POLITECNICO DI TORINO

Mathematics in Machine Learning

Wine Quality Analysis

Professors

Prof. Francesco VACCARINO

Prof. Mauro GASPARINI

Alumn

Daniele GENTA

July 2020

Table of Contents

Chapter 1

Introduction

Rating a good wine based on previous labelled samples could be an interesting activity [1]. The aim of this project is to build such a Machine Learning model by using the Wine Quality data-set as a reference[1].
In particular a complete data processing pipeline has been implemented from data exploration and preprocessing to the actual classification algorithms able to label unseen instances of the data-set.

1.1 Dataset

The Wine Quality data-set is made of two different parts regarding respectively the red and the white wines. This thesis considers the union of the two data-sets and in particular a new categorical attribute has been added in order to make the model sort of task-aware (red/white).
So the data-set is made of 13 features and 6497 total samples. The inputs include objectivity tests, *e.g. pH values*, whereas the output is based on sensory data, i.e. median of at least 3 evaluations made by wine experts.
Each expert graded the wine quality between 0 (very bad) and 10 (excellent).

1.2 Environment

This thesis makes primarily use of Python to implement the different Machine Learning techniques of interest. In particular, some specific libraries have been used such as: Sklearn, Pandas, Numpy, Matplotlib and Seaborn.

[1] https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Chapter 2

Exploratory Data Analysis

The goal of this chapter is to perform a preliminary data exploration in order to become familiar with the data-set and its characteristics. In particular, the exploration is focused on understanding the features of the data-set and their distributions, the balance of the target column, the correlation between features and the possible presence of outliers.

2.1 Data-set description

Firstly the two data-sets, corresponding to red and white wines, are read.
Then, those are merged and a categorical attribute is created in order to distinguish between red and white wines: *type*. In fact, merging the two data-sets without keeping trace of the origin could lead to errors since white and red wines might have different scales and results.
All the attributes, excluding *type*, are numerical. Using some basic commands from Pandas allow to print, in the following table, the schema of the data-set, Table 2.1, and an example of it, Figure 2.1.

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	red	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	red	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	red	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	red	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	red	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Figure 2.1: Example of the data-set content

Attribute	Type	Description
Type	Categorical	Red or white
Fixed acidity	Numerical	fixed acids include tartaric, malic, citric, and succinic acids which are found in grapes
Volatile acidity	Numerical	These acids are to be distilled out from the wine before completing the production process
Citric acid	Numerical	This is one of the fixed acids which gives a wine its Freshness
Residual sugar	Numerical	This typically refers to the natural sugar from grapes which remains after the fermentation process stops, or is stopped.
Chlorides	Numerical	Chloride concentration in the wine is influenced by terroir and its highest levels are found in wines coming from countries where irrigation is carried out using salty water or in areas with brackish terrains
Free sulfur dioxide	Numerical	This is the part of the sulphur dioxide that when added to a wine is said to be free after the remaining part binds. Winemakers will always try to get the highest proportion of free sulphur to bind. They are also known as sulfites and too much of it is undesirable and gives a pungent odour
Total sulfur dioxide	Numerical	This is the sum total of the bound and the free sulfur dioxide. This is mainly added to kill harmful bacteria and preserve quality and freshness. There are usually legal limits for sulfur levels in wines and excess of it can even kill good yeast and give out undesirable odour
Density	Numerical	This can be represented as a comparison of the weight of a specific volume of wine to an equivalent volume of water. It is generally used as a measure of the conversion of sugar to alcohol
pH	Numerical	Also known as the potential of hydrogen, this is a numeric scale to specify the acidity or basicity the wine. Fixed acidity contributes the most towards the pH of wines
Sulphates	Numerical	These are mineral salts containing sulfur
Alcohol	Numerical	It's usually measured in % vol or alcohol by volume (ABV).
Quality	Numerical	Score between 0 and 10

Table 2.1: Data-set features description

2.2 Data-set exploration

2.2.1 Type balance

The aim of this section is to point out the distribution of red and white wines. Please note that this is important since *type* is a categorical attribute created to distinguish the two data-sets that have been merged, since wines belonging to the two categories might have different peculiarities.

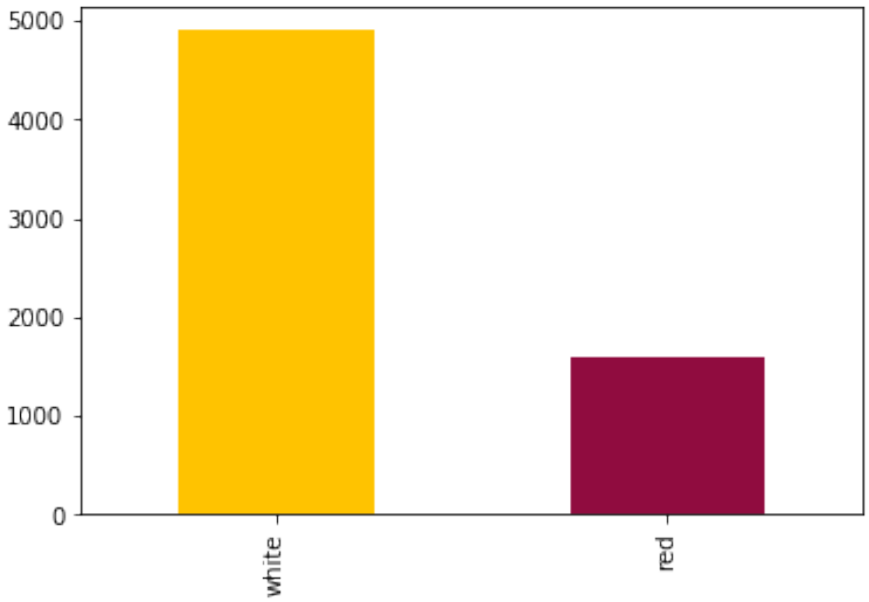

Figure 2.2: Red/white distribution

It can be seen from Figure 2.2 that the data-set is very unbalanced towards white wine records, which are 75% of the total. Methodologies to address this issue will be presented in the Data Preprocessing chapter.

2.2.2 Quality balance

The aim of this section is to investigate the distribution of the target column: *quality*. In particular, quality is a score between 0 and 10 that describes the goodness of a certain wine according to the average of three scores given by experts. It can be seen from Figure 2.3 and Table 2.2 that the data-set is very unbalanced towards the middle classes.
A target attribute which is not equally distributed is not desirable since it could lead to a bias and a poor generalization on the test set. Moreover the plot highlights that some classes are completely unrepresented, e.g. the very high and low ones.

So, we could expect the model to have a very poor generalization on a test set which instead includes all the possible classes, i.e. scores between 0 and 10.
Further preprocessing to address these issues will be reported in the next chapter.

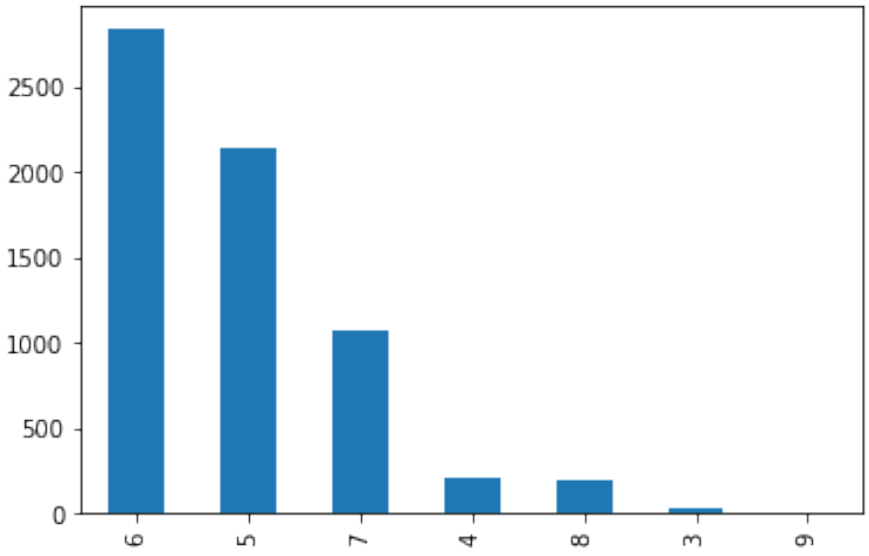

Figure 2.3: Quality distribution

Value	Count
6	2836
5	2138
7	1079
4	216
8	193
3	30
9	5

Table 2.2: Quality counts

2.2.3 Missing values

Missing values, i.e the lack of information in certain samples caused by different issues such as: sampling errors, defective instruments and actual lack of knowledge, are not desirable because they make our data-set unable to be fitted by Machine

Learning models.
Luckily, in this project the data-set does not contain missing values, Table 2.3, for each of its features. So, no samples need to be dropped and no values have to be filled in this phase.

Feature	Count Missing values
type	0
fixed acidity	0
volatile acidity	0
citric acid	0
residual sugar	0
chlorides	0
free sulfur dioxide	0
total sulfur dioxide	0
density	0
pH	0
sulphates	0
alcohol	0
quality	0

Table 2.3: Missing values counts per feature

2.2.4 Features distribution

Now the distribution of each feature is considered with respect to the different classes: below are reported a box plot and distribution plot for each feature in the data-set, Figure 2.4, 2.5. Please note that, in both figures, a *binning* has already been applied to make the classes more representative.
As previously said, the bin implementation will be discussed in detail in the following chapter.
Looking at the box plots it can be seen which features have got more outliers and how different features could impact the score, moreover it is immediate to recognize the separation in terms of features' value among the different classes.
We're interested in the features whose pair of box plots is well separated because these features will be the ones impacting the model the most, e.g. *alcohol, volatile acidity*.
Both the distribution and the box plots allow to understand which is the domain in which the features live, this is a starting point to the normalization step carried out

in the next chapter, in this data-set in fact some features have a different domain in terms of order of magnitude.

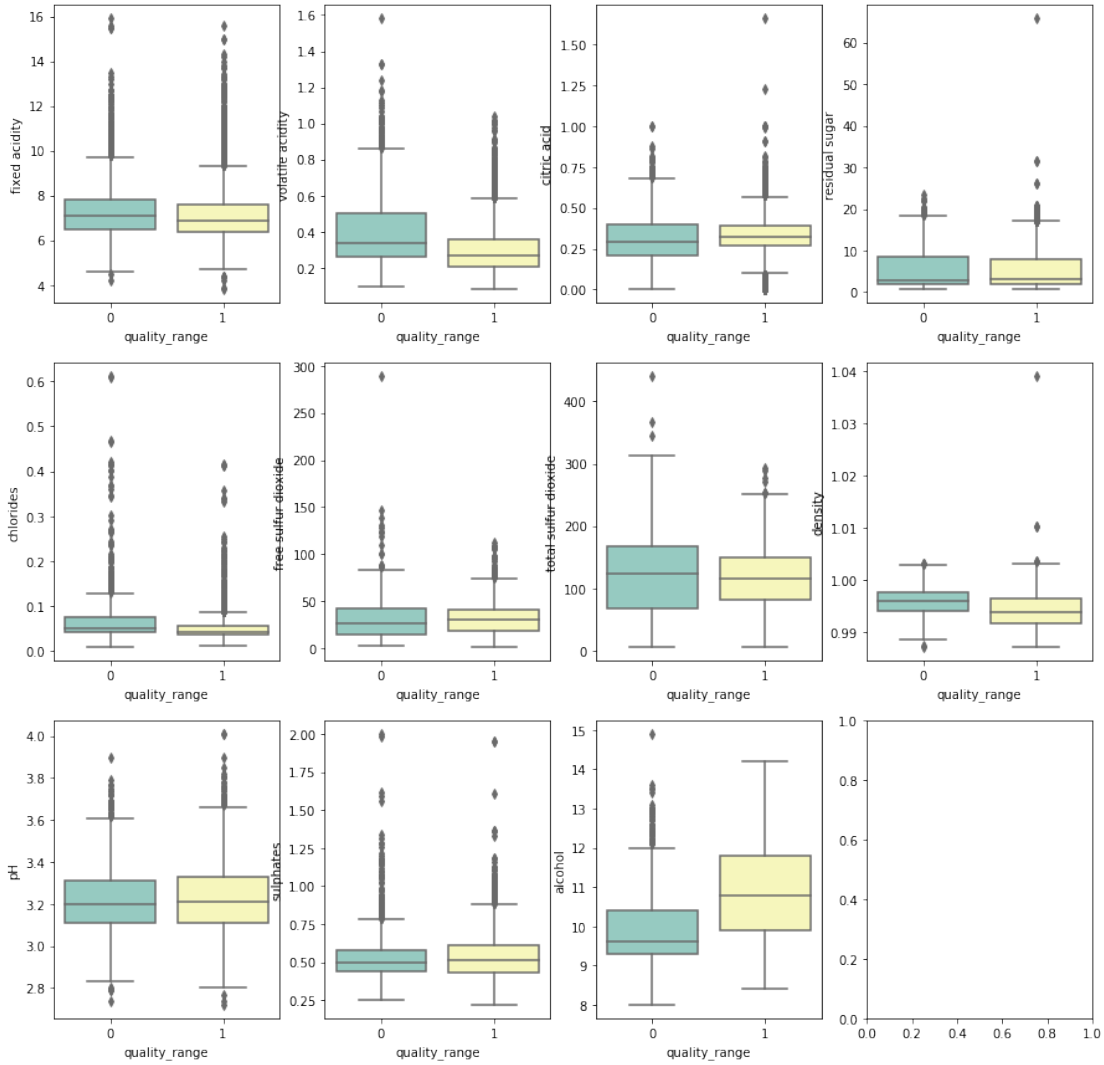

Figure 2.4: Features distribution - boxplots

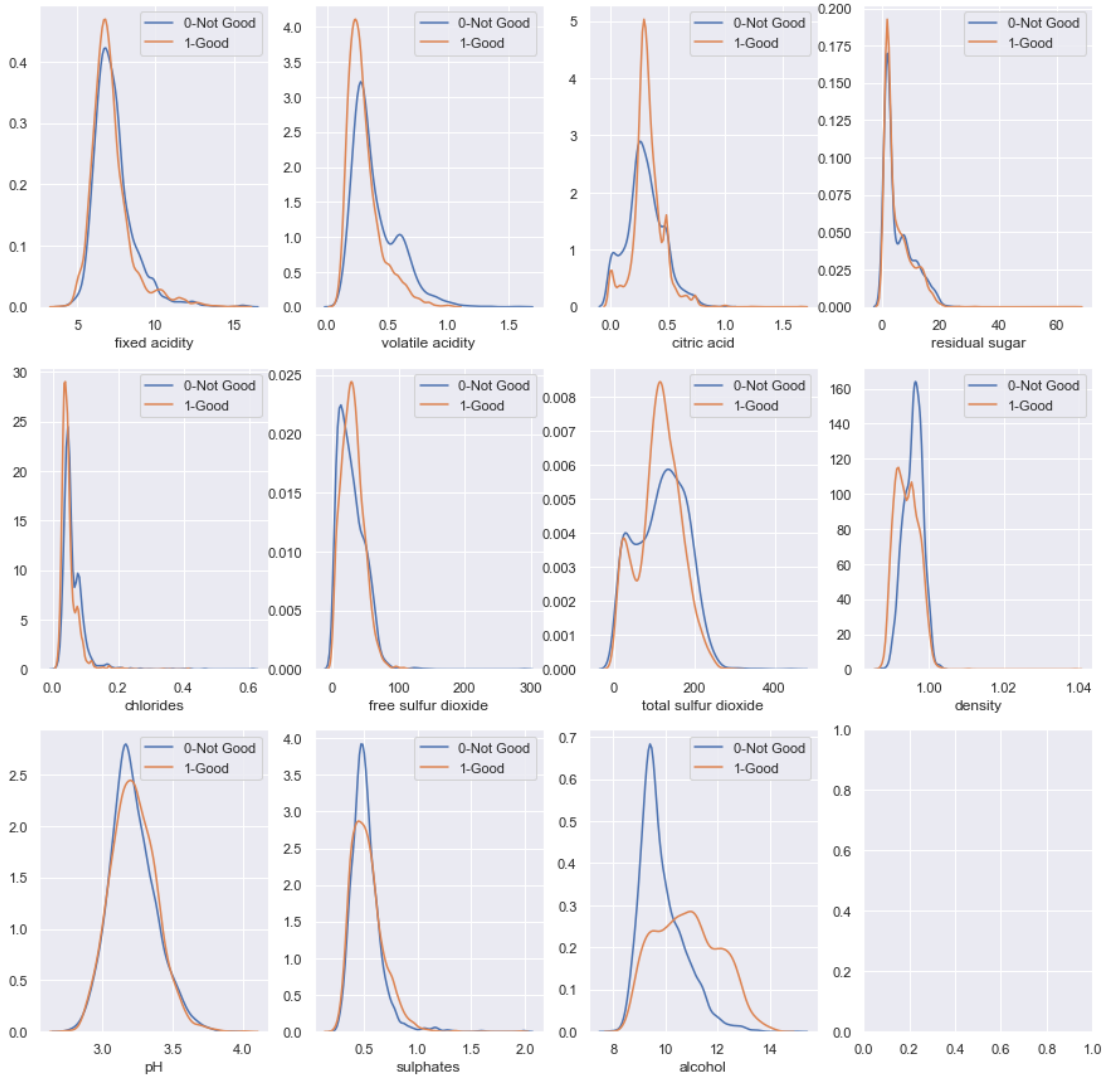

Figure 2.5: Features distribution - distplots

2.2.5 Features correlation

Another important step is to check the correlation of each pair of features in the data-set. Strongly correlated features can make our models overfit the training set. The problem is typically solved using some *dimensionality reduction* or *features selection technique*, which helps reducing both the overfitting risk and the training complexity. In the following plot, Figure 2.6, we can investigate the features' correlation in our training set.
Please note that features that are strongly correlated will achieve an absolute correlation score close to one, whereas if the same score is < 0.5 we usually do not talk about correlation.
Correlation is a number in the interval [-1, 1] that indicates, for each pair of features, the degree of relationship between them. It is computed as reported in Equation 2.1.

$$\rho_{x,y} = corr(X,Y) = \frac{cov(X,Y)}{\sigma_x \sigma_y} = \frac{E[(X-\mu_x)(Y-\mu_y)])}{\sigma_x \sigma_y} \tag{2.1}$$

As we can see from the heat-map associated with the correlation matrix, Figure 2.6, most of the features are not strongly correlated whereas *total sulfur dioxide* has the highest correlation with *free sulfur dioxide* and also *density* and *residual sugar* have got a meaningful positive correlation.
Please note that in the figure it has been reported the lower triangular portion of the symmetric matrix due to better interpretability.
In the Data Preprocessing chapter the above considerations will be addressed through features selection and dimensionality reduction.

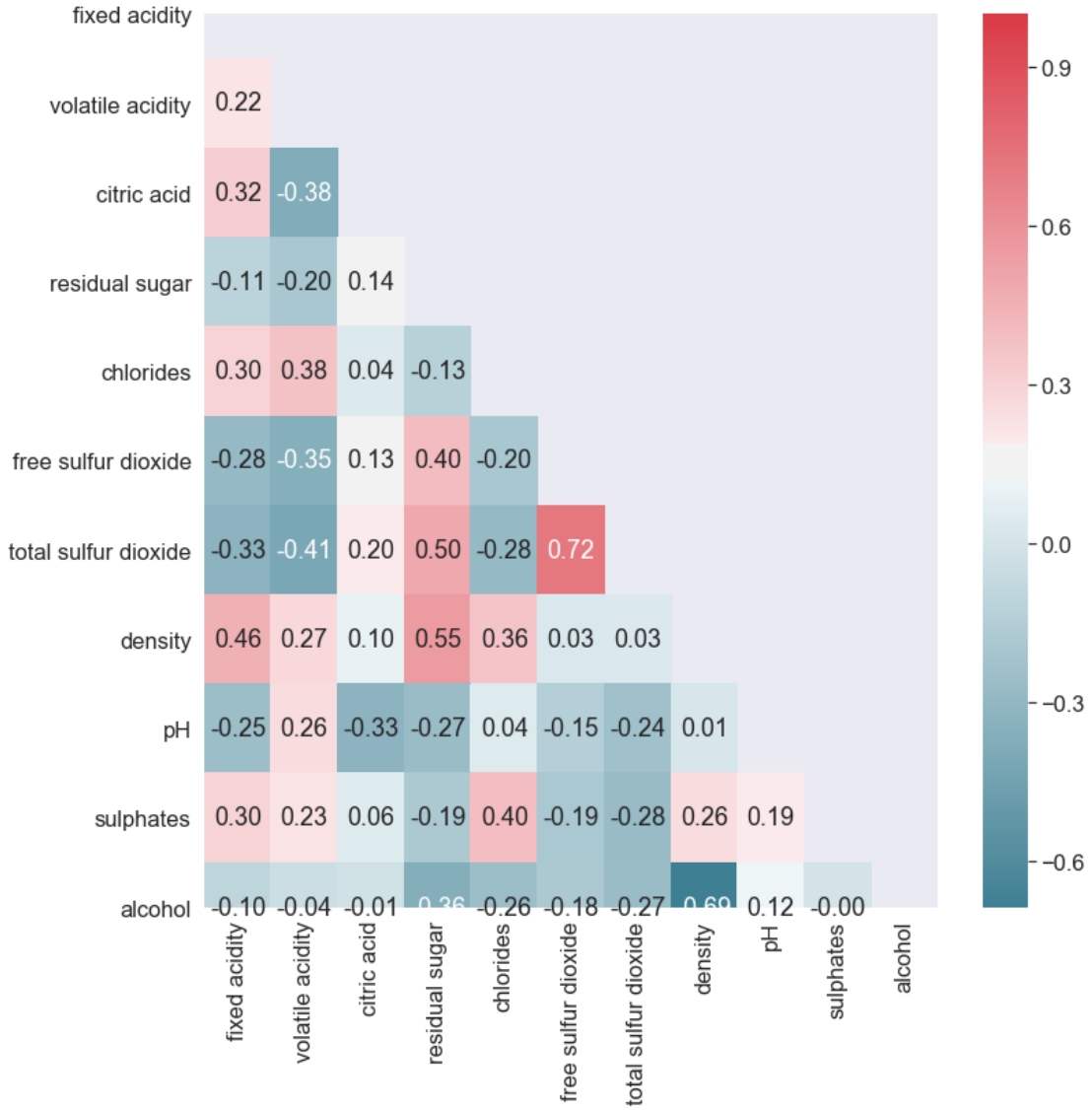

Figure 2.6: Features correlation

Chapter 3

Data Preparation - Preprocessing

The goal of this chapter is to exploit different preprocessing methodologies in order to obtain a data-set made of meaningful features for the ML models that will be implemented in the following chapters.
It could be argued that data preprocessing is one of the most important steps in a data processing pipeline, according to the mantra *"garbage in, garbage out"*.

3.1 Feature selection

Firstly, we want to simplify the model by removing the feature *free sulfur dioxide*, since it has a very high correlation with another feature, *total sulfur dioxide*, and does not add informational value to our model.
This is done by using the following line of code, in Python:

```
1    dataset.drop(['free sulfur dioxide'], axis=1)
```

3.2 Type encoding, balancing

The type of wine has been implemented as a categorical attribute whose values are *"red"* or *"white"*, in order to use it in the following Machine Learning algorithms we need to make it numerical.
This is done by using a simple encoding that produces whether a *zero* or a *one* in correspondence to the original value of *type*.
The line of code used and the result are reported below, Figure 3.1.

```
1    # encode categorical value
2    encoded_dataset.type = dataset.type.map({'white':0, 'red':1})
```

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates	alcohol	quality_range
0	1	7.4	0.70	0.00	1.9	0.076	34.0	0.9978	3.51	0.56	9.4	0
1	1	7.8	0.88	0.00	2.6	0.098	67.0	0.9968	3.20	0.68	9.8	0
2	1	7.8	0.76	0.04	2.3	0.092	54.0	0.9970	3.26	0.65	9.8	0
3	1	11.2	0.28	0.56	1.9	0.075	60.0	0.9980	3.16	0.58	9.8	1
4	1	7.4	0.70	0.00	1.9	0.076	34.0	0.9978	3.51	0.56	9.4	0

Figure 3.1: Example of the data-set after performing one hot encoding on type

As we have seen in Figure 2.2, the data-set is very unbalanced towards white wines (75% of instances belong to white wines).
Since this is the result of the union of two data-sets we'd like to have a coherent baseline that is made of approximately the same number of samples among the two types.
The rebalancing problem could be addressed in different ways such as *under-sampling* and *over-sampling*. The first one reduces the number of records of the majority class, while oversampling produces several synthetic records from the data distribution of the minority class in order to match the number of records of the majority class or just resamples from the training set instances belonging to the minority class, Figure 3.2.

In our case, we could think that the data-set is big enough and apply random sub-sampling of the majority class has been used. Operatively, it consists of a sampling without replacement of n records belonging to the majority class, where n is the number of records belonging to the minority class.
As it will be demonstrated through experiments in the next chapter, performing under-sampling in this specific problem actually reduces the accuracy of our models, this could be explained as a loss of valuable data that makes the classifier weaker.
Therefore a random over-sampling has been implemented as well as an alternative. It consists in randomly sampling instances from the minority class in order to enlarge its size.
As a result, in both cases, we have a data-set which has the same number of samples among the two types, i.e. *red* and *white*.

Figure 3.2: Oversampling and Undersampling schema

3.3 Quality binning, balancing

As we observed in Figure 2.3, the classes (quality scores) are highly unbalanced and certain values do not even appear (e.g. 0, 1, 10). Using this kind of target would lead to a biased model that is not able to generalize and recognize all classes since it has been trained just on a subset of them.
In order to address the issue, it has been decided to apply a binning strategy to reduce the problem to a binary classification one, Figure 3.3.
In particular, scores between 0 and 5 are considered as *not good* whereas scores from 6 to 10 are considered as *good*. Different types of binning could have been implemented as alternatives.

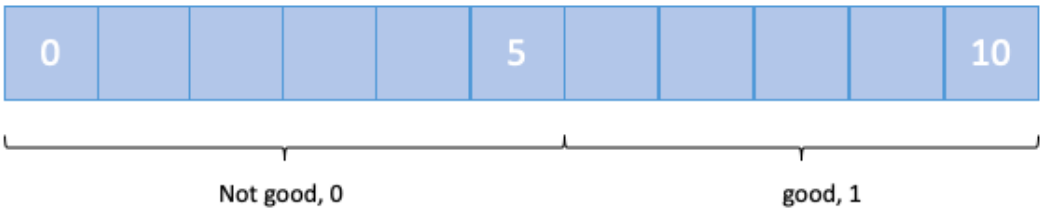

Figure 3.3: Quality scores binning

Unfortunately by printing again the class distribution with respect to the binned data-set, Figure 3.4, it can be seen that the quality scores binned into 0 (not good) and 1 (good) follow again an unbalanced distribution.
Again, an undersampling/oversampling should be performed in order to balance the classes.

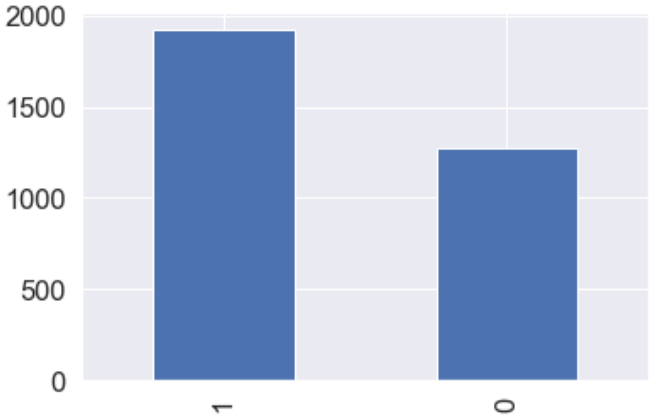

Figure 3.4: Quality scores binning distribution - before balancing

3.4 Normalization and Standardization

Looking at the distribution plots regarding the features, Figure 2.5, it can be seen that the different features live in very heterogeneous intervals.
Machine Learning algorithms fed with features having a different scales could result in bias and errors, hence both a normalization and a standardization are performed on each numerical feature, i.e. all the features except the categorical *type* and the target *quality*.
A first approach to address normalization might be to map each feature's interval of each sample to the interval [0, 1], using the following formula, Equation 3.1

$$x_{norm} = \frac{x}{||\mathbf{x}||_2} \tag{3.1}$$

Standardization, instead, considers each feature's distribution and transforms it to have zero mean and unitary standard deviation. In particular, for each sample it is subtracted for each feature the mean of that specific feature, the result is divided by the standard deviation of the feature. Standardization is the method that was implemented in this project, Equation 3.2, using the *StandardScaler* class of Sklearn. A standardized example of the data-set is reported in Figure 3.5.

$$\begin{aligned} z &= \frac{x - \mu}{\sigma} \\ \mu &= \frac{1}{n}\sum_i x_i \\ \sigma^2 &= \frac{1}{n}\sum_i (x_i - \mu)^2 \end{aligned} \tag{3.2}$$

Finally, standardizing the features of the data-set is the first step towards performing dimensionality reduction using PCA, that will be presented in the following section.

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates	alcohol
count	11942.000000	1.194200e+04	1.194200e+04	1.194200e+04	1.194200e+04	1.194200e+04	1.194200e+04	1.194200e+04	1.194200e+04	1.194200e+04	1.194200e+04
mean	0.513649	1.308988e-16	1.332788e-16	-1.427987e-16	4.759958e-18	-1.427987e-16	-4.164963e-17	1.947775e-14	-1.005541e-15	2.570377e-16	5.521551e-16
std	0.499835	1.000042e+00	1.000042e+00	1.000042e+00	1.000042e+00	1.000042e+00	1.000042e+00	1.000042e+00	1.000042e+00	1.000042e+00	1.000042e+00
min	0.000000	-2.449918e+00	-1.694830e+00	-1.786077e+00	-8.970258e-01	-1.240870e+00	-1.436428e+00	-2.917295e+00	-3.042015e+00	-2.076345e+00	-2.086914e+00
25%	0.000000	-6.497572e-01	-7.905798e-01	-5.300490e-01	-6.000798e-01	-5.344505e-01	-9.237518e-01	-6.478459e-01	-6.683277e-01	-6.678647e-01	-7.688995e-01
50%	1.000000	-1.997170e-01	-2.379822e-01	8.248912e-03	-4.858697e-01	-1.812410e-01	-6.377946e-02	1.579979e-01	-1.556354e-02	-1.396844e-01	-2.416935e-01
75%	1.000000	3.789061e-01	7.165044e-01	6.063577e-01	3.136003e-01	2.823465e-01	7.465791e-01	6.676769e-01	6.372006e-01	3.884959e-01	6.369832e-01
max	1.000000	5.329349e+00	5.840591e+00	8.142528e+00	1.399596e+01	1.198241e+01	5.741034e+00	1.494558e+01	4.494443e+00	8.369887e+00	3.975954e+00

Figure 3.5: Standardized data-set properties

3.5 Dimensionality reduction: PCA

PCA (Principal Component Analysis) is an unsupervised learning methodology which, given a set of predictors X with a large number of correlated features p, aims to summarize this set with a smaller number of representative variables n that collectively explain most of the variability in the original set.
This is done by projecting the data to a n dimensional space, where n is the new number of dimensions chosen.
Each of the dimension found by PCA is called *principal component*.
In particular the first principal component found by PCA has the direction of largest variance in the data (minimum projection error), each further principal component will be orthogonal to the previous one and in the direction of maximum variance of the data in the residual subspace, Figure 3.6.

PCA Algorithm:
Start with
X: initial data-set
p: number of features in X
n: new number of features ($n < p$)
Do:
Standardize (or just center) the data: $z_i = \frac{x_i - \hat{\mu}}{\hat{\sigma}}$
Define the Scatter Matrix, $S = Z^T Z$
Calculate the eigen values and the eigen vectors of S through, for instance, Singular Value Decomposition (SVD)
Sort the eigen vectors according to the descending order of the eigen values
Define a new projection space W by concatenating as its columns the first n eigen vectors found
Project the data-set X onto the new space W, $Z_{new} = ZW$

When implementing PCA it is important to choose properly n, the number of principal components to consider. In order to do that we have to consider the trade-off between the total variance explained by the n principal components and the complexity of the problem.
Looking at Figure 3.7, it can be seen that in this case a good number of principal components might be 6, which explains $\sim 87\%$ of the total variance. Generally the decision is taken by looking at the curve's elbow.

So, performing PCA allows to summarize the potentially correlated features of the

original data-set in a smaller number of representative variables that explain most of the variance in the set, the cost is a minor interpretability.
Now that the data have been preprocessed, we're ready to apply different Machine Learning models and algorithms to carry out the classification, this will be addressed in the next chapter.

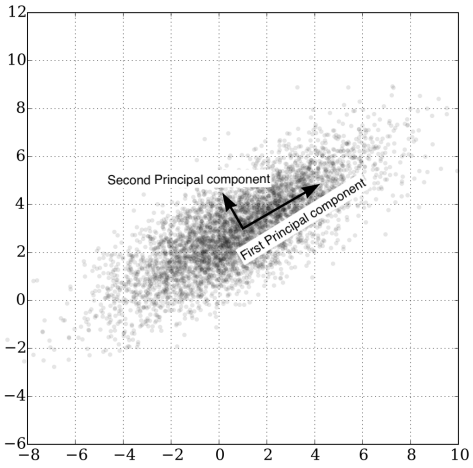

Figure 3.6: Basic PCA schema

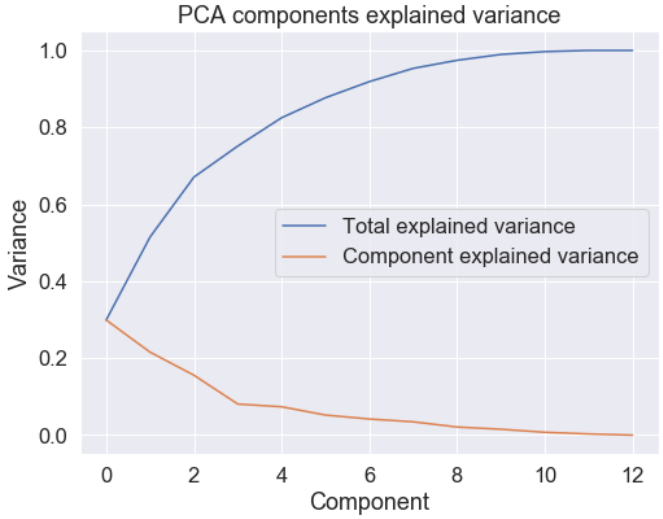

Figure 3.7: Explained variance per #components

Chapter 4

Model Selection

The goal of this chapter is to implement different classification models and evaluate their performances in order to assess which is the one that fits best this specific problem.
Firstly the data-set is randomly split into three subsets: training, validation and testing. Since each specific algorithm requires different parameters that may have a huge impact on their performances, a *grid search* is performed for each experiment in order to choose the best performing hyper-parameters on the validation set. Each model will be validated on the validation set (hyper-parameters tuning) and tested on the test set, in order to evaluate its goodness. This will be done by using different score metrics: *accuracy, recall, precision and F1-score*.
Each experiment will be carried out with and without PCA and by performing an over/under sampling or by keeping the default data-set distribution.

4.1 Train, Test and Validation

In order to evaluate the goodness of the models implemented in the next sections, it is needed to divide the data-set into a train, a test and a validation portion.
In order to create the test set a random splitting is performed, following the proportion 80/20, this set will give us the overall goodness and generalization ability of our model.
As for the validation set, it is used to tune the hyper-parameters of the model and provides an estimate for the test error. As for its creation, it could be achieved by using different techniques. A first naive approach might be to randomly split the train data-set into two portions by using a certain proportion, e.g. train 70% and validation 30% (*validation set approach*).
However, this results in a highly variable estimate for the test error, moreover just a subset of the training set is used to train the model.

The state of the art approach is *K-Fold Cross Validation*: the train set is further split into K folds, cyclically *K-1* are used to actually train the model whereas the other is used to validate it, Figure 4.1. The overall score achieved, i.e. the estimate for the test error, is the arithmetic average of the scores obtained at each of the K iterations. Finally the model is evaluated on the test set.
In this project K was set equal to 5.

Figure 4.1: K-Fold Cross Validation schema

4.2 Metrics

The starting point for evaluating the models presented in the following section is the *confusion matrix*, Figure 4.2.

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

Figure 4.2: Confusion Matrix Schema

From the Confusion Matrix different metrics are extracted: Equation 4.1, 4.2,

4.3, 4.4.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

$$precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{4.4}$$

Finally, AUC ROC (Area Under Curve Receiver Operating Characteristics) has been used, it basically presents how much the model is able to distinguish the two different classes using the True Positive Rate (TPR) and the False Positive Rate (FPR), Equation 4.5.

$$\begin{aligned} TPR &= \frac{TP}{TP + FN} \\ FPR &= \frac{FP}{FP + TN} \end{aligned} \tag{4.5}$$

4.3 Decision Trees and Random Forests

4.3.1 Decision Trees

Decision Tree is a supervised learning algorithm that aims to predict classes via segmenting the predictors space into a number of simple regions.
Decision Trees are very interpretable and easy to implement, however they're not competitive in terms of accuracy with the others supervised learning algorithms. *Bagging, Random Forests and Boosting* are therefore used to dramatically increase their performances. Owing to their huge limitations, single decision trees were not implemented in this project.

4.3.2 Random Forests

Random Forests, Figure 4.3, combine multiple decision trees in order to enhance the robustness and the accuracy of the model.
A first step might be to use *bagging* on Decision Trees: we simply train multiple Decision Trees on bootstrapped sets from the training set and assign the result via a majority voting approach.

Using *bootstrap* means that each of the new training sets created for the different decision trees is built by sampling with replacement n record from the original training set, where n is the total number of samples.
Random Forests provide an improvement by considering multiple *decorrelated* trees, this reduces the variance when we average the results.
In practice, as in bagging trees are built on bootstrapped training samples, however when a tree is built it considers, at each split, a *random selection of m predictors* out of the original n predictors (as a rule of thumb, usually $m = \sqrt{n}$).

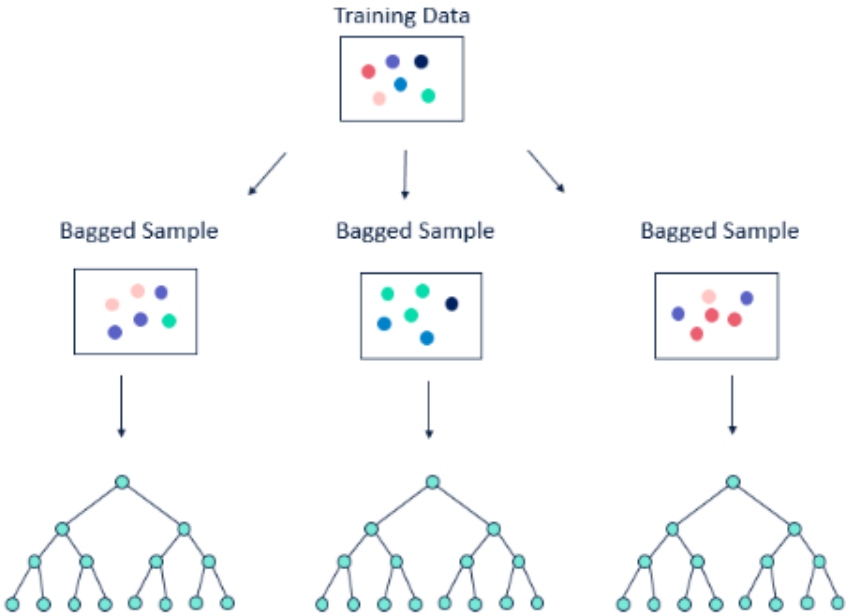

Figure 4.3: Random Forest - basic schema

As for deciding which feature of the data-set will be used for the split, decision trees use a recursive top-down greedy approach in which it is selected the *best feature*, i.e. the feature that maximizes the node purity. Usually two measures might be adopted:

$$GINI = 1 - \sum_{i=1}^{n} p_i^2 \tag{4.6}$$

$$Cross\_Entropy = -\sum_{i=1}^{n} p_i * log_2(p_i) \tag{4.7}$$

In this project both GINI and Cross Entropy were used, which are a measure of heterogeneity and separation; the lower the indexes the more the node is pure and

therefore the split is desirable.
Random Forests overcome most of the problems of Decision Trees such as their tendency to over-fit the training set and their low accuracy; in contrast they are less interpretable.
Below are reported the parameters being used in the grid-search, Table 4.1, the experiments and the corresponding results in terms of different metrics, Table 4.2, as well as the figures corresponding to these experiments in terms of *confusion matrices* and *ROC curves*, Figure 4.4, 4.5, 4.6, 4.7.

Name	Values
num. estimators	10, 50, **80**
criterion	GINI, **cross entropy**
max. depth	None, **20**, 50, 100
bootstrap	**True**, False

Table 4.1: Grid Search - Random Forest[1]

Model	Accuracy Test	Precision	Recall	F1
Original	0.805	0.868	0.828	0.848
Original + PCA	0.780	0.845	0.812	0.828
Undersampling + PCA	0.771	0.772	0.77	0.77
Oversampling + PCA	**0.964**	**0.957**	**0.969**	**0.963**

Table 4.2: Results Random Forest

[1]in bold are highlighted the best results achieved

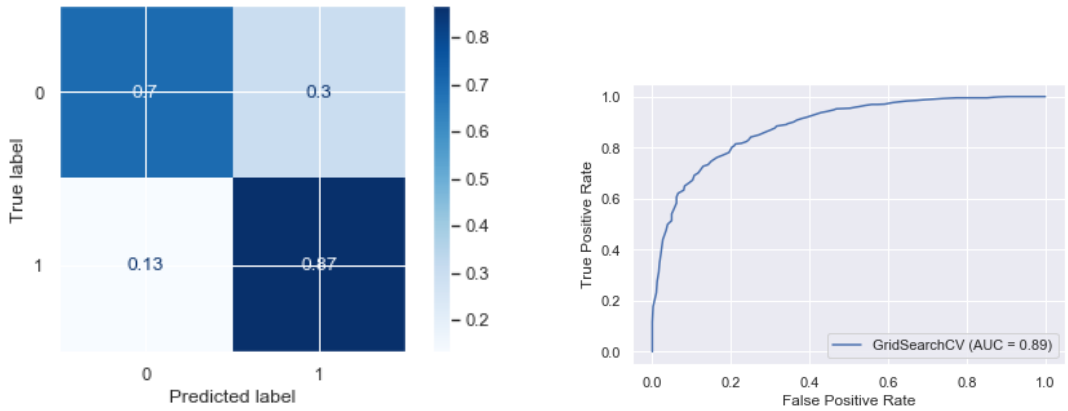

Figure 4.4: Original (no pca nor sampling) results - Random Forest

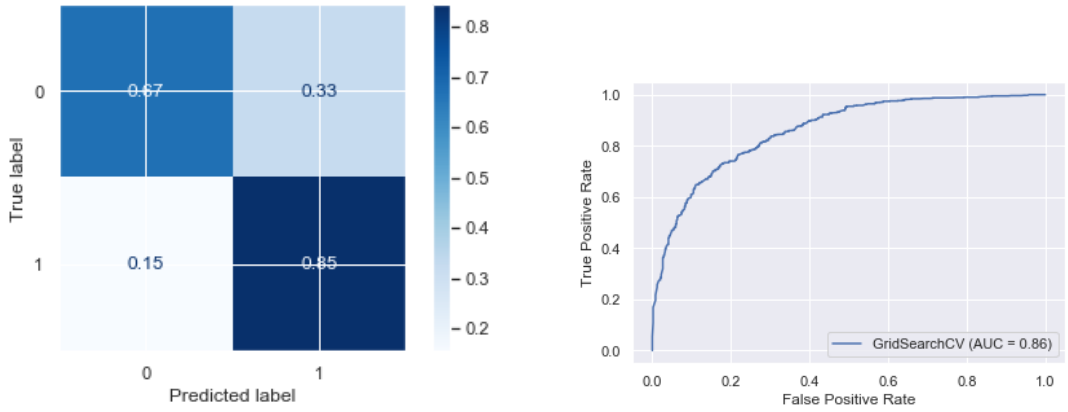

Figure 4.5: Original + PCA results - Random Forest

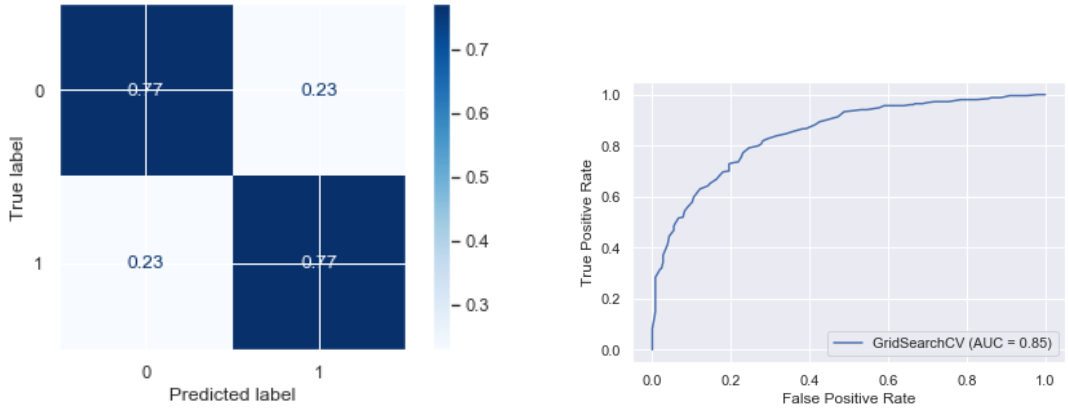

Figure 4.6: Under-sampling + PCA results - Random Forest

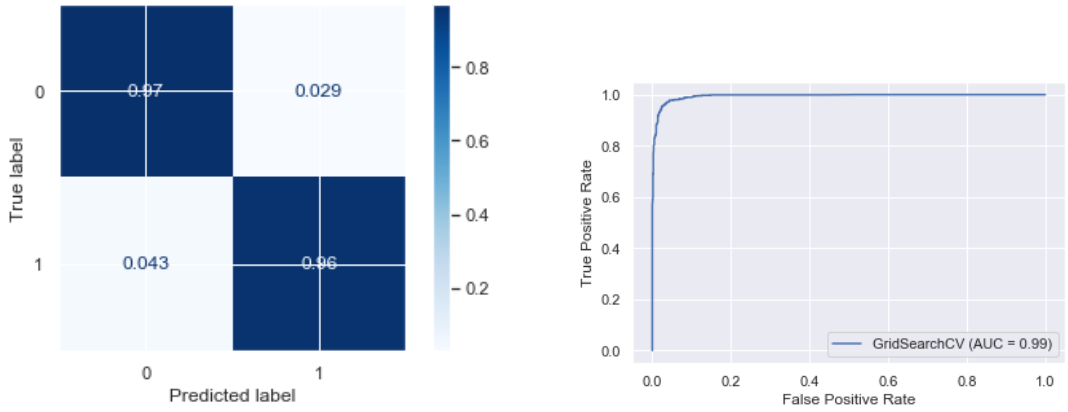

Figure 4.7: Over-sampling + PCA results - Random Forest

4.4 Logistic Regression

Logistic Regression, Figure 4.8, tries to classify instances by exploiting the probability that a given sample belongs to a certain class.
Putting ourselves in a binary classification setting, we could think of generalizing *linear regression* in order to express the results as a probability.
This could be obtained by applying a *sigmoid* function to the linear regression output, Equation 4.8, and secondly by replacing the Gaussian distribution for Y with a Bernoulli.

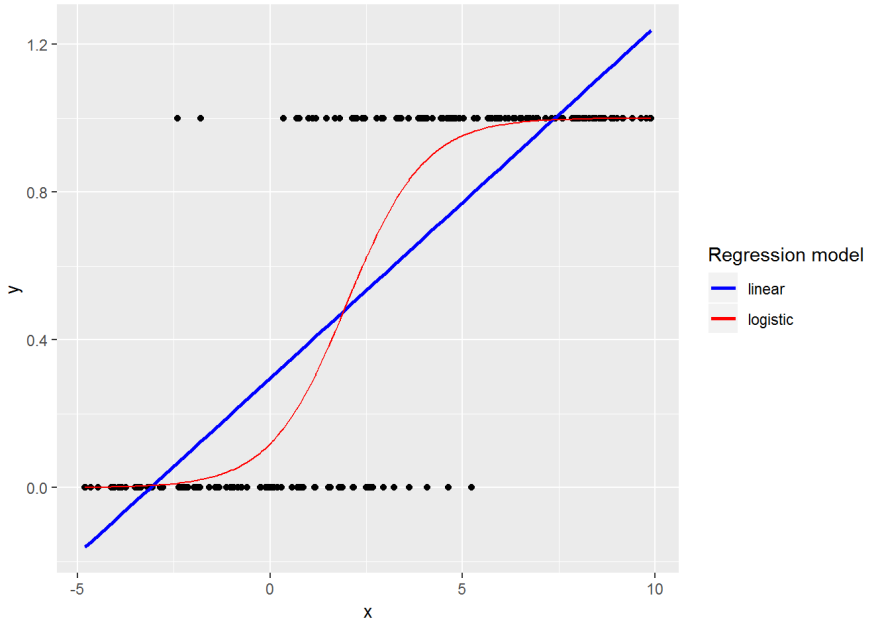

Figure 4.8: Logistic Regression v Linear Regression - basic schema

$$z(x_i) = \langle \mathbf{w}, x_i \rangle + b$$
$$p(x_i) = \frac{1}{1 + e^{-z(x_i)}} = \frac{e^{z(x_i)}}{e^{z(x_i)} + 1} = \frac{e^{\langle \mathbf{w}, x_i \rangle + b}}{e^{\langle \mathbf{w}, x_i \rangle + b} + 1} \tag{4.8}$$

Where $z(x_i)$ represents the linear regression prediction for x_i and $p(x_i)$ the logistic regression output, i.e. a probability. In a binary classification setting $p(x_i)$ corresponds to $p(x_i) = P(Y|X = x_i)$, hence the probability of getting a label Y given the sample x_i. The model is trained to estimate the parameters $\mathbf{w}$ and b, in particular this is done by maximizing the likelihood function: Equation 4.9.

$$l(\mathbf{w}, b) = \prod_{i}^{n} P(Y = y_i | X = x_i) = \prod_{i}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$
$$\max_{\mathbf{w},b} l(\mathbf{w}, b) \tag{4.9}$$

Below are reported the parameters being used in the grid-search, Table 4.3, the experiments and the corresponding results in terms of different metrics, Table 4.4, as well as the figures corresponding to these experiments in terms of *confusion matrices* and *ROC curves*, Figure 4.9, 4.10, 4.11, 4.12.
An important parameter to optimize is C which is inversely proportional to the strength of regularization (the smaller the C, the stronger the regularization), which aims at reducing the overfitting that might occur consequently to the maximization of the likelihood.

Name	Values
C	0.1, **1**, 10
penalty	l1, **l2**

Table 4.3: Grid Search - Logistic Regression

Model	Accuracy Test	Precision	Recall	F1
Original	0.73	0.834	0.757	0.795
Original + PCA	0.708	0.831	0.736	0.781
Undersampling + PCA	0.733	0.701	0.748	0.724
Oversampling + PCA	0.722	0.706	0.724	0.715

Table 4.4: Results Logistic Regression

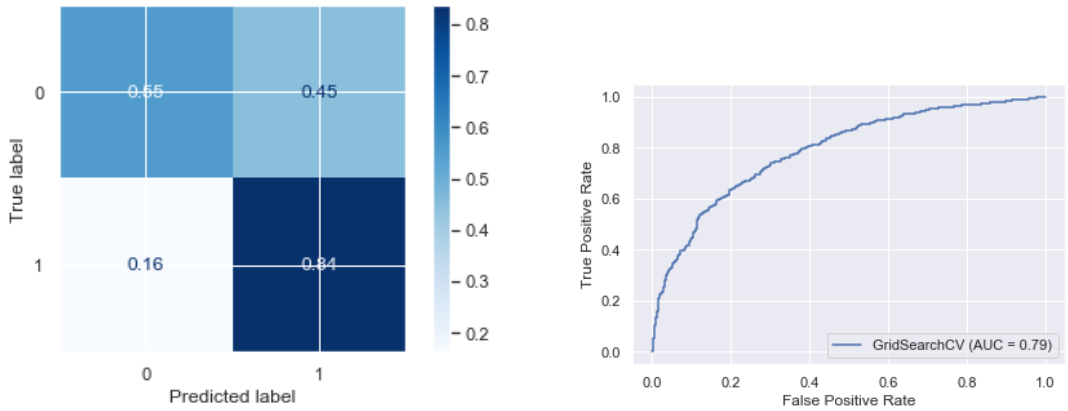

Figure 4.9: Original (no pca nor sampling) results - Logistic Regression

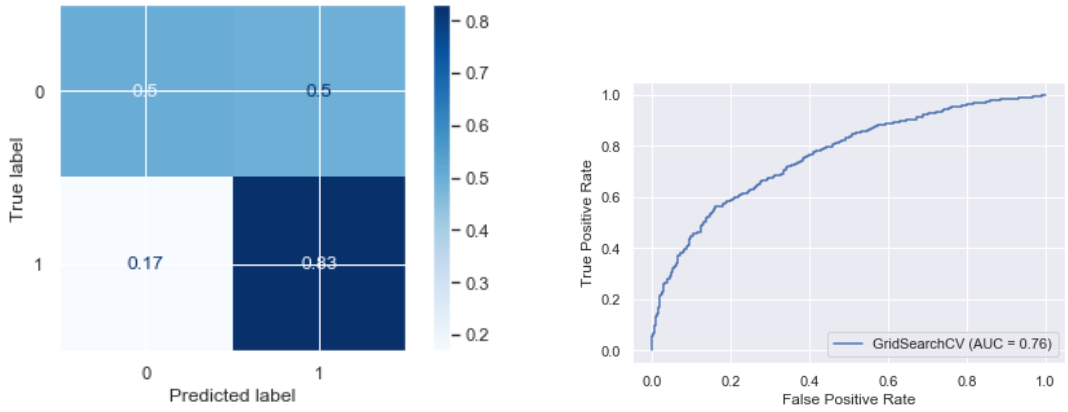

Figure 4.10: Original + PCA results - Logistic Regression

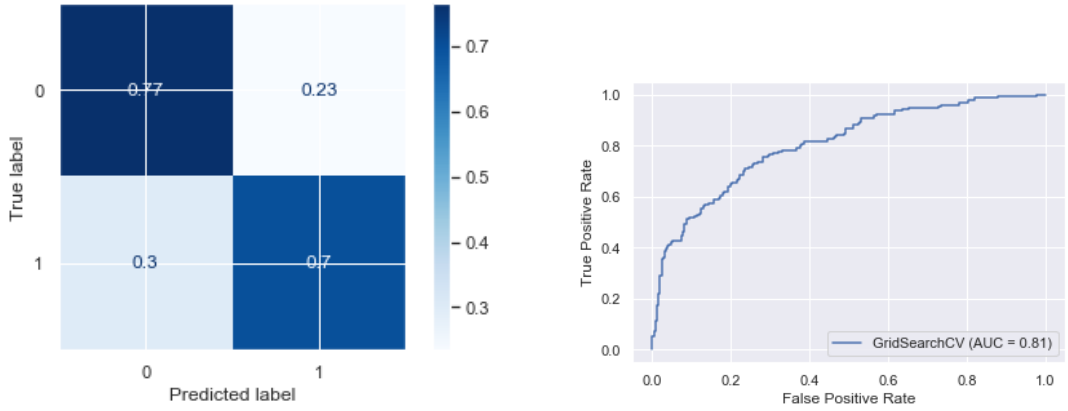

Figure 4.11: Under-sampling + PCA results - Logistic Regression

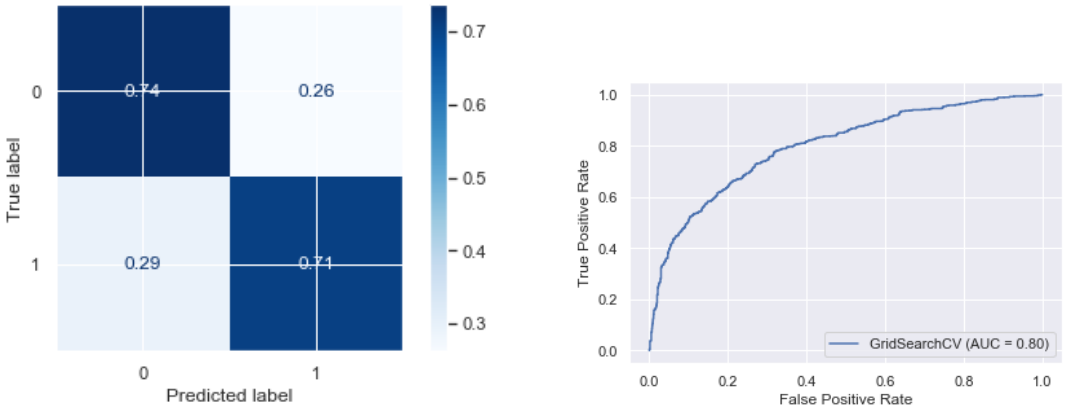

Figure 4.12: Over-sampling + PCA results - Logistic Regression

4.5 Support Vector Machines

Support Vector Machine (SVM), Figure 4.13, is a classifier which classifies instances by finding the best hyperplane that separates the classes in the feature space.

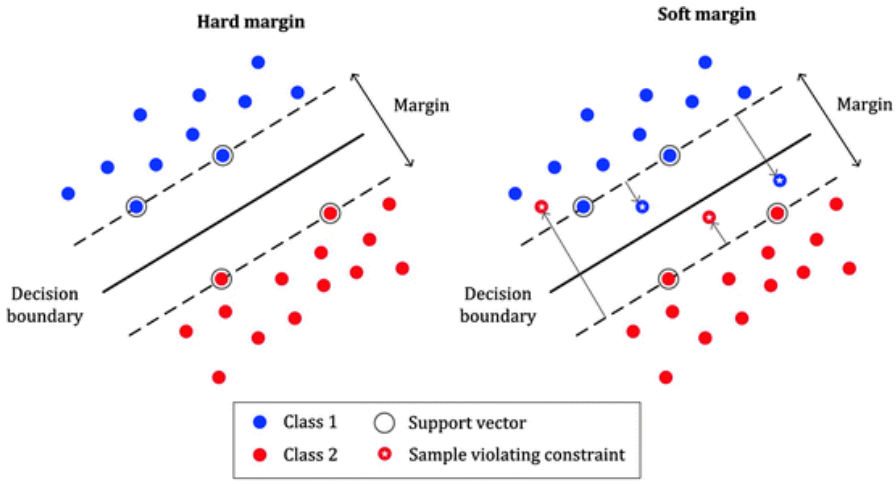

Figure 4.13: SVM - basic schema

If the dataset is linearly separable then the problem is to find the *maximum margin hyperplane*, Equation 4.10, this is called an *Hard Margin Classifier*.

$$\begin{aligned} \min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \\ s.t. \\ y_i(\langle \mathbf{w}, x_i \rangle + b) \geq 1, \forall i \end{aligned} \tag{4.10}$$

However the data-set might be not linearly separable, then we have to accept some mistakes, Equation 4.11. In this case it is used the *Soft Margin Classifier*. By tuning the parameter C, i.e. *cost for mis-classification*, we balance the trade off between two objectives of the SVM models: finding the maximum margin classifier and correctly classify most of the instances.

$$\begin{aligned} \min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ s.t. \\ y_i(\langle \mathbf{w}, x_i\rangle + b) \geq 1 - \xi_i, \forall i \\ \xi_i \geq 0, \forall i \end{aligned} \tag{4.11}$$

In case the data-set is impossible to separate linearly, we resort in the kernel trick which allow to efficiently compute non-linear separators by exploiting the characteristics of an higher order feature space, with no need to actually compute all the higher order features.
Practically we need a kernel function, Equation 4.12, which is essentially a symmetric function that enables us to compute dot products efficiently. In order to exploit the kernel trick some theoretical requirements must be met, in particular the *Mercer's Theorem*.

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle \tag{4.12}$$

Below are reported the parameters being used in the grid-search, Table 4.5, the experiments and the corresponding results in terms of different metrics, Table 4.6, as well as the figures corresponding to these experiments in terms of *confusion matrices* and *ROC curves*, Figure 4.14, 4.15, 4.16, 4.17.

Name	Values
C	0.1, 1, **10**
kernel	linear, poly, **rbf**
gamma	scale, **auto**

Table 4.5: Grid Search - SVC

Model	Accuracy Test	Precision	Recall	F1
Original	0.758	0.841	0.787	0.813
Original + PCA	0.742	0.842	0.768	0.803
Undersampling + PCA	0.735	0.728	0.737	0.733
Oversampling + PCA	0.805	0.778	0.819	0.798

Table 4.6: Results SVC

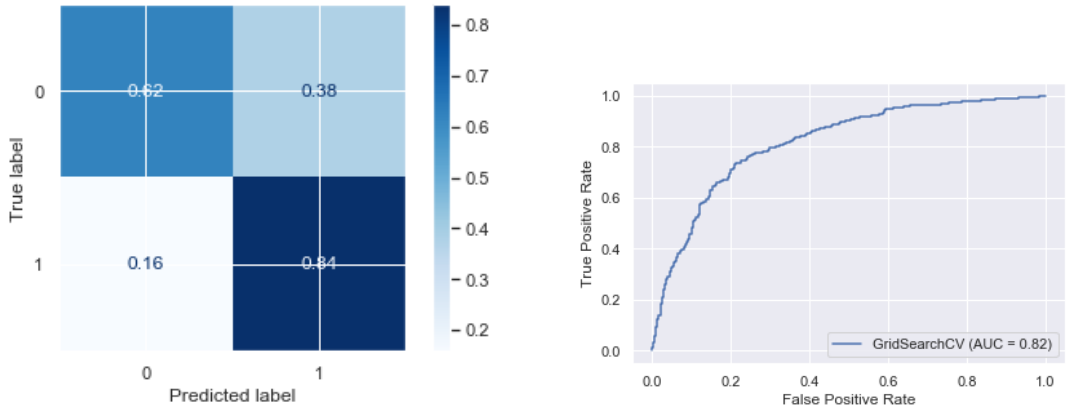

Figure 4.14: Original (no pca nor sampling) results - SVC

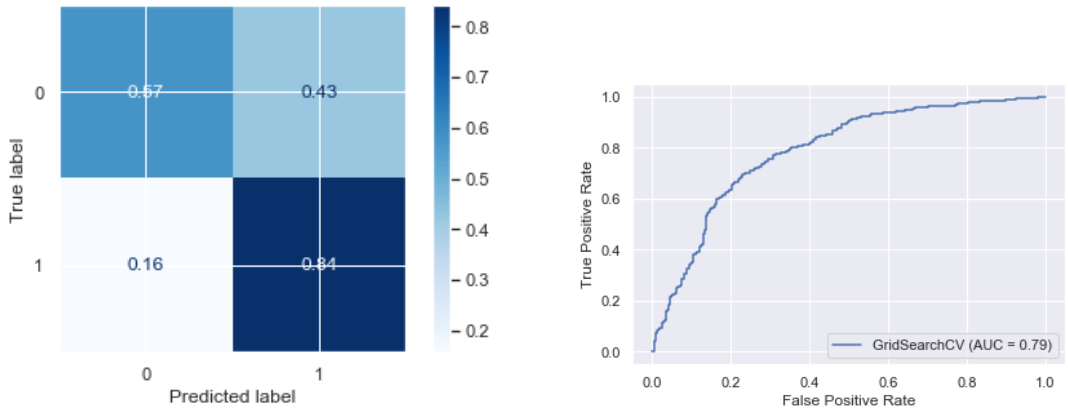

Figure 4.15: Original + PCA results - SVC

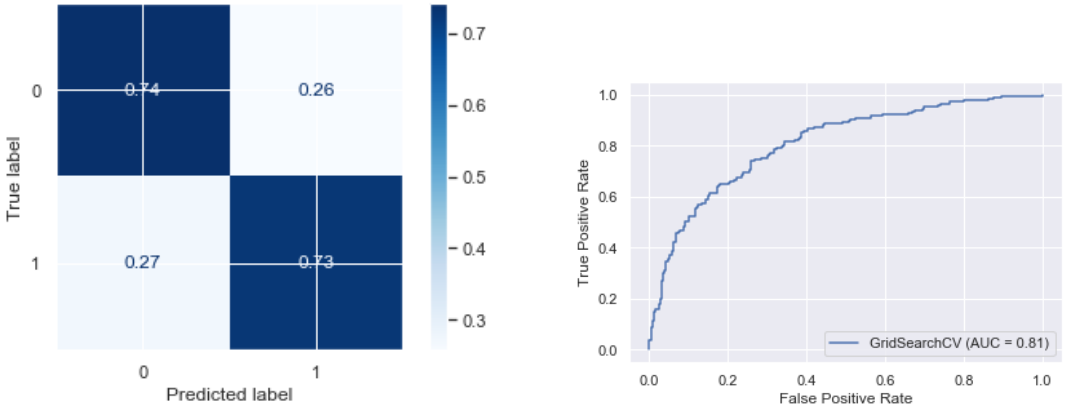

Figure 4.16: Under-sampling + PCA results - SVC

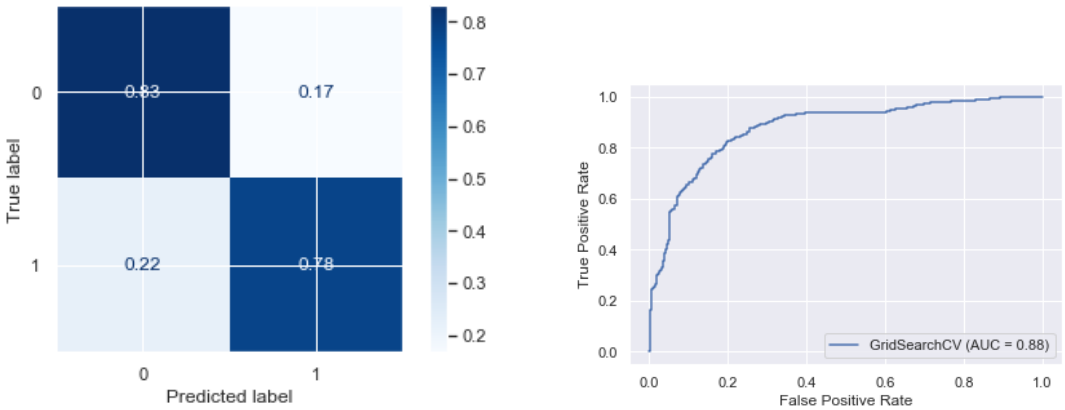

Figure 4.17: Over-sampling + PCA results - SVC

4.6 K-nearest neighbors

K nearest neighbor is a non-parametric classification model, i.e. no assumptions are made about the shape of the decision boundary, that classifies each instance by observing the class of the K nearest instances, then the class is assigned by observing which class is more frequent in the neighbors, i.e. *majority voting approach*. *KNN* is not very scalable since computing the distances between each instance is an onerous operation, as the number of instances increase.
In order to set up the KNN model correctly, the choice of K is critical: if it is too small, then the model is sensible to outliers whereas if it is too large then we might consider instances of other classes as neighbors.
So, K has been chosen by performing a grid search and evaluating the goodness of the model on the validation set.

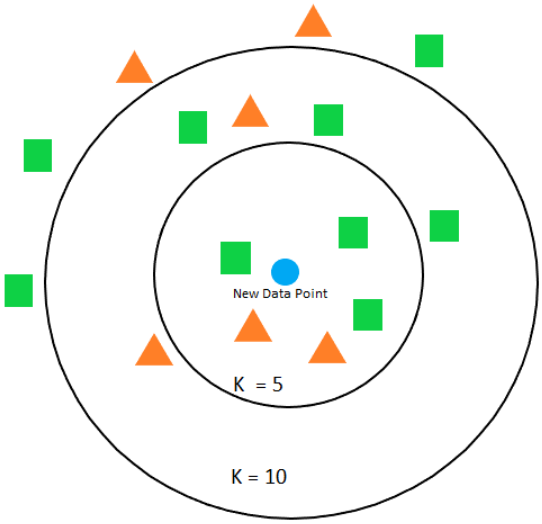

Figure 4.18: KNN - basic schema

Below are reported the parameters being used in the grid-search, Table 4.7, the experiments and the corresponding results in terms of different metrics, Table 4.8, as well as the figures corresponding to these experiments in terms of *confusion matrices* and *ROC curves*, Figure 4.19, 4.20, 4.21, 4.22.
Beside K another important parameter is which metric to use in order to actually compute the distances between samples.

Name	Values
K	**1**, 3, 5, 10
metric	**minkowski**, mahalanobis, euclidean

Table 4.7: Grid Search - KNN

Model	Accuracy Test	Precision	Recall	F1
Original	0.758	0.814	0.802	0.808
Original + PCA	0.751	0.801	0.801	0.801
Undersampling + PCA	0.716	0.669	0.736	0.701
Oversampling + PCA	0.961	0.959	0.962	0.960

Table 4.8: Results KNN

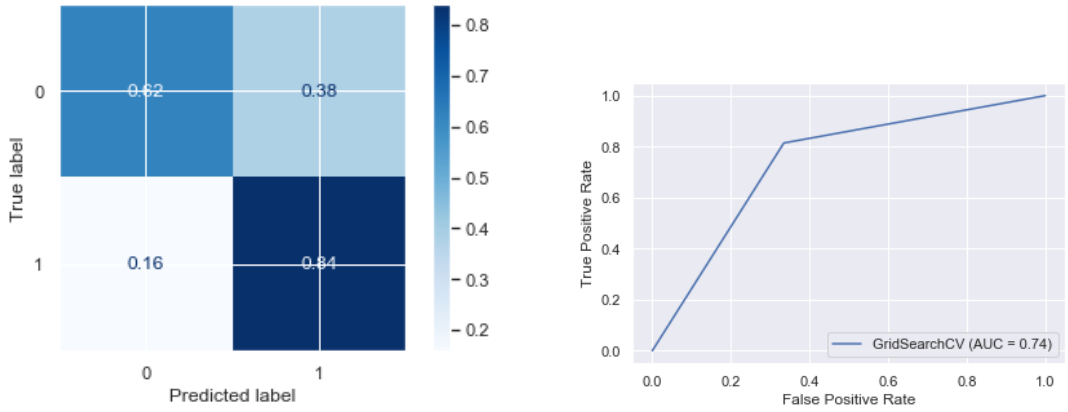

Figure 4.19: Original (no pca nor sampling) results - KNN

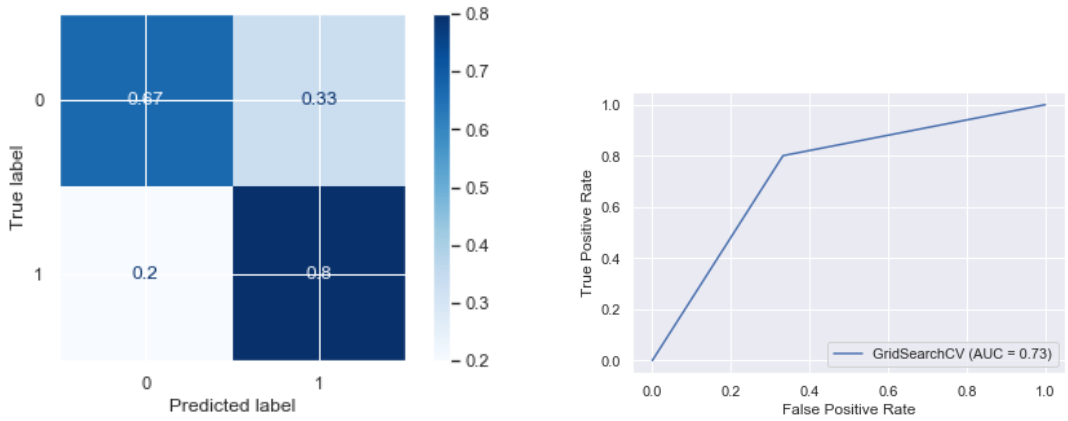

Figure 4.20: Original + PCA results - KNN

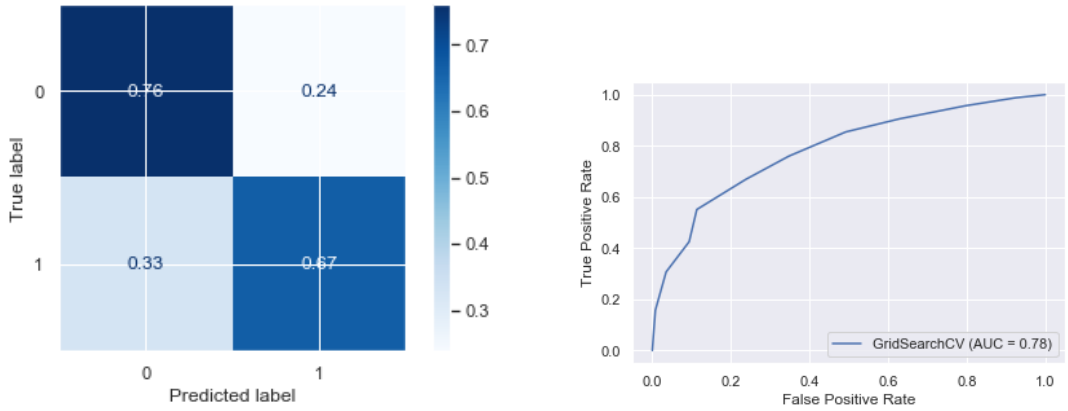

Figure 4.21: Under-sampling + PCA results - KNN

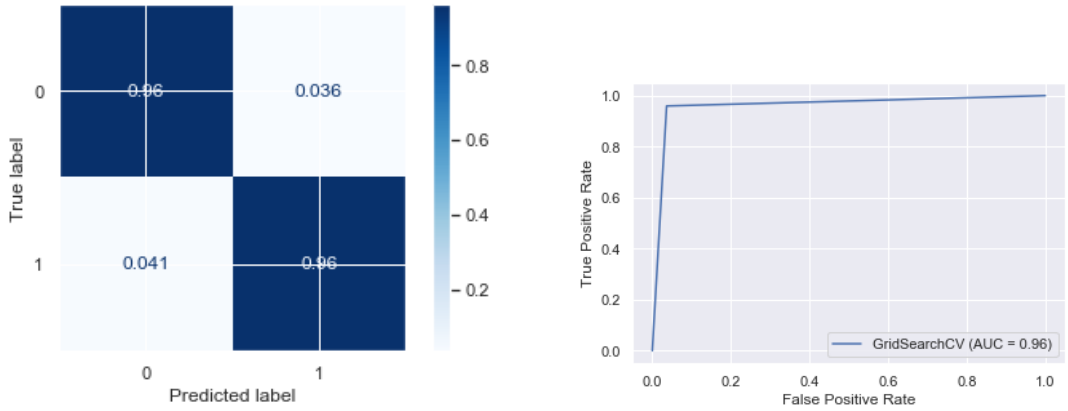

Figure 4.22: Over-sampling + PCA results - KNN

Chapter 5

Conclusions

The best results have been achieved by using the setting with *oversampling + PCA* and the *Random Forest* model.
The results are expressed by using different metrics such as accuracy, precision, recall, F1 and ROC curve, all evaluated on the test set.

Intuitively we could think that the effect of PCA was to increase the generalization capability of the models, keeping the training complexity low and lessening the features correlation issue, but as the previous experiments demonstrated some models achieved good performances also without PCA.
Regarding the balancing between scores and types (*red/white*), it has been proved that oversampling is the best choice, in fact under-sampling could, in this specific problem, cause the loss of valuable samples and end up in a weaker model (this might be because the number of samples is not great).
Interestingly, performing no sampling at all allowed to obtain discrete results.
Further studies might focus on more complex over sampling techniques able to build synthetic samples from the existing ones, e.g. SMOTE, instead of random sampling from the training set.

Binning the scores into categories was necessary due to the mis-representation of some values on the scale (very low and high scores), further studies might try out different binning strategies or try to acquire more data in order to perform a complete modelling of the data-set (through multi-class classification or regression).

Bibliography

[1] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. «Modeling wine preferences by data mining from physicochemical properties». In: *Decision Support Systems* 47.4 (2009). Smart Business Networks: Concepts and Empirical Evidence, pp. 547–553. ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2009.05.016. URL: http://www.sciencedirect.com/science/article/pii/S0167923609001377 (cit. on p. 1).