POLITECNICO
DI TORINO

Mathematics in Machine Learning Exam

Wine Quality Prediction

a. y. 2019/2020

Prof.
Francesco Vaccarino
Mauro Gasparini

Student
Ilio Di Pietro
matr. 266393

Contents

1 Introduction

Once viewed as a luxury good, nowadays wine is increasingly enjoyed by a wider range of consumers.
Portugal is a top ten wine exporting country, with 3.17% of the market share in 2005. Exports of its **vinho verde** wine have increased by 36% from 1997 to 2007. To support its growth, wine certification and quality assessment are key elements. Certification prevents the illegal adulteration of wines and assures quality for the wine market. Quality evaluation is often part of the certification process and can be used to improve wine making (by identifying the most influential factors) and to stratify wines such as premium brands.
Advances in information technologies have made it possible to collect, store and process massive, often highly complex datasets. All this data hold valuable information such as trends and patterns, which can be used to improve decision making and optimize chances of success. [1]

2 Wine Quality Prediction

2.1 Introduction to Wine Quality Prediction

This study will consider *vinho verde*, a unique product from the Minho (north-west) region of Portugal. In particular we will analyze the two most common variants, white and red. In order to give a meaning to a particular wine in therms of quality, a 10-point grading scale is used, where 0 means very bad and 10 is excellent.
Dealing with red and white wines, the output variable (quality) can be modeled using at least two supervised approaches:

1. Binary classification – *Good wine* if `quality` > 5, else *Bad wine*;
2. Regression – on the output variable `quality` (numeric output between 0 and 10).

The aim of this work will be to analyze our datasets with different classification and regression algorithms.

2.2 Dataset and environment setup

The analysis and the experiments are performed on UCI Machine Learning Wine Quality Data Set [2].

The dataset contains red and white variants of the "Vinho Verde" wine. The data were collected from May/2004 to February/2007.

All results are generated using Python programming language and *Colab* environment. Mainly, `sklearn`, `imblearn` and `numpy` libraries are used.

All the code can be found on GitHub at **link**.

3 Dataset Analysis

The Wine Quality dataset contains two datasets, one showing the qualities of red wines (with 1599 entries) and one regarding the ones in white wines (with 4898 entries).

The datasets show no missing values.

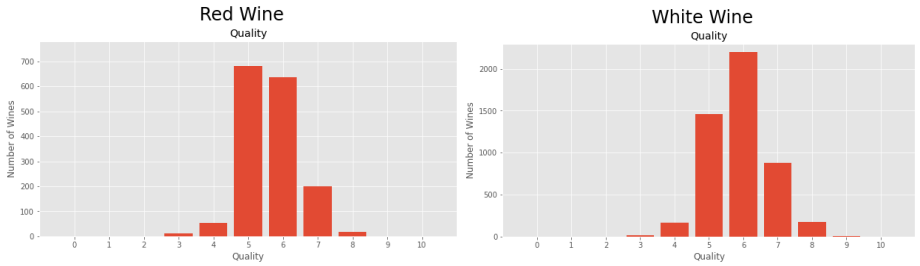
3.1 Features Description and Analysis

Each datum sample is characterized by the same 12 attributes, shown in Table 1. They give additional information about the wines, such as density, pH, alcohol and so on. All the attributes are described by integer and real numbers.

Attribute	Description
fixed acidity	amount of Fixed acids (tartaric, malic, citric, and succinic acids) per volume (numeric: from 3.8 to 15.9)
volatile acidity	amount of Volatile acids which must be distilled out from the wine before completing the production process (numeric: from 0.1 to 1.6)
citric acid	one of the fixed acids which gives a wine its freshness (numeric: from 0.0 to 1.7)
residual sugar	natural sugar from grapes which remains after the fermentation process stops, or is stopped (numeric: from 0.6 to 65.8)
chlorides	Chloride concentration in the wine (numeric: from 0.01 to 0.61)
free sulfur dioxide	part of the sulphur dioxide that when added to a wine is said to be free after the remaining part binds (numeric: from 1 to 289)
total sulfur dioxide	sum total of the bound and the free sulfur dioxide (numeric: from 6 to 440)
density	comparison of the weight of a specific volume of wine to an equivalent volume of water (numeric: from 0.987 to 1.039)
pH	numeric scale to specify the acidity or basicity the wine (numeric: from 2.7 to 4.0)
sulphates	mineral salts containing sulfur.They are connected to the fermentation process and affects the wine aroma and flavour (numeric: from 0.2 to 2.0)
alcohol	measured in % vol or alcohol by volume (numeric: from 8.0 to 14.9)
quality	final grades which represent quality scores (numeric: from 0 to 10, output target)

Table 1: Attributes description

The attribute that interests us most is `quality`, which represents the wines' final score and is our target output. Figures 1 show its distribution. The distributions are skewed: most of the values are between 3 and 9 while they are completely absent, wines totally bad or excellent.



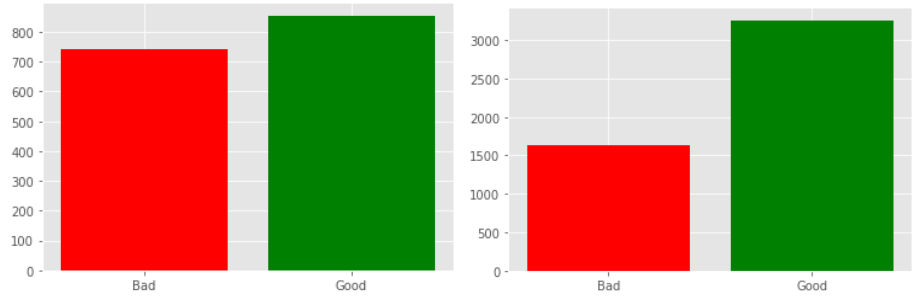
(a) *Red Wine dataset* (b) *White Wine dataset*

Figure 1: Distribution of "quality"

For our purpose, this attribute has been analyzed in two different ways to model it properly, both for classification and regression.

In order to perform binary classification, the variable is mapped from [0, 10] to [0,1], where 0 means *bad quality* and 1 *good quality*, according to the following criteria: if `quality` > 5, wine is good; otherwise, it is bad.

As it can be seen in Figure 2, despite the Red Wine dataset is balanced, White Wine dataset is heavily skewed towards good wines and this could make the employed algorithms skewed as well. Imbalanced datasets pose a challenge for predictive modelling as most of the machine learning algorithms used for classification are designed around the assumption of an equal number of examples for each class. That results in models having poor predictive performance, specifically for the minority class. This issue will be further discussed in Chapter 5.



(a) *Red Wine* (b) *White Wine*

Figure 2: Binary distribution of "quality"

For the regression task, since we didn't group all the quality values lower than a threshold, to understand if the data representation was fairly distributed, we displayed the box plots of the two datasets.

From Figure 3, we can see that the distributions are very similar: both red and white wines quality lies between 4 and 7. Furthermore, we can see that some points are pretty far from their distribution and that could induce some error in our models: in this case, we consider these samples as outliers and for this reason, before approaching with regression tasks, we will remove these instances from both datasets.

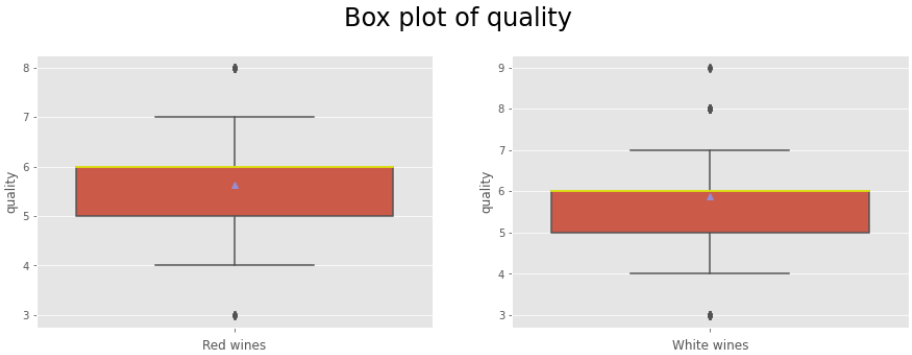


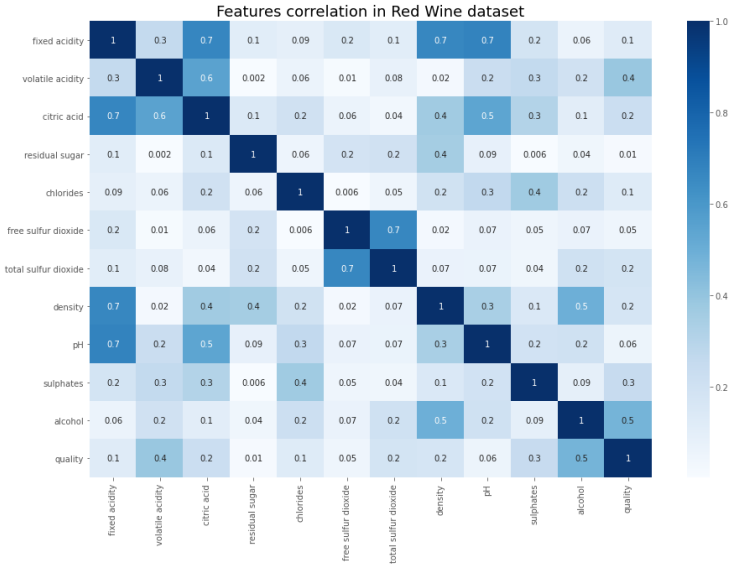
Figure 3: Boxplots of "quality"

In order to gain a better understanding of the features and to investigate the dependence between them, the correlation matrices are computed: each cell in the table shows the pairwise Pearson Correlation Coefficient r, calculated with the formula (3.1)

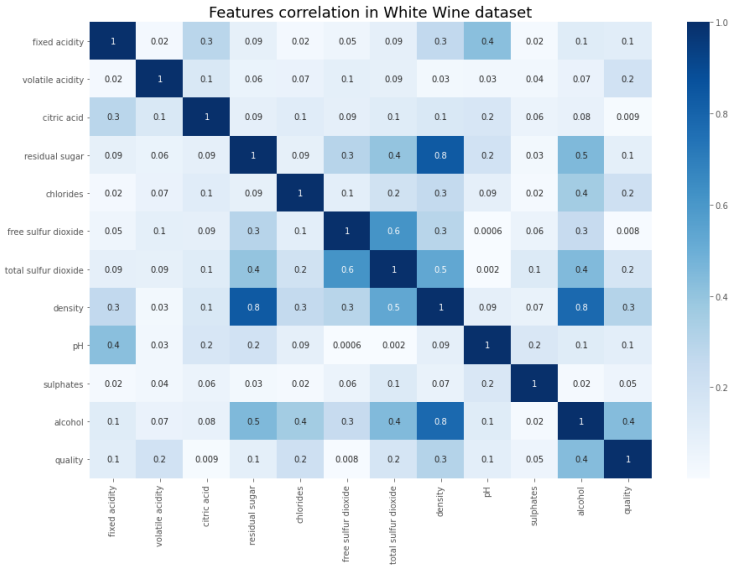
$$r_{ij} = \frac{cov(X_i, X_j)}{\sigma_{X\,i}\sigma_{X\,j}} \tag{3.1}$$

where cov is the covariance and σ is the standard deviation of the features X_i and X_j. The correlation coefficient ranges from -1 to 1. A value of 1 implies that a linear equation describes the relationship between X_i and X_j perfectly, with all data points lying on a line for which X_j increases as X_i increases. A value of -1 implies that all data points lie on a line for which X_j decreases as X_i increases. A value of 0 implies that there is no linear correlation between the variables. Therefore, the higher the absolute value of the coefficient, the more correlated the two variables are.

Figure 4 shows that there are some low correlated features to the target variable `quality`, in both datasets. In particular, the attributes with a correlation belonging to the interval $[-0.07, 0.07]$ are dropped, so as to keep working only on relevant information.
As for the Red Wine data, the attributes `residual sugar`, `free sulfur dioxide`, `pH` are discarded, which leaves us with 9 remaining characterizing features;
as for the White Wine dataset, the attributes `citric acid`, `free sulfur dioxide`, `sulphates` are loosely correlated to `quality` and therefore 9 attributes are kept.



(a) *Red Wine*



(b) *White Wine*

Figure 4: Correlation Matrices

4 Data Preprocessing

4.1 Bootstrap for Feature Scaling

The ***bootstrap method*** is a statistical technique for estimating statistics about a population by averaging estimates from multiple small data samples [3].

In particular, bootstrap addresses the issue of collecting information about a population, when we are provided only with a sample of it. So how can we be sure that sample is representative of the whole population? Namely, does the mean of our sample well approximate the true mean of the population?

Therefore, the bootstrap approach proposes the idea of estimating our statistics many times, by resampling *with replacement* our original sample, instead of doing it only once on the obtained sample realization. It allows us to mimic the process of obtaining new data sets, so as to estimate the variability of our estimate, without generating additional samples. Each of those "bootstrap data sets" is the same size of the original data set and inside each of them some observations may appear more than once, whilst some not at all.

Specifically, bootstrap allows us to obtain **Standard Errors** (SEs) of estimators, without assuming any parametric form of the population in the presence of i.i.d. (independent and identically distributed) sampling.

Given the parameter to estimate θ, its estimator $\hat{\theta}$, the population P_x, $x_1, ..., x_n$ i.i.d. P_x, if we can simulate from P_x, then we can estimate the SE of any $\hat{\theta}$ with the following algorithm:

1. Simulate $x_1^*, ..., x_n^*$ i.i.d. P_x
2. Calculate $\hat{\theta}^*$ on $x_1^*, ..., x_n^*$
3. Iterate over 1. and 2. for a large number of times B
4. Compute $\hat{SE}^*(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{i=1}^{B}(\hat{\theta}_i^* - \bar{\theta^*})}$.

By the Law of Large Numbers, for large values of B, $\bar{\theta^*} \to \theta$ and $\hat{SE}^*(\hat{\theta}) \to SE(\theta)$.

As in our case, most of the time P_x is unknown, so the bootstrap method suggests to use $\hat{P}_x$ instead, which is the empirical distribution of $x_1, ..., x_n$. As a consequence, the bootstrap estimate $\hat{SE}^*(\hat{\theta})$ is based on $\hat{P}_x$ and the simulation of $\hat{P}_x$ is easily accomplished by resampling $x_1, ..., x_n$ with replacement.

In our case, the bootstrap approach was used to obtain estimates of mean and standard deviation values closer to the real ones and their related SEs.

The estimates of mean and standard deviation are used to **standardize** the data. Standardization is a scaling technique for centering the values around their mean with a unit standard deviation, following the formula 4.1:

$$X' = \frac{X - \mu}{\sigma} \tag{4.1}$$

where in our case X is the train or test set, μ and σ are, respectively, the estimated mean and standard deviation of the features values. Standardizing the features is important when we compare measurements that have different units. Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bias. Moreover, the resulting scaled

data has a Gaussian distribution and meets the requirements of those algorithms that assume the data has such distribution, such as Logistic Regression.

4.2 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is an unsupervised method that aims to find the smallest subspace such that as much information about the original data as possible is preserved.

Given a fixed number of points, PCA has as its goal to find an orthogonal set of linear basis vectors such that the average reconstruction error is minimized. In the found optimal low-dimensional encoding of the data, the mean squared distance between the data points and their projection is minimized and the variance of the projected data maximized (the higher the value of the variance, the more information is stored).

Given data described by D variables, we aim to find a reduced subspace of dimensions $d < D$, such that the most variability of the data is kept. The d variables describing the new space are called Principal Components (PCs) and each one of them is orthogonal to the previous one and points in the direction of the largest variance. We will denote them as $u_1, ..., u_d$.

Specifically, let PC_j be a linear combination of $x_1, ..., x_D$, defined by the weights

$$\mathbf{w}^{(j)} = (w_1^{(j)} ... w_D^{(j)})^T$$

i.e. $u_j = {\mathbf{w}^{(j)}}^T \mathbf{x}$.

The first PC u_1 is chosen to have maximum variance:

$$Var(u_1) = Var({\mathbf{w}^{(1)}}^T \mathbf{x}) = {\mathbf{w}^{(1)}}^T S \mathbf{w}^{(1)} \tag{4.2}$$

where S is the sample covariance. $Var(u_1)$ is maximized if the eigenvalue λ_1 is the maximum eigenvalue of S and u_1 is its corresponding eigenvector.

All PCs are eigenvectors of S and their eigenvalues satisfy the condition

$$\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_D$$

so that

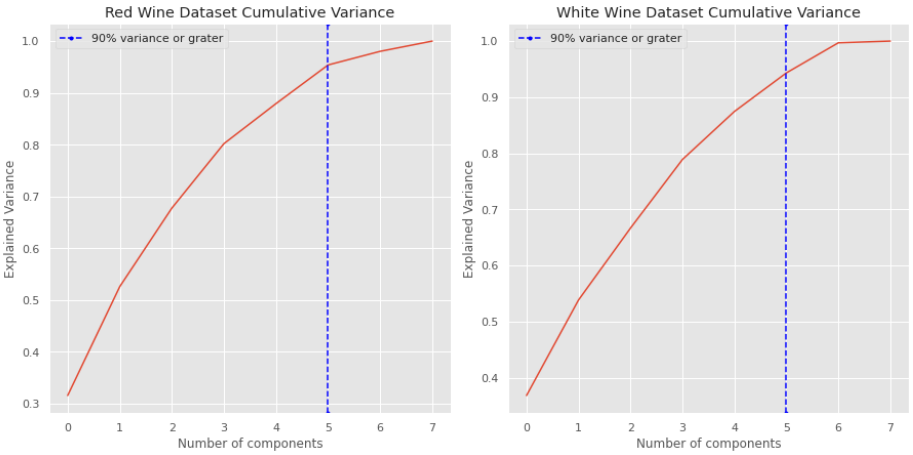
$$Var(u_1) \geq Var(u_2) \geq ... \geq Var(u_D)$$

The total variance of the data is consequently decomposed as:

$$\sum_{i=1}^{D} Var(u_i) = \sum_{i=1}^{D} \lambda_i = \sum_{i=1}^{D} Var(x_i)$$

Those conditions allow us to better understand why it is highly recommended to standardize the data before applying PCA: the principal directions are the ones along which the data shows maximal variance; this means PCA can be misled by directions in which the variance is high merely because of a measurement scale [4].

Here, PCA was applied to obtain the PCs retaining 90% or more of the cumulative variance. As Figure 5 shows, as for both datasets, 5 PCs were selected out of 9.



(a) *Red Wine* (b) *White Wine*

Figure 5: Cumulative Variance of Principal Components

5 Classification

5.1 Brief introduction to Classification

In machine learning and statistics, classification is a supervised learning problem with the goal of identifying to which set of categories (target label) a new observation belongs, on the basis of a training set of data whose labels are known.

More in details, we want to learn a mapping from inputs x to outputs y, where $y \in \{1, ..., C\}$, with C being the number of classes. One way to formalize the problem is as **function approximation**. We assume $y = f(x)$ for some unknown function f, and the goal of learning is to estimate the function f given a labeled training set, and then to make predictions using $\hat{y} = \hat{f(x)}$.[4]

In the following section, the classification task will be performed. Given the train data, our goal is to predict the quality of the wines which belong to the test set.
Before that it will be necessary to prepare our data for a binary classification.

5.2 Oversampling with ADASYN

As explained above (see section 3.1), in order to perform a binary classification, our target variable was mapped from $[0, 10]$ to $[0, 1]$ but as result, the white wines dataset result imbalanced.
When dealing with imbalanced data sets, two main solutions can be applied:

1. Usage of resampling techniques;
2. Change in the evaluation metrics (see section 5.3).

As for the sampling techniques, they are used to create balanced datasets before fitting a classifier on it. They can be divided into three main classes:

- **undersampling**, consisting in sampling from the majority class to keep only a part of those points;
- **oversampling**, consisting in replicating some points from the minority class in order to increase its cardinality;
- **generating synthetic data**, that is to say, creating new synthetic points from the minority class to increase its cardinality.

It is important to remark that resampling techniques should only be applied on the training set, so as not to distort testing outcomes.

The main disadvantage in using standard oversampling is the major probability of overfitting, due to the fact that the model sees the same data points more than once at training time. A possible solution to that problem is to create synthetic samples starting from the existing ones, instead of simply duplicating them: this method can be seen as a form of data augmentation for the minority class and the one that will be used is the Adaptive Synthetic Sampling Method, or ADASYN.

ADASYN is based on the idea of adaptively generating minority data samples according to their distributions: more synthetic data is generated for minority class samples that are harder to learn compared to those minority samples that are easier to learn. The ADASYN method can not only reduce the learning bias introduced by the original imbalance data distribution, but can also adaptively shift the decision boundary to focus on those difficult to learn samples.[5]

The algorithm is the following:

Define m_s and m_l as the number of minority class examples and the number of majority class examples, respectively. Therefore, $m_s \leq m_l$ and $m_s + m_l = m$.

1. Calculate the degree of class imbalance:

$$d = \frac{m_s}{m_l}$$

   where $d \in (0, 1]$

2. If $d < d_{th}$ then (d_{th} is a preset threshold for the maximum tolerated degree of class imbalance ratio):

   (a) Calculate the number of synthetic data examples that need to be generated for the minority class:

$$G = (m_l - m_s) \times \beta$$

   Where $\beta \in [0, 1]$ is a parameter used to specify the desired balance level after generation of the synthetic data. $\beta = 1$ means a fully balanced data set is created after the generalization process.

   (b) For each example $x_i \in minorityclass$, find K nearest neighbors based on the Euclidean distance in n dimensional space, and calculate the ratio r defined as:

$$r_i = \frac{\Delta_i}{K} \quad i = 1, ..., m_s$$

   where Δ_i is the number of examples in the K nearest neighbors of x_i that belong to the majority class, therefore $r_i \in [0, 1]$;

   (c) Normalize r_i according to $\hat{r_i} = r_i / \sum_{i=1}^{m_s} r_i$, so that $\hat{r_i}$ is a density distribution ($\sum_i \hat{r_i} = 1$);

   (d) Calculate the number of synthetic data examples that need to be generated for each minority example x_i:

$$g_i = \hat{r_i} \times G$$

   (e) For each minority class data example x_i, generate g_i synthetic data examples according to the following steps:
   Do the **Loop** from 1 to g_i:

      i. Randomly choose one minority data example, x_{zi}, from the K nearest neighbors for data x_i;

      ii. Generate the synthetic data example:

$$s_i = x_i + (x_{zi} - x_i) \times \lambda$$

      where $(x_{zi} - x_i)$ is the difference vector in n dimensional spaces, and λ is a random number: $\lambda \in [0, 1]$.
      End **Loop**.

The key idea of ADASYN algorithm is to use a density distribution $\hat{r}_i$ as a criterion to automatically decide the number of synthetic samples that need to be generated for each minority data example. Physically, $\hat{r}_i$ is a measurement of the distribution of weights for different minority class examples according to their level of difficulty in learning. [5]

Starting from the imbalanced distribution presented in White Wine (see Figure 2), the data set reach the following more balanced configuration: from 1312 to 2417 *False* labels.

Since for the White Wine dataset we now have two different train sets available (one resulting from the PCA transformation and one reduced and then oversampled with ADASYN), the classification tasks will be performed on both datasets and the results will be compared, in order to find the best one.

5.3 Metrics

The metrics with which machine learning models are generally evaluated is **accuracy**, which is mathematically defined as:

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

where TP stands for *True Positive* (True labels predicted as True), TN are the *True Negatives* (False labels predicted as False), FP are the *False Positives* (False labels predicted as True) and FN means *False Negative* (True labels predicted as False). So that implies accuracy represents the fraction of predictions got right by the model.

When using imbalanced datasets, accuracy turns out not to always be a good evaluation metrics: 66.52% of the *White Wine* dataset consist of wine which are good, so even if all wines were classified as good and, therefore, all bad wines were misclassified, we would still have a total accuracy of 67%. In order to understand if the model is classifying correctly even the minority class, the solution is to look at different evaluation metrics. In particular, in this notebook, we make use of the following ones:

- **confusion matrix**, a representation of the results divided into TP, TN, FP, FN
- **recall** = $\frac{TP}{TP+FN}$, so the percentage of good wines that were correctly identified by our model
- **precision** = $\frac{TP}{TP+FP}$, which tells us how many predicted samples are relevant i.e. our mistakes into classifying sample as a correct one if it's not true.
- **F1-score** = $2 \cdot \frac{precision \cdot recall}{precision + recall}$, harmonic mean of precision and recall, which is ideal in the case of imbalanced data.

5.4 K-fold Cross Validation and Hyperparameters Tuning

K-fold cross-validation is a common approach to estimate test errors. The idea is to randomly divide the data into K equal sized parts. For $k = 1, 2, ..., K$, the k_{th} subset is left out, while $K - 1$ ones are used to fit the model. Then,

predictions are obtained for the left out k_{th} part. Finally, results computed on all values of k are combined.

Specifically, let the K parts be $C_1, C_2, ..., C_K$, given n multiple of K, $n_k = \frac{n}{K}$ the observations contained in part k,

$$CV_{(K)} = \sum_{k=1}^{K} \frac{n_k}{n} MSE_k \tag{5.1}$$

where MSE is the Mean Squared Error and is calculated as

$$MSE_k = \sum_{i \in C_k} \frac{(y_i - \hat{y}_i)^2}{n_k} \tag{5.2}$$

and y_i is the real response value and $\hat{y}_i$ is the fit for the observation i, computed on data when the k_{th} part is removed.

If $K = n$, the method is called *leave-one-out cross-validation* (LOOCV).

In order to find the models parameters performing best with our data, a **Grid Search** is proposed: it performs an exhaustive search over specified parameter values for an estimator and returns the ones achieving the best results at training time according to a certain metrics, which is accuracy in our case. The parameters of the estimator are optimized by a 5-folds cross-validated grid-search over a parameter grid, so the total number of fits performed for each classifier on each set is

$$tot\_fits = \prod_{i=1}^{n\_parameters} n\_candidates_i \cdot 5$$

where $n\_parameters_i$ is the number of parameters to be optimized for the current classifier and $n\_candidates_i$ is equal to the possible candidates for that parameter.

5.4.1 Logistic Regression

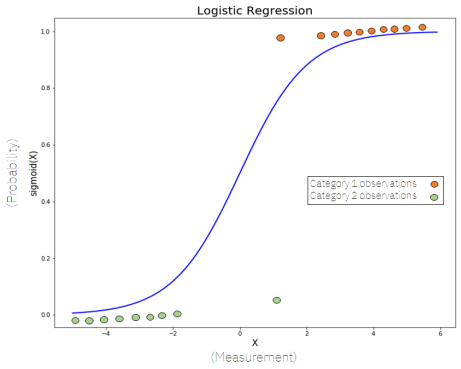


Figure 6: General Logistic Regression [6]

Logistic Regression (LR) is a binary classification model used to model the relationship between a binary target features y and independent features x,

corresponding to the following definition [4]:

$$p(y = 1|\mathbf{x}, \mathbf{w}) = Ber(y|sigm(\mathbf{w}^T\mathbf{x})) \tag{5.3}$$

where y is the vector of training labels, $\mathbf{w}$[1] is the vector of regression weights and defines the normal to the decision boundary, $\mathbf{x}$ is the training data, Ber is the Bernoulli distribution (5.4) and $sigm(x)$ is the sigmoid function (5.5).

$$Ber(x|\mu) = \mu^x(1-\mu)^{1-x} = (1-\mu)\, e^{(x\, log(\frac{\mu}{1-\mu}))},\ x \in \{0, 1\} \tag{5.4}$$

$$sigm(x) := \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{5.5}$$

Let's denote $p(y|\mathbf{x}, \mathbf{w})$ as $p(y)$. The formula 5.3 becomes

$$p(y) = \frac{1}{1+e^{-(w_0+w_1x)}} \tag{5.6}$$

The logistic model is used to model the probability of a certain class or event existing. Using the Sigmoid function, each object being detected is assigned a probability between 0 and 1, with a sum of one.

Our goal is to find w_0 and w_1, by maximizing the likelihood:

$$l(w_0, w_1) = \prod_{i;y_i=1} p(y_i) \prod_{i;y_i=0} (1 - p(y_i)) \tag{5.7}$$

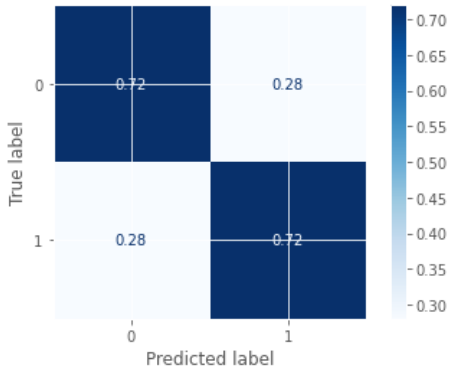
Logistic Regression is one of the most popular approaches to classification for several reasons, including the following ones: LR models are easy to fit to data (simple and fast algorithms), are easy to interpret and to extend to multi-class classification.

The tuned parameter is $C = \frac{1}{\lambda}$, the inverse of *regularization strength*. λ aims to find a trade-off between model simplicity (high λ value), which might lead to underfitting, and too high a complexity (low λ value), which might result in overfitting the data: as a consequence, lowering C implies strengthening the lambda regulator.
The values of C used to perform an exhaustive search are:

- **C**: 0.001, 0.01, 0.1, 1, 10, 100, 1000 .

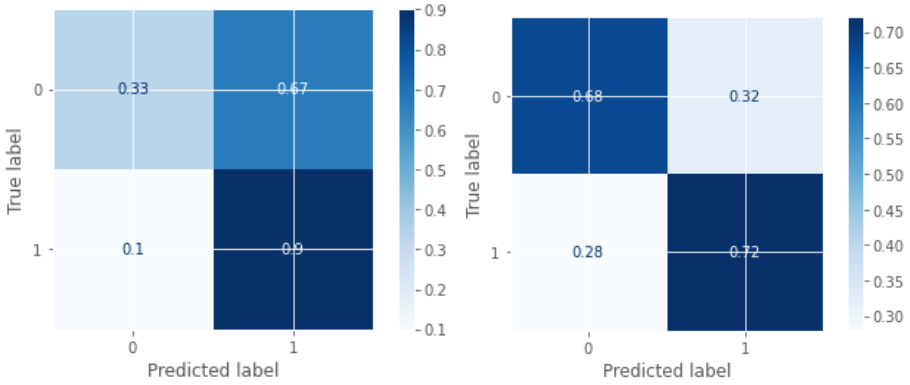
Figure 7 and 8 show the confusion matrices obtained with the Logistic Regression algorithm on all datasets. The resulting accuracies and F1-scores can be found in Table 2 and in Table 3. Mostly, the model succeeds in predicting both True Positives and True Negatives. In the White Wine dataset, the application of the oversampling improves the performances: the classifier is now able to predict in a better way points belonging to the minority class.

[1] in statistics we refer to $\mathbf{w}$ as $\boldsymbol{\beta}$, which is the real goal of the analysis: the purpose is to give an interpretation of the effects of the different predictors on the response. (see Section 6 for more information about General Linear Models)



(a) *C = 0.01*

Figure 7: Confusion Matrices for Logistic Regression on *Red Wine* Dataset



(a) *Non-oversampled dataset* $C = 1$

(b) *ADASYN oversampling* $C = 0.1$

Figure 8: Confusion Matrices for Logistic Regression on *White Wine* Dataset

5.4.2 Support Vector Machines (SVMs)

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The problem SVMs tries to solve is to find the optimum hyperplane that divides the data and maximizes the distance between points belonging to the positive class (1) and the ones belonging to the negative class (-1). The first assumption is the classification problem we want to solve is linearly separable, which means it must exist an hyperplane that divides all data belonging to the two classes.

We define a real-valued function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ as:

$$f(x) = \langle \mathbf{wx} \rangle + b \tag{5.8}$$

where $(\mathbf{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ are the parameters that control the function and the decision rule is given by $\text{sgn}(f(\mathbf{x}))$.

We introduce two parallel hyperplanes that divide the samples, described by the following formulations:

$$f(x) = \langle \mathbf{wx} \rangle + b = 1 \tag{5.9}$$
$$f(x) = \langle \mathbf{wx} \rangle + b = -1 \tag{5.10}$$

Points belonging to positive class satisfy the relation $\langle \mathbf{wx} \rangle + b >= 1$, while the ones belonging to negative class satisfy $\langle \mathbf{wx} \rangle + b <= -1$.

We define the **margin** as the half distance between those two hyperplanes (dashed lines in Figure 9):

$$margin = \frac{1}{\|\mathbf{w}\|} \tag{5.11}$$

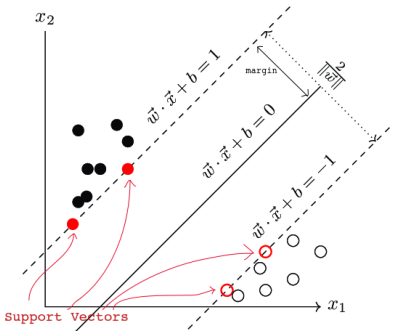


Figure 9: Example of margin in SVM [7]

The points lying on the margin identify the distance we want to calculate: we need to find w and b such that the margin is maximized and at the same time all points are correctly classified. We are interested in the largest margin because it give us maximum robustness relative to uncertainty and independence from correctly classified instances.
Mathematically speaking, that formulation is equivalent to:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s. t.} \quad \forall i, y_i(\langle \mathbf{w}, x_i \rangle + b) > 1 \tag{5.12}$$

where y_i is the true label and the prediction is the evaluated distance of the sample from the hyperplane. It means that if the label and the prediction have the same sign, the prediction is correct.

Once the algorithm converged, the model can be described using only the points on the margin, called **support vectors**. That is the reason why SVM can easily scale.

As we said before, being able to solve this problem means that our problem is linearly separable (**hard margin problem**). In real world, unfortunately, the hypothesis of linear separability may not hold and the algorithm may not converge: we need to relax this constraint, introducing the **soft margin problem**.

Soft Margin Problem We add to the previous formulation (5.12) a new therm:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \quad \text{s. t.} \quad \forall i, y_i(\langle \mathbf{w}, x_i\rangle + b) > 1 - \xi_i \quad \text{and} \quad \xi_i >= 0 \tag{5.13}$$

where ξ_i, called **slack variable**, is the distance of x_i from the corresponding class margin, if x_i is on the wrong side of the margin, and 0 otherwise. C is a constant used to tune the level of misclassification that we are willing to accept.

Kernel Trick In some cases, problems are not well separable on the original feature spaces and a non-linear decision boundary is needed. A possible solution is to map our features in a higher dimensional space, usually a Hilbert space, in which, hopefully, a linear separation becomes possible.

A Hilbert space H is a real or complex inner product space that is also a complete metric space with respect to the distance function induced by the inner product [8].

In the new space, the training samples become $((\phi(x_1), y_1), ..., (\phi(x_n), y_n))$, where $\phi : \chi \to H$ is a feature map, with $\chi \subset \mathbb{R}^d$ being the input space and H the new inner product space. Unfortunately, following this approach, the inner product is very hard to compute. To reduce the computational effort, we can use a symmetric function $K : \chi \times \chi \to \mathbb{R}$, implementing the inner product in the features space, which has lower computational cost. K is called *kernel function* and is such that:

$$K(x_i, y_i) = (\phi(x_i), \phi(y_i)) \tag{5.14}$$

In order to apply this mapping, the kernel function must satisfy **Mercer's theorem**, stating that: a symmetric function $K : \chi \times \chi \to \mathbb{R}$ implements an inner product in some Hilbert space, if and only if, for all $x_1, x_2...x_m$, the Gram matrix $G_{ij} = K(x_i, x_j)$ is positive semidefinite. A matrix is semidefinite if and only if $x^T G x \geq 0 \quad \forall(x) \in \mathbb{R}^n$.

Here, two different kernel functions were used: **Linear** kernel and **Radial Basis Function**.

Linear SVM Given two data points x and x', the linear kernel function k is defined as:

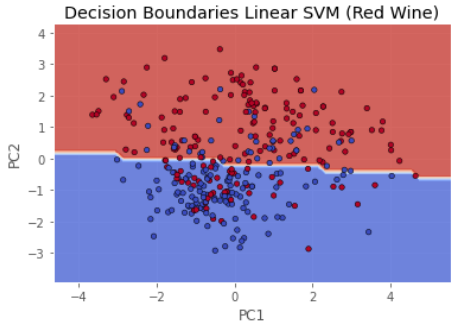
$$k(x, x') =< x, x' > \tag{5.15}$$

Given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new samples.

The parameter C was tuned to regularize the amount of misclassified training points allowed in the model. For large values of C, a smaller-margin hyperplane will be chosen, if that hyperplane gets all the training points classified correctly. Too high values of C might cause overfitting. Conversely, for a very small value of C, the optimizer will look for a larger-margin separating hyperplane, therefore for a simpler decision function, even if that choice brings to heavy misclassification. In this case, the chosen range is:

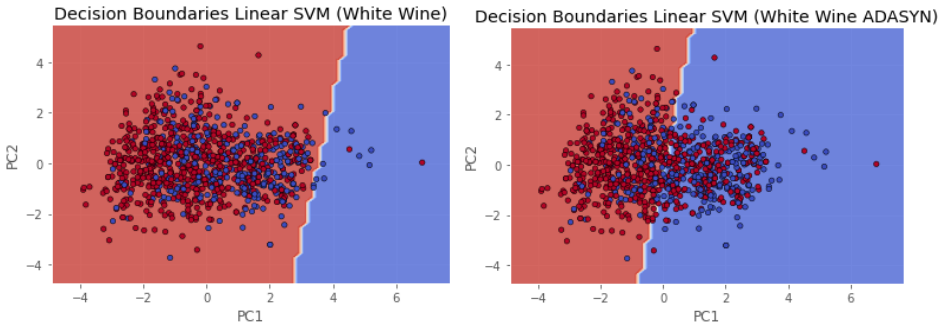
- $C = [10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3]$.

Those values are used to train a linear SVM on the training set. The obtained decision boundaries are shown in Figure 10 and 11. In the first case, the model is capable of correctly classifying most points. Whit White Wine dataset our model is able to classify very well the true labels (red dots in Figure), while it demonstrates some uncertainty with the negative class (blue dots), due to the imbalance of the dataset. When ADASYN is applied, performances improves.



(a) $C = 0.01$

Figure 10: Decision Boundaries on test set (*Red Wine* Dataset)



(a) *Non-oversampled dataset* $C = 0.1$

(b) *ADASYN oversampling* $C = 10$

Figure 11: Decision Boundaries on test set (*White Wine* Dataset)

RBF Kernel Given two data points x and x', the Radial Basis Function (RBF) kernel SVM k is defined as:

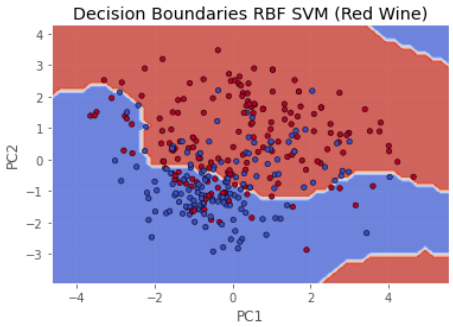
$$k(x, x') = e^{(-\gamma \|x - x'\|^2)} \tag{5.16}$$

where γ must be greater than 0 and represents the inverse of the radius of the area of influence of samples selected by the model as support vectors, and $\|x - x'\|^2$ is the squared Euclidean distance between two data points x and x'.

In this case, an exhaustive search (using the function `GridSearchCV` from `sklearn.model_selection`) over the following specified parameters was performed:

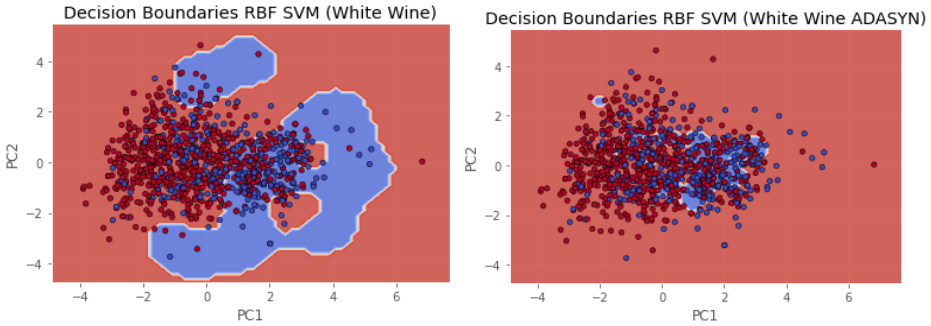
- $C = [10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3]$;
- $\gamma = [10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}, 0.5, 1, 10, 10^2]$.

Figure 12 and 13 shows the decision boundaries on the test For Red Wine dataset our model performs more or less in the same way we described for the linear kernel both in terms of accuracy and F1-score. Regarding the White Wine ones, it would seems that the model is able to classify in a better way the blue dots with respect to linear model, because it seems that the margin are more defined. In practice, however, there is no improvement, and also when oversampling is applied, performances doesn't improves.



(a) $C = 1, \gamma = 0.5$

Figure 12: Decision Boundaries on test set (*Red Wine* Dataset)



(a) *Non-oversampled dataset* $C = 1, \gamma = 1$

(b) *SMOTE oversampling* $C = 10, \gamma = 10$

Figure 13: Decision Boundaries on test set (*White Wine* Dataset)

5.5 Classification Results

Table 2 and 3 show the achieved results by all classifiers, respectively in terms of accuracy and F1-score on the `False` label. In the former case, one of the model that seems to have worked worse is SVM when linear kernel is applied. It is interesting to note however that the application of ADASYN oversampling

for White Wine dataset seems to have brought significant improvements, indeed the accuracy score remained more or less unchanged (from 0.68 to 0.69). However, this is not coherent with what we see in the table 3: the F1-score has increased significantly, from 0.20 to 0.61. This can be seen as an example of how the accuracy is not always the right metric to evaluate the model when dealing with imbalanced datasets. F1-score, in this case, shows a more realistic view.

An overall view on the application of the ADASYN technique shows improvements in the majority of the cases.

	Red Wine	*White Wine*	*White Wine ADASYN*
Logistic Regression	0.72	0.71	0.71
Linear SVM	0.71	0.68	0.69
RBF SVM	0.71	0.72	0.77

Table 2: Classification Results in terms of accuracy

	Red Wine	*White Wine*	*White Wine ADASYN*
Logistic Regression	0.70	0.43	0.61
Linear SVM	0.70	0.20	0.61
RBF SVM	0.69	0.52	0.58

Table 3: Classification Results in terms of F1-score on the *False* label

6 Regression

6.1 Introduction to Regression

Regression refers to a broad class of supervised learning techniques where the aim is to predict a set of quantitative responses $y_1, ..., y_n$ via a function of set of predictors, collected in a matrix $\mathbf{X}$, consisting of p features, each of which can be continuous or discrete.

Mathematically:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{6.1}$$

where:

- $\mathbf{Y} = [Y_1, ..., Y_n]^T$ is the response vector (a random vector which becomes a numerical vector $y_1, ..., y_n$ when observed);
- $\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$ is a constant matrix[2] containing the values of p predictors;
- $\boldsymbol{\beta} = [\beta_0, ..., \beta_{p-1}]^T$ is a set of parameters usually unknown and the main object of a statistical analysis. In Machine Learning, there are a set of weights assigned to the predictors;
- $\boldsymbol{\varepsilon} = [\varepsilon_1, ..., \varepsilon_n]^T$ is a set of unobservable random variables, called errors, assumed to have $E(\boldsymbol{\varepsilon}) = 0$ and $Var(\boldsymbol{\varepsilon}) = I\sigma^2$ (situation of i.i.d. noise added to our signal $\mathbf{X}\boldsymbol{\beta}$).

6.1.1 Least Squares Estimate

As said before, we want to make an estimate of the parameter vector $\boldsymbol{\beta}$. For this purpose, we will assume that in (6.1), $\boldsymbol{\varepsilon}$ represents a Gaussian noise, so:

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 \mathbb{1})$$

This implies:

$$\mathbf{Y} = (\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}) \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbb{1})$$

since $\mathbf{X}\boldsymbol{\beta}$ is a constant. Thus, we can compute the likelihood of $\mathbf{Y}$ (which is independent normal) as:

$$\ell(\mathbf{Y}) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\{-\frac{1}{2\sigma^2}(y_i - \mu_i)^2\}$$

where μ_i is the mean of Y_i given by the i_{th} row of $\mathbf{X}\boldsymbol{\beta}$.

$$\ell(\mathbf{Y}) = (2\pi\sigma^2)^{-\frac{n}{2}} \exp\{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_i)^2\}$$

[2] since the the expression $\beta_0 + \beta_1 x_1 + ... + \beta_n x_n$ is linear in β but not in x, usually the first column of the $\mathbf{X}$ matrix is a column of 1's. In this way $[1, \mathbf{X}]\boldsymbol{\beta}$ becomes linear in the features spaces.

where σ is constant with respect to β, so to maximize the likelihood with respect to β we are interested only in $(y_i - \mu_i)$. Since it is a decreasing function, its maximization is equivalent to a minimization problem:

$$min_\beta(\sum_{i=1}^{n}(y_i - \mu_i)^2)$$

That can also be written as:

$$min_\beta(y - \mu)^{'}(y - \mu)$$

To solve the minimization problem, we compute the derivative with respect to β and we set it equal to 0:

$$\frac{\partial}{\partial\beta}(y - X\beta)^{'}(y - X\beta) = 0$$

$$\frac{\partial}{\partial\beta}(y^{'}y - \beta^{'}X^{'}y - y^{'}X\beta + \beta^{'}X^{'}X\beta) = 0$$

$$-2X^{'}y + 2X^{'}X\beta = 0$$

$$X^{'}X\beta = X^{'}y$$

Assuming that $X^{'}X$ is invertible:

$$\hat{\beta} = (X^{'}X)^{-1}X^{'}y \tag{6.2}$$

$\hat{\beta}$ is the **least square estimate** of β.

Observations:

- its relative estimator $\hat{\beta} = (X^{'}X)^{-1}X^{'}Y$ is an unbiased estimator;
- since it is a linear transformation of Y, is also Normal:

$$\hat{\beta} \sim \mathcal{N}(\beta, \sigma^2(X^{'}X)^{-1})$$

- $\mathbf{X\hat{\beta}}$, geometrically, is the projection of the vector y on the subspace of $\mathbb{R}^n$ spanned by the columns of X.

6.2 Evaluating performances

In order to evaluate the performance of a statistical learning method on a given data set, we need some way to measure how well its predictions actually match the observed data. That is, we need to quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation. In the regression setting, the most commonly used measure is the mean squared error (MSE), given by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\hat{x}_i))^2$$

where $f(\hat{x}_i)$ is the prediction that $\hat{f}$ gives for the i_{th} observation. The MSE will be small if the predicted responses are very close to the true responses, and will be large if for some of the observations, the predicted and true responses differ substantially.[9]
Other metrix are avaiable, like the R^2, the adjusted R^2, however we decided to focus on MSE because we are interested in knowing how much the qualities predicted are close to the real ones.

6.3 Linear Regression

The most basic regression model involves a linear relationship between a qualitative response Y and a single explanatory variable X:

$$Y_i = \beta_0 + \beta_1 x_i \quad i = 1, .., n \tag{6.3}$$

for certain unknown parameters β_0 and β_1. The unknown line

$$y = \beta_0 + \beta_1 x \tag{6.4}$$

is called the regression line. Thus, we view the responses as random variables that would lie exactly on the regression line, were it not for some "disturbance" or "error" term represented by the ε_i [10].
The best estimation for the two unknown parameters β_0 and β_1 that is possible to obtain is such that the resulting line is as close as possible to the training data, so we have:

$$\hat{\mathbf{Y}} = \hat{\beta}_0 + \mathbf{X}\hat{\beta}_1 \tag{6.5}$$

A possible measure of closeness is given by the Least Squares Estimate, as we have seen in section 6.1.1. Using this criterion it is possible to show that:

- $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$
- $\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$

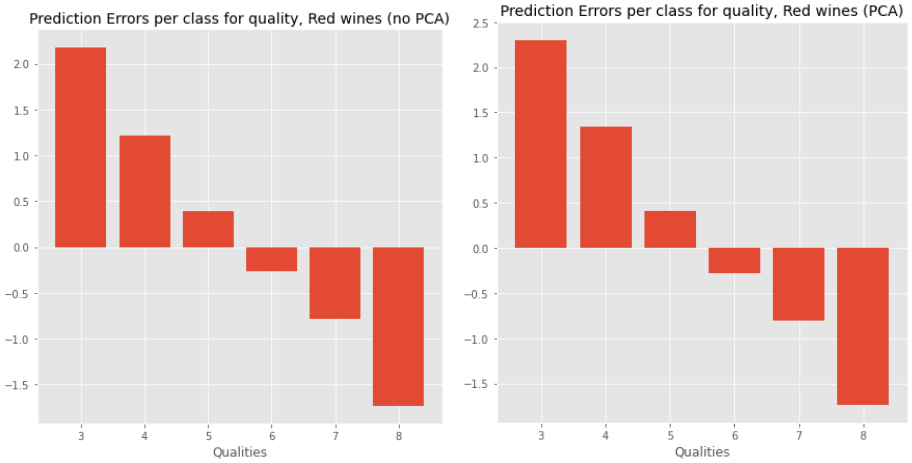
where $\bar{y}$ and $\bar{x}$ are the predictors and samples means.

However in practice, we often have more than one predictor variable. One possible option is to run different linear regression models, one for each predictor, and then combine them, unfortunately this approach is not satisfactory. A better approach consists of extending the linear regression model to receive multiple predictor variables $\mathbf{x}_1, ..., \mathbf{x}_n$, each predictor will receive a different slope. The multiple linear regression model has the form:

$$Y_i = \beta_0 + \beta_1 x_i + ... + \beta_n x_n \tag{6.6}$$

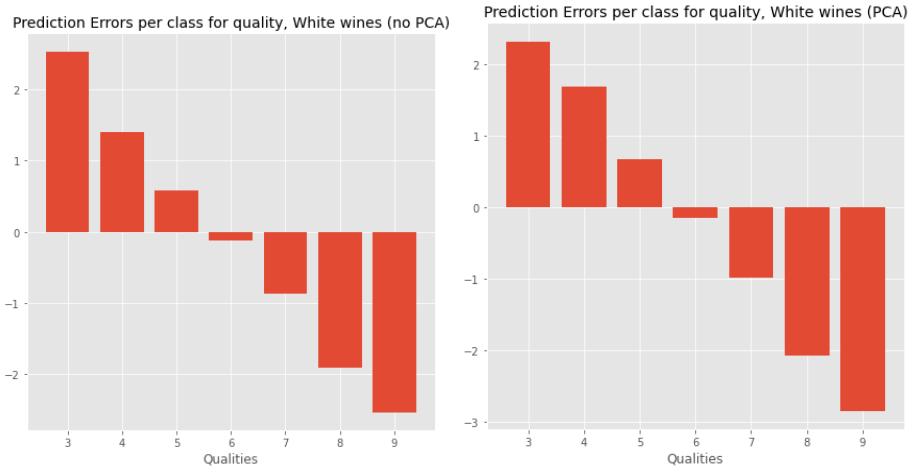
Figure 14 and 15 shows the prediction errors per class, obtained considering the mean of the predicted values and subtracting the expected value. In this way we can see, for each quality score, if our model is able to predict well unseen points or if the prediction is far away from the true label.
For red wines, our model can predict very well wines with qualities between scores 5 and 6 because the prediction error is very low (less than 0.5), while it demonstrates uncertainty with the remaining ones. For white wines we can see more or less the same trend: the model in this case is excellent in predicting

points belonging to the class 6 and less good for the other classes. In particular it has a hard time recognizing wines with labels 3 and 8 for red wines dataset and 3, 8 and 9 for white dataset: this does not surprise us, because at training time we choose to drop wines with this labels, since they were considered outliers. All these observations are coherent with the distributions of the target variable indeed in our datasets the majority of the samples lies between labels 5 and 6, as shown in figure 1.



(a) *No-PCA dataset* (b) *PCA dataset*

Figure 14: Prediction Errors per class (*Red Wine* Dataset)



(a) *No-PCA dataset* (b) *PCA dataset*

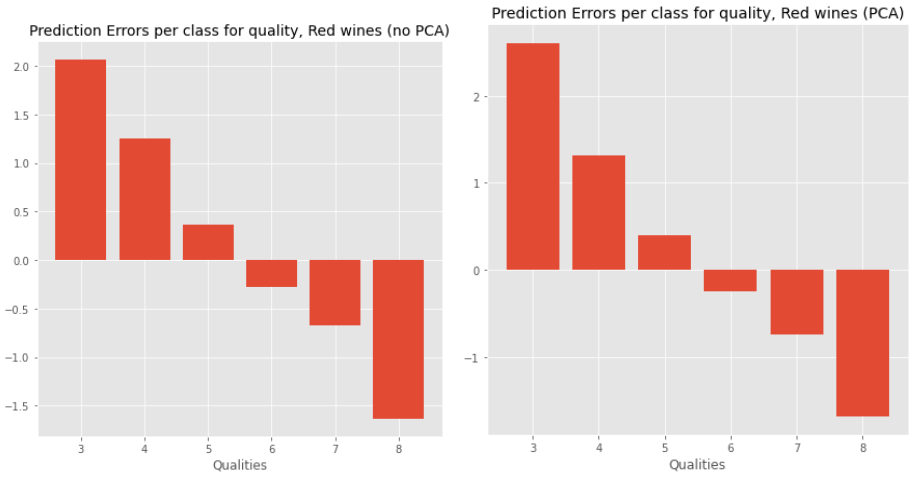
Figure 15: Prediction Errors per class (*White Wine* Dataset)

6.4 Polynomial Regression

In some cases the true relationship between the predictors and the response is not linear. To manage this situation it is possible to use the polynomial regression which extends the linear model to accommodate non-linear relationships. It consists of including non-linear associations in a linear model by including transformed versions of the predictors. If, for example, we add the squared version of the predictor we build a polynomial up to the second degree ($p = 2$), and if we have only two predictors, the model becomes:

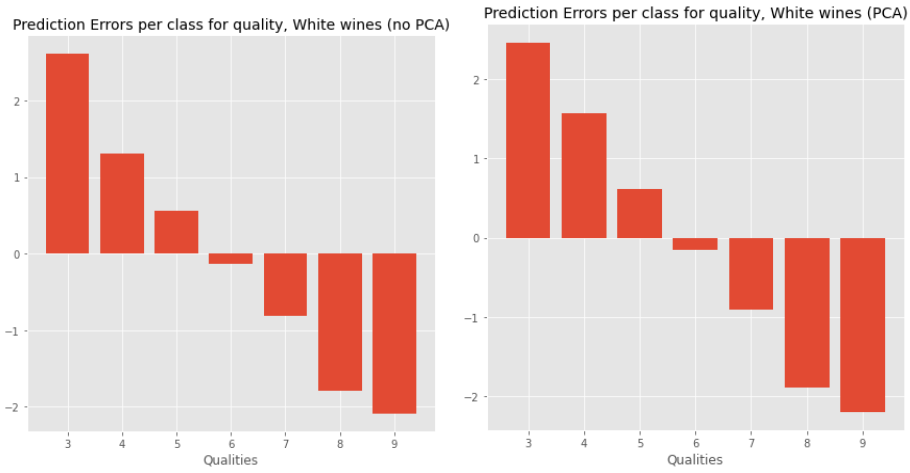
$$Y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2 + \beta_4 x_2^2 + \beta_5 x_1 x_2 \tag{6.7}$$

It is still a linear in β because we apply the polynomial transformation to the feature before fitting the model. Thus, it can be solved by using the standard algorithm for the linear regression, in particular it is a multiple linear regression model in which for each predictor it is also present its squared version and all the possible interactions between them, up to the p^{th} degree.
Figure 16 and 17 shows prediction errors obtained by applying polynomial regression to our datasets. The results do not differ much from the previous case. For red wine dataset, the model is able to predict classes 5 and 6 with extremely high precision, which goes down for the other classes, until it reaches very bad results in correspondence of classes unseen at training time, like we discuss for linear regression.



(a) *No-PCA dataset* (b) *PCA dataset*

Figure 16: Prediction Errors per class (*Red Wine* Dataset)



(a) *No-PCA dataset* (b) *PCA dataset*

Figure 17: Prediction Errors per class (*White Wine* Dataset)

6.5 Ridge Regression

Starting from the linear regression model (6.1), we center the x-column in 0, so that the estimate of the intercept is set at $\beta_0 = \bar{y}$, so we don't consider the usual first column $[1...1]^T$.
The aim is to estimate β, so we are looking for $\hat{\beta}$. We have to choose a prior distribution of β, and eventually obtain the posterior, which is proportional to the likelihood multiplied by the prior.
In particular since Y and the x columns are centered, and assuming that $y|\beta, X \sim \mathcal{N}(X\beta, \sigma^2\mathbb{1})$ the likelihood is:

$$(2\pi\sigma^2)^{-\frac{n}{2}} \exp\{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_i x_{1i} - ... - \beta_p x_{pi})^2\} \tag{6.8}$$

and we assume σ^2 is known.
We assume that the prior distribution of β is such that β_k's are normal *i.i.d.* centred in zero, namely $\beta \sim \mathcal{N}(0, \sigma^2\mathbb{1})$. The prior will then have the following distribution:

$$\prod_{k=1}^{p} \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\{-\frac{\beta_k^2}{2\sigma_0^2}\} \tag{6.9}$$

Then posterior will be proportional to:

$$\exp\{-\frac{1}{2\sigma^2}(\sum_{i=1}^{n}(y_i - \beta_i x_{1i} - ... - \beta_p x_{pi})^2 + \frac{\sigma^2}{\sigma_0^2}\sum_{k=1}^{p}\beta_k^2)\} \tag{6.10}$$

where $\lambda = \frac{\sigma^2}{\sigma_0^2}$ is our tuning parameter that defines how much penalty, given the L2-norm $\sum(\beta_k^2)$, the model will receive.
We consider now the MAP (Maximum A Posteriori) estimate, obtained by maximizing the posterior or, equivalently, by minimizing the negative log-likelihood.

We obtain:

$$min_\beta(\sum_{i=1}^{n}(y_i - \beta_i x_{1i} - ... - \beta_p x_{pi})^2 + \lambda \sum_{k=1}^{p} \beta_k^2) \tag{6.11}$$

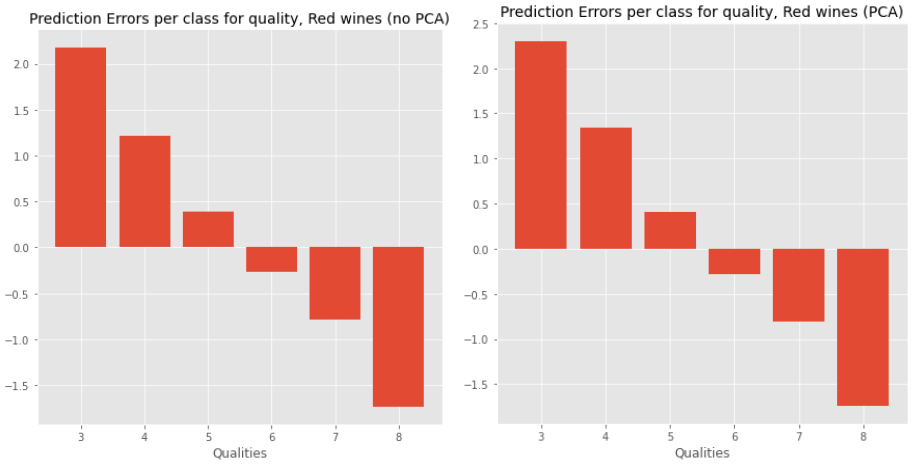
It can be show by algebra that the Ridge regression estimate is:

$$\tilde{\boldsymbol{\beta}} = (\lambda \mathbf{I}_p + \mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'y \tag{6.12}$$

If $\lambda = 0$, we have no penalty on the model, that will lead to the same results of the standard least squares estimates.
If instead $\lambda \to \infty$, we impose a high penalty and $\tilde{\beta} \to 0$ since the model tries to minimize its values.
The optimal value for λ will be found by a cross validation process with 5-folds over the following values:

- $\lambda = [0.001, 0.01, 0.1, 1, 10, 100, 1000]$

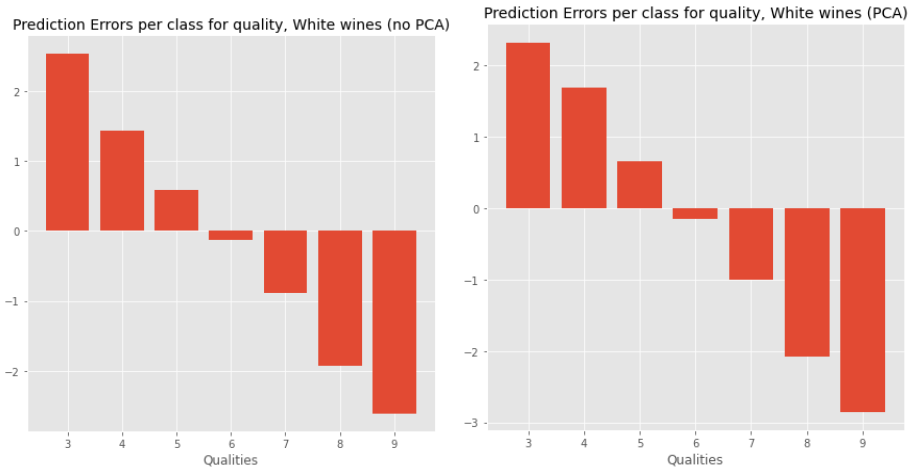
Prediction errors for Ridge regression model are shown in Figure 18 and 19. Classes that are predicted better are, like the other cases, 5 and 6 for the red dataset, and 6 for the white ones. Also in this case we do not notice any deviation from what we expect from the distribution of our data.



(a) *No-PCA dataset* $\lambda = 10$

(b) *PCA dataset* $\lambda = 10$

Figure 18: Prediction Errors per class (*Red Wine* Dataset)



(a) *No-PCA dataset* $\lambda = 100$

(b) *PCA dataset* $\lambda = 100$

Figure 19: Prediction Errors per class (*White Wine* Dataset)

6.6 Lasso Regression

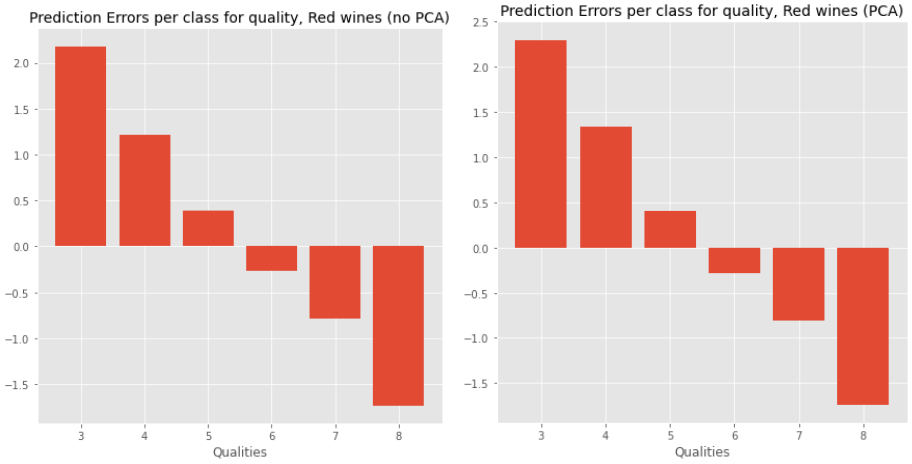
In the Lasso regression, we use the same approach used in the Ridge regression, but instead of using as penalty therm the L2-norm, we use the L1-norm, defined as $\sum_{k=1}^{p} |\beta_k|$, so the corresponding minimization problem becomes:

$$min_\beta = \sum_{i=1}^{n}(y_i - \beta_i x_{1i} - ... - \beta_p x_{pi})^2 + \lambda \sum_{k=1}^{p} |\beta_k| \tag{6.13}$$

The solution, for which no explicit formula exist, (numerical optimization needed) is called the **Lasso estimates**.
The peculiarity of the LASSO is that the absolute value of the model forces some coefficient to be exactly equal to 0, and this performs an automatic features selection.
As before when we have $\lambda = 0$ we have no penalty on the model and the result is the same of the standard least squares estimates, instead if $\lambda \to \infty$ the model becomes the null model since all coefficient are equal to zero, except for the intercept to the axis that was removed from the model a priori. Even in this case, the optimal value for λ is found by a cross-validation process with a 5-fold technique over the following values:

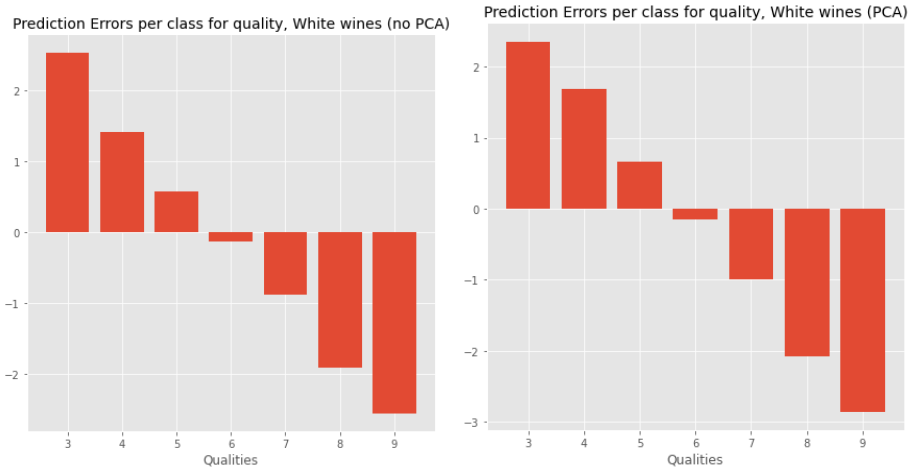
- $\lambda = [0.001, 0.01, 0.1, 1, 10, 100, 1000]$

Final results are shown in Figure 20 and 21. The prediction error tends to decrease as we approach the classes that present more samples.



(a) *No-PCA dataset* $\lambda = 0.001$ (b) *PCA dataset* $\lambda = 0.001$

Figure 20: Prediction Errors per class (*Red Wine* Dataset)



(a) *No-PCA dataset* $\lambda = 0.001$ (b) *PCA dataset* $\lambda = 0.01$

Figure 21: Prediction Errors per class (*White Wine* Dataset)

6.7 Regression Results

Table 4 shows the achieved results of all the regression models previously described in terms of Mean Squared Error. An overall view show us how all the models have reached a very good results: all the squared errors belong to the interval $[0.40, 0.68]$. There was no algorithm that performed significantly better than the others, but if we have to pick one that performed well for both datasets, Polynomial regressor seems a good choice.

The results obtained without using the PCA are a little better than the one with the PCA but not too far away. Moreover we consider that reducing the number of features with the help of the PCA was a good choice since the results are similar.

A final consideration might be the imbalanced distribution of the target variable "quality", which certainly influenced the predictions: having the majority of the label between scores 5 and 6 led the algorithms to became skewed as well. As a result, the models proposed shows a similar trend.

	Red PCA	*Red No PCA*	*White PCA*	*White No PCA*
Linear Regression	0.42	0.40	0.68	0.59
Polynomial Regression	0.43	0.40	0.61	0.57
Ridge Regression	0.42	0.40	0.68	0.60
Lasso Regression	0.42	0.40	0.68	0.59

Table 4: Regression Results in terms of MSE

7 Conclusions

The current thesis introduced an analysis performed on the UCI Machine Learning Wine Quality Data Set [2].

In particular, starting from two different sample sets - one focused on red wines and the other one on white wines - binary classification was performed, having wines' final quality as target variable (*Good* or *Bad* wine quality).

Before applying classification algorithms, the data was preprocessed in different steps:

1. **Feature scaling**: data was standardized, according to the estimates of mean and standard deviation computed with the bootstrap method;
2. **Dimensionality Reduction** with **PCA** to decrease the number of features describing the dataset, retaining at least 90% of its cumulative variance;
3. **Oversampling with ADASYN** to balance the White Wine data set, which were imbalanced on the positive class.

In order to find the best hyperparameters for each classification algorithm, a grid search with 5-fold cross validation was applied.

Models were trained on both oversampled and non-oversampled training sets. Results were evaluated in terms of accuracy and F1-score.

The proposed methods were:

- **Logistic Regression**;
- **Support Vector Machines**.

Subsequently, regression was performed to predict wines' quality over all possible output values.
Before applying the regression models, outliers were found and dropped in order to achieve a better result.
The proposed methods were:

- **Linear Regression**;
- **Polynomial Regression**;
- **Ridge Regression**;
- **Lasso Regression**.

All algorithms were performed on both PCA-reduced and standard (without PCA) datasets.

References

[1] Fernando Almeida Telmo Matos José Reis Paulo Cortez, António Cerdeira. Modeling wine preferences by data mining from physicochemical properties. *Elsevier*, 05 2009.

[2] UCI Machine Learning. Wine quality data set. `https://archive.ics.uci.edu/ml/datasets/Wine+Quality`.

[3] Machine Learning Mastery. A gentle introduction to the bootstrap method. `https://machinelearningmastery.com/a-gentle-introduction-to-the-bootstrap-method/`, May 2018.

[4] Kevin P. Murphy. *Machine Learning A Probabilistic Perspective*. 2012.

[5] Haibo He, Yang Bai, Edwardo Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. pages 1322 – 1328, 07 2008.

[6] Towards Data Science. Logistic regression explained. `https://towardsdatascience.com/logistic-regression-explained-9ee73cede081`.

[7] Research Gate. An example of svm classification. `https://www.researchgate.net/figure/An-example-of-SVM-classification-An-example-of-SVM-classification_fig1_336085357`, September 2019.

[8] Wikipedia. Hilbert space. `https://en.wikipedia.org/wiki/Hilbert_space#Definition`.

[9] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[10] D.P. Kroese, Z.I. Botev, T. Taimre, and R. Vaisman. *Data Science and Machine Learning: Mathematical and Statistical Methods*. Chapman & Hall/CRC machine learning & pattern recognition. CRC Press, Boca Raton, 2019.

[11] Wikipedia. Cross-validation (statistics). `https://en.wikipedia.org/wiki/Cross-validation_(statistics)`.