained a model on the same data that are being used to test it, we would suffer of data leakage. This means that the model would be overly optimistic, since it would already know information about how outputs and inputs are correlated on test data and we would create an overfitted model.
To avoid this phenomenon, we split the original data in two datasets, training set and test set, and all the operation will be executed only considering the train set. The proportion we chose for the split is 75/25.

4.2 Outliers

After having plotted the numerical data we have decided to remove the outliers from the dataset. We identify the outliers by calculating the 99.9-percentile for the features *balance*, *campaign*, *p-days* and *previous*, and by removing all records that exceed for the respective feature its percentile.
The records for which we removed outliers are *balance*, *duration*, *campaign*, *pdays*, *previous*. These were chosen by looking at the violinplots.

Feature	0.1-percentile	50-percentile	99-percentile	99.9-percentile
Age	19	39	70	80
Balance	-1497.9	445.0	12773.4	26106.5
Day	1.0	16.0	30.0	30.0
Duration	5.0	181.0	1237.0	1840.5
Campaign	1	2	15	26
P_days	-1	-1	369	521
Previous	0	0	8	16

Table 2: Percentiles of numerical features.

Feature	Removed records
balance	33
duration	34
campaign	31
p_days	34
previous	32

Table 3: Records removed after outliers elimination for each feature.

4.3 Correlation

Correlation is a statistical measure that expresses the extent to which two variables are linearly related. The closer the value of the correlation is to zero, the weaker the linear relationship.

$$\rho = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y}$$

$$\text{cov}(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

The correlation heatmap is a graphical way to show the correlation between all the numerical feature. If two features are heavily correlated (a standard threshold is 0.9), it means that they carry the same information, therefore one of them is useless.
In our case the maximum correlation absolute value is 0.45, between p-days and previous, which is too low to consider removing one of the features.

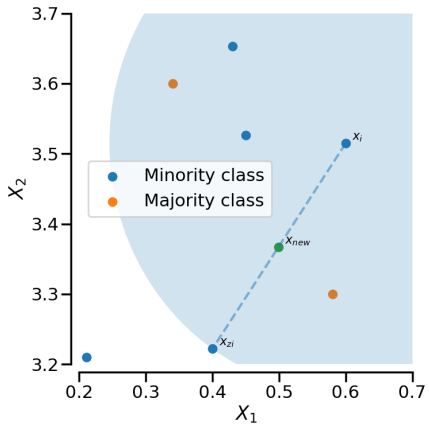
4.4 Class Imbalance and SMOTE-NC

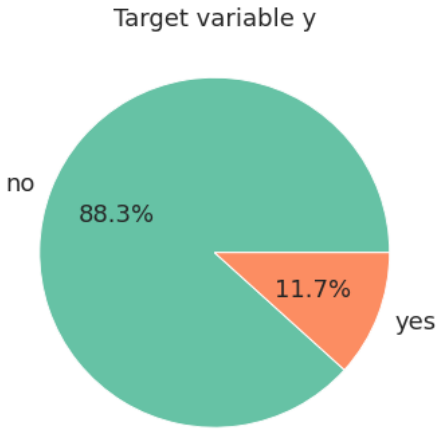
As we have seen at the start, the problem is unbalanced. In particular after the outlier removal, we have, with respect to the target variable y, 29827 negative records and only 3907 positive records.
Class imbalance can cause a model to overfit. For example in our case if we created a basic model that always returned a negative result, we would have an accuracy of 88.4%, even if the model is clearly inadequate.
To solve this problem we use an oversampling technique called SMOTE-NC, which is a slightly different version of SMOTE. To create new samples of the classes with the lowest frequency, SMOTE consider a sample x_i, then consider its k nearest-neighbors and picks randomly one of them. Lastly it generates the

new sample using the formula

$$x_{new} = x_i + \lambda(x_z - x_i)$$

where λ is a random number in the range [0,1].
SMOTE-NC is a version of SMOTE that is implemented to improve the results on data that have both numerical and categorical features. Compared to the standard version of SMOTE, the categories of a new generated sample are decided by picking the most frequent category of the nearest neighbors present during the generation.





4.5 Encoding categorical data

Machine learning algorithms need numerical data, so we are going to encode the categorical feature into numerical features. The encoder we have chosen is the

one-hot encoder, which create from a feature that has k possible different values, k - 1 binary features. If the feature's value is the i-th value, then the i-th binary feature value will be one. If the feature's value corresponds to the k-th value, then all the new binary features will equal to zero. For the particular case of binary features (Ex. housing, default...) it will be created only one feature that will be equal to one if the original value was "yes", 0 otherwise After applying the one-hot encoder we have in total 42 features.
If we re-calculate the correlation matrix, this time including also the encoded categorical features, we don't find again features with correlation value upper than 0.9 in absolute value.

$$feature_i = \begin{cases} 1 & \text{if } feature = value_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

4.6 Feature selection

Among the different possible explanatory variables, we want to select those which best explain the observed response. By eliminating redundant explanatory variables, we reduce the statistical error without increasing the approximation error, and thus reduce the (expected) generalization risk of the learner. We will try and compare two different methods for selecting features: forward selection and backward elimination.
Forward selection is an iterative model in which we build a logistic regression model trying to predict the target variable y using in each step of the cycle a different feature. Then we pick the feature that has the lowest p-value (unless it is bigger than a threshold value 0.5) and we add it to the model. Lastly we reiterate the model trying adding again all the features to the ones we already selected and we pick again the best one.

1. Initialize list of selected feature to empty .
2. For each feature not already selected:
   (a) Build regression model y feature + already selected features
   (b) Calculate p-value
   (c) If p-value < 0.05 save it
3. If we previously saved any p-value
   (a) Select the lowest p-value
   (b) Add the feature with the lowest p-value to the list
   (c) Go to 2

Backward elimination is also an iterative model, but in contrast to feature selection we start with the complete model. Then at each step we remove the

variable with the highest p-value, as long as it is not significant (greater than 0.05).

1. Initialize list of selected feature to every feature .
2. Build logistic regression model using the selected features
3. Calculate p-values for each feature
4. Pick the max p-value
5. If p-value > 0.05 remove the feature and go to 2

The two models we have used only use single variables, without taking account the possible combinations of them.
In the end the forward selection does not remove any features, even tweaking the α variable. Instead the backtrack selection removes 'job_unknown', 'poutcome_unknown', 'balance'. It makes sense that if the job is unknown or if the person's outcome of the previous campaign is unknown, these don't add relevant information for the prediction, so we choose the results of the backtrack selection.

4.7 Principal Component Analysis

PCA is an unsupervised dimensionality reduction technique that constructs relevant features through linear (linear PCA) or non-linear (kernel PCA) combinations of the original features. It is used to avoid the curse of dimensionality, which refers to the fact that when we have points in a high-dimensional space, they tend to be isolated between each other. The curse of dimensionality is caused by the intrinsic geometry of the space, it's not about the algorithm.
The construction of relevant features is achieved by projecting the original data into the reduced PCA space using the eigenvectors of the covariance matrix, since if our data has a normal multivariate distribution, the direction of largest variance is given by the eigenvector corresponding to the largest eigenvalue of the covariance matrix. The second principal component is orthogonal to the first one and represent the largest variance of the remaining subspace and so on so forth for the others
We chose to take k eigenvectors, such as the sum of the k corresponding eigenvalues' explained variance percentage is 0.9.
The optimization problem of PCA can be written as:

$$J(e_1, ..., 1_k, \alpha_1, ..., \alpha_k) = \sum_{j=1}^{n} \|x_j - \sum_{i=1}^{k} \alpha_{ji} e_i\|^2$$

After looking for the minimun for α we find that:

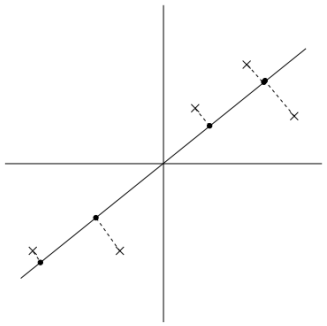
$$J(e_1, ..., e_k) = \sum_{i=1}^{k} \|x_j\|^2 - \sum_{i=1}^{k} e_i^t S e_i$$

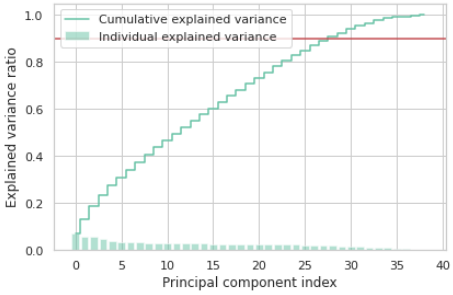
, where S: Scatter matrix
So minimizing J is equal to maximize $\sum_{i=1}^{k} e_i^t S e_i$, enforcing $e_i^t e_i = 1 \forall i$
After applying Lagrangian relaxation we find:

$$Se_m = \lambda_m e_m$$

The PCA's algorithm is:

1. Calculate $z_i = x_i - \hat{u}$, where $\hat{u} = \frac{1}{n} \sum_{i=1}^{n} x_i$
2. $S = \sum_{i=1}^{n} z_i z_i^t$
3. Compute k largest eigenvalues and eigenvectors $e_1, ..., e_k$ of S
4. Let $E = [e_1, ..., e_k]$
5. Project z on E, $y = E^t z$





5 Classification

Now we are going to use different classification algorithms to build different models.

5.1 Logistic Regression

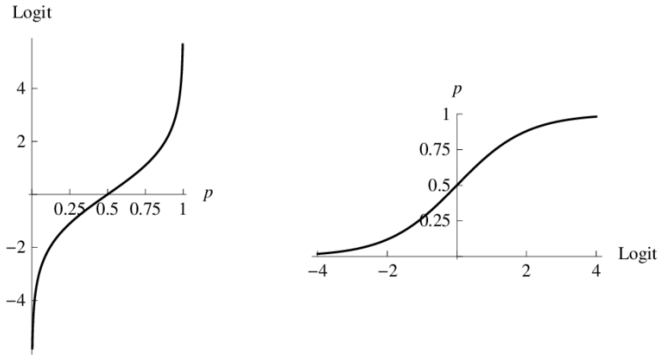
Logistic regression is a classification model that is used to model binary and multi-variable outcomes. In the logistic model the log odds of the outcome is modeled as a linear combination of the predictor variables. The log-odds are used because $X\beta$ can go from $-\infty$ to $+\infty$, while $p \in [0, 1]$.

$$\theta = logit(p) = \log \frac{p}{1-p} = X\beta$$

Applying the log-odds to p, the variable $\theta \in R$ and it is equal to 0 if p = 1/2. Log-odds are usually used in gambling, in which p represents the probability of an event happening (Ex. teams A win) and 1 - p of not happening. So if the

probability of a team winning is 1/3, the odds are 1/2, which corresponds to a possible return of the double of the money gambled (plus the money originally gambled) if the odds are fair.
From the logit formula, we can find p:

$$p = \frac{1}{1 + e^{-X\beta}}$$



To minimize the misclassification rate, we predict Y=1 when p >= 0.5 and Y = 0 when p < 0.5. This means guessing 1 when $X\beta$ is non-negative and 0 otherwise.

Because logistic regression predicts probabilities, we can use maximum likelihood to find the optimal β.

$$l(\beta) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i} = \prod_{i=1}^{n}(\frac{p_i}{1-p_i})^{y_i}(1-p_i) =$$

$$= \prod_{i=1}^{n} exp(y_i \log\frac{p_i}{1-p_i} + \log(1-p_i)) = exp(\sum_{i=1}^{n} y_i \log\frac{p_i}{1-p_i} + \log(1-p_i))$$

Considering log-likelihood and that $\theta = logit(p) = x_i\beta$

$$ll(\beta) = \sum_{i=1}^{n} y_i\theta_i + \log(\frac{1}{1+e^{-\theta_i}}) = \sum_{i=1}^{n} y_i x_i\beta + \log(\frac{1}{1+e^{-x_i\beta}})$$

We are not going to be able to set this to zero and solve exactly. However we can approximately solve it numerically

5.2 Random forest

A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique

that combines many classifiers to provide solutions to complex problems. A random forest algorithm consists of many decision trees. In the case of the random forest classifier, the decision for the classification of new data is made by majority voting. This method is also called bagging.
Decision trees usually work top-down, by choosing a variable at each step that best splits the set of items. There are various measures to know which is the best feature to pick for the splitting. One of the most popular one is the Gini index.

Gini index for a given node t:

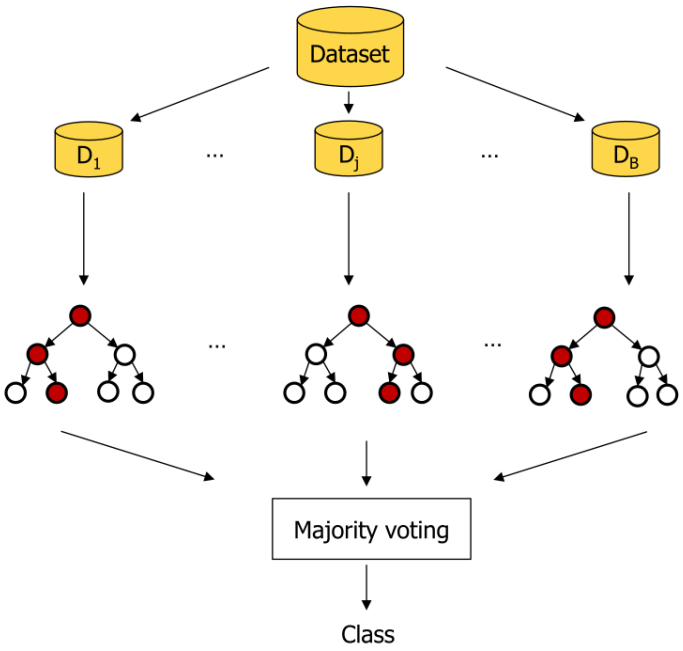
$$GINI(t) = 1 - \sum (p(j|t))^2$$

where $p(j|t)$ is the relative frequency of class j at node t
When a node p is split into k partitions (children), the quality of the split is computed as

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

The feature with the lowest $Gini_{split}$ will be chosen.

The nodes are expanded until all leaves are pure or until all leaves contain less than a certain amount of samples defined by an hyperparameter.
Each tree does not work on the original dataset, but it samples it with raplacement obtaining the same number of data of the original. Moreover it does not consider all the features, but also in this case a subset of it (Usually $\sqrt{p}$, with p = total number of features). Feature subsets are sampled randomly, hence different features can be selected as best attributes for the split in different trees.
Random forest are more precise and more robust to noise compared to decision trees. However the prediction is not interpretable (The choice may be given by hundred of trees).



5.3 K-nearest neighbor

A k-NN classifier is among the simplest machine learning algorithms. The classifier does not need training and does not build a model. However to be executed the entire dataset is needed. The idea is to predict the label of new data based on the data that are "similar" in the dataset. This method is based on the assumption that the data which are "similar" will have the same label. The similarity between two records is often calculated using the Euclidean distance, but there are many other options such as Minkovsky, Manhattan and Chebyshev distances. Notice that the Euclidean distance is a particular case of the Minkowsky distance. The Minkowsky distance can be defined as:

$$d(x_i, x_j) = (\sum_{i=1}^{d} (x_i - x_j)^p)^{\frac{1}{p}}$$

In the case of p = 2 we have the Euclidean distance.

6 Metrics and evaluation

To evaluate how good our models are we used the f_1 score, which is the harmonic mean of the precision and of the recall. The f_1 score $\in [0, 1]$, where a score of 1 means a perfect classifier and a score of 0 a really bad classifier.

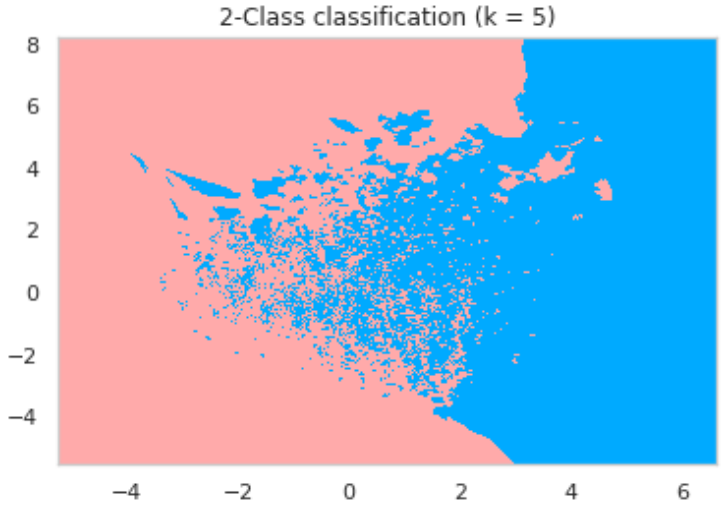


Figure 3: k-NN classification plot in 2 dimension

In particular, the accuracy for a class c is a measure that describes the fraction of records correctly assigned to the class c over the records that really belong to the class c. Instead, the precision for a class c is a measure that describes the fraction of records correctly assigned to the class c over the records that have been assigned to the class c (So including the records not assigned correctly) In formula:

$$Recall(c) = \frac{TP}{TP + FN}$$

$$Precision(c) = \frac{TP}{TP + FP}$$

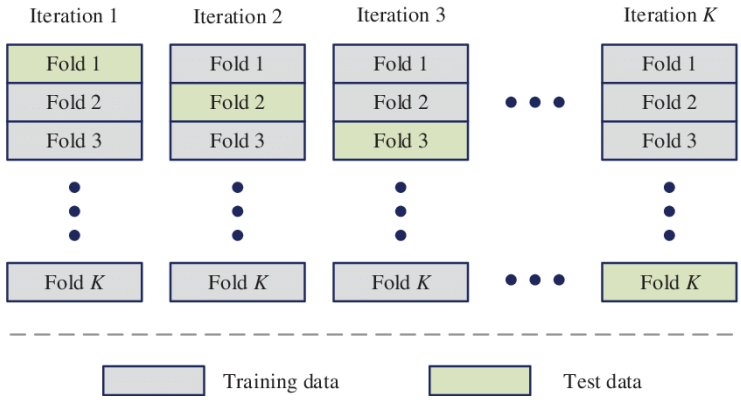
$$F_1(c) = \frac{2 * Recall * Precision}{Recall + Precision}$$

The confusion matrix is a visualization tool that permits to see in a graphical way the performance of a model.

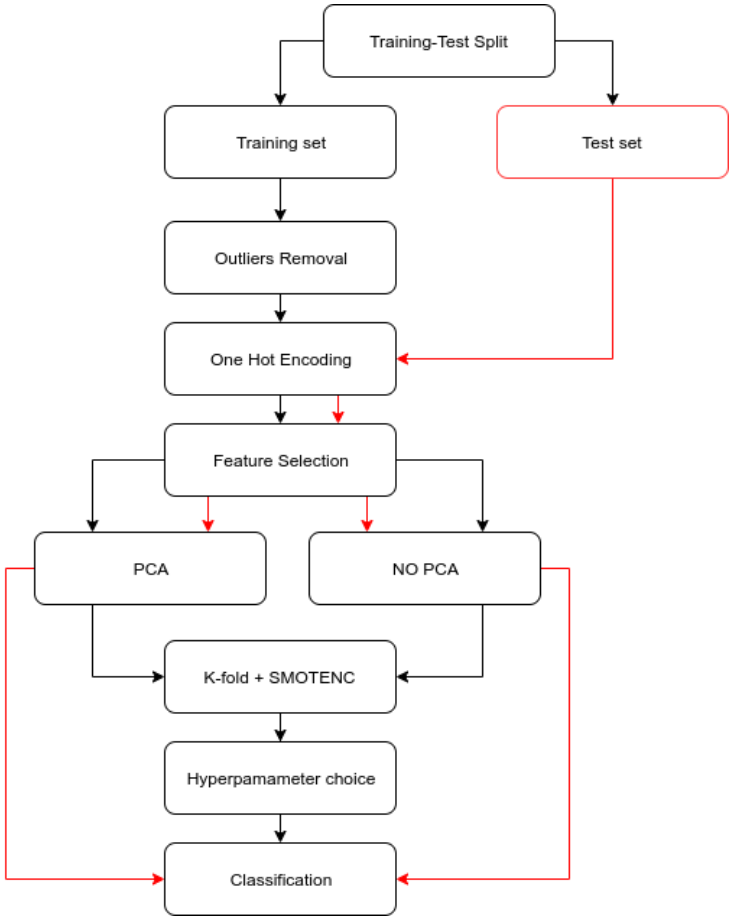
		Predicted class	
		Positive	Negative
Actual class	Positive	**True positive (TP)**	**False Negative (FN)**
	Negative	**False positive FP)**	**True positive (TN)**

7 Model selection

As we said before, if we tested our model on the same dataset that we used for training, the model created would overfit on the data and it would perform poorly on new set of data. One way to solve this problem is to split the dataset in three parts: the training set, the test set and the validation set. The training set is the part of the dataset which is used to train the model, while the validation set is the part that is used to choose the best hyperparameters for our model. Lastly the test set is the one used to test the model. One of the most common choices is a 70/15/15 split
However, the choice of the split is arbitrary, and it can lead to poor results. For example it could happen that all the records that belong to a class are in the evaluation split. This is the reason that we didn't adopt a train/validation/test approach, but a k-fold CV approach.
In k-fold we divide our training set in k equal parts, we train our model on k - 1 parts and we evaluate on the part that was excluded from the training phase. This procedure is repeated k times, in order that each part is used as a evaluation set exactly one time. Then we take the average of all our k evaluation metrics, and what we obtain is the final performance measure with a particular set of hyperparameters. We will choose the best performing parameters for each model and then we will test on the original test set
To try all different combination of hyperparameters we used a *GridSearchCV*, which is a Python class that permits to easily test different hyperparameters for a classification model and pick the one which performs the better.



7.1 Schema



8 Results

Now that we have explained every classification algorithm that we used, we are going to show the results. We have tested the Logistic regression classifier, the Random Forest classifier and the K-Neighbors classifier on both data before and after PCA.

Logistic regression:

Attribute	values
max_iter	{ 100,200 }

Random Forest classifier:
K-Neighbors classifier:

Attribute	values
min_samples_split	{ 2,4,8 }
n_estimators	{ 50,100 }

Attribute	values
p	{ 1,2,3 }
n	{ 5,10,15 }

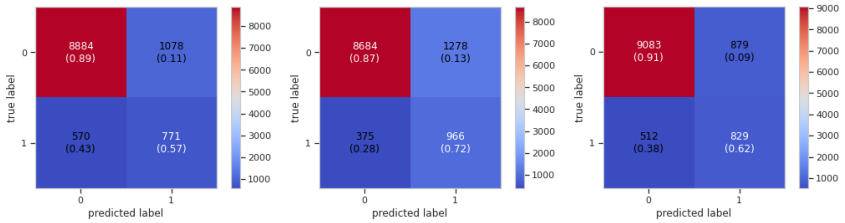
After the k-fold evaluation, the hyperparameters that were chosen were:

- Logistic regression
  - max_iter: 100
- Random Forest classifier
  - min_samples_split: 2
  - n_estimators: 100
- K-Neighbors classifier
  - p: 2
  - n: 5

The result after PCA dimensionality reduction were:

Classifier	Accuracy	Precision	Recall	F1
K-Neighbors	0.85	0.41	0.57	**0.48**
Logistic regression	0.85	0.43	0.72	**0.54**
Random forest	0.87	0.48	0.61	**0.54**

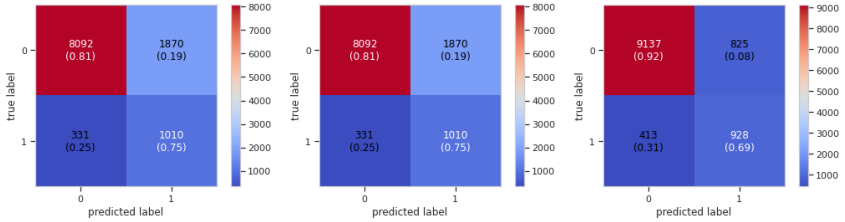
Table 4: Result with PCA



Instead, the results without applying PCA were:

Classifier	Accuracy	Precision	Recall	F1
K-Neighbors	0.80	0.35	0.75	**0.47**
Logistic regression	0.85	0.73	0.73	**0.53**
Random forest	0.89	0.53	0.65	**0.59**

Table 5: Result without PCA



As excepted, for the K-neighbors the accuracy is better after applying PCA, being the only algorithm out of the three that uses distances. In all 6 cases the precision is a lot lower than the recall, highlighting the fact that in the test set there are probably way more negative samples and that many of them have been predicted positive incorrectly. Because of the imbalance in the test set, the f1-score is the best metric in this case, and the random forest is the most performing model with an f1-score of 0.59 in the case without dimensionality reduction.

9 References

1. N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, SMOTE: Synthetic Minority Over-sampling Technique

2. Shai Shalev-Shwartz, Shai Ben-David: Understanding Machine Learning:From Theory to Algorithms

3. Cosma Rohilla Shalizi: Advanced Data Analysis from an Elementary Point of View