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).

In this work we will focus on the first task, the binary classification.

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

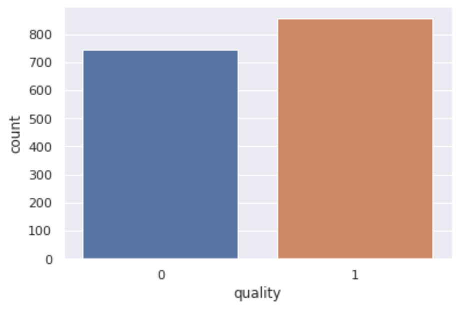
Each datum sample is characterized by the same 12 attributes, shown in Table[1]. They give additional information about the wines, such as density, pH, alcohol and so on. All the attributes are described by integer and real numbers.

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)

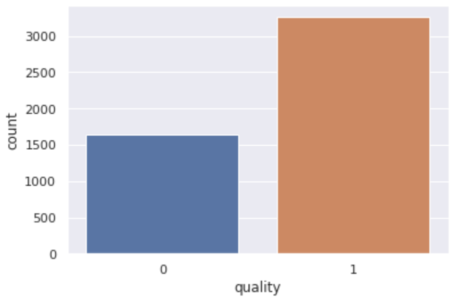
Table 1: Attributes description

The attribute that interests us most is `quality`, which represents wines' final score and is our target output. 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, the wine is good; otherwise, it is bad.

As it can be seen in Figure[1], despite the Red Wine dataset is balanced, the number of wines in White Wine dataset is heavily skewed towards good 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 1: Distribution of the target variable "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)

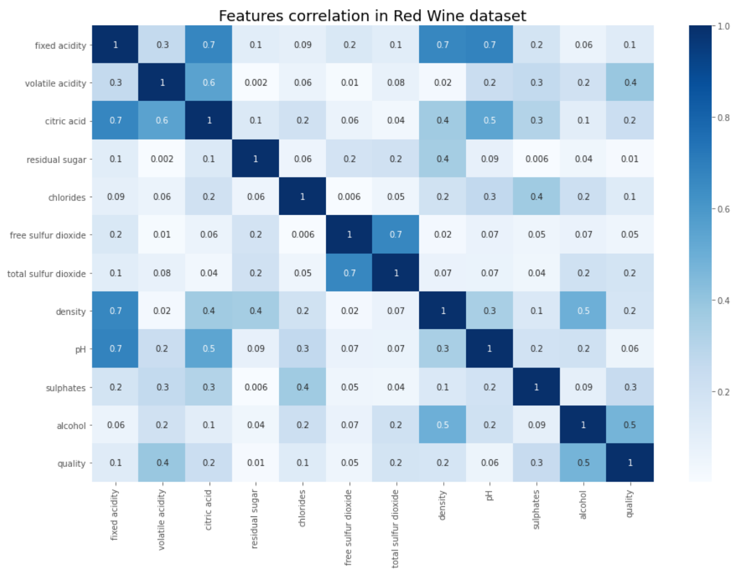
$$r_{ij} = \frac{cov(X_i, X_j)}{\sigma_{Xi}\sigma_{Xj}} \tag{3.1}$$

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. So, the higher the absolute value of the coefficient, the more correlated the two variables are.

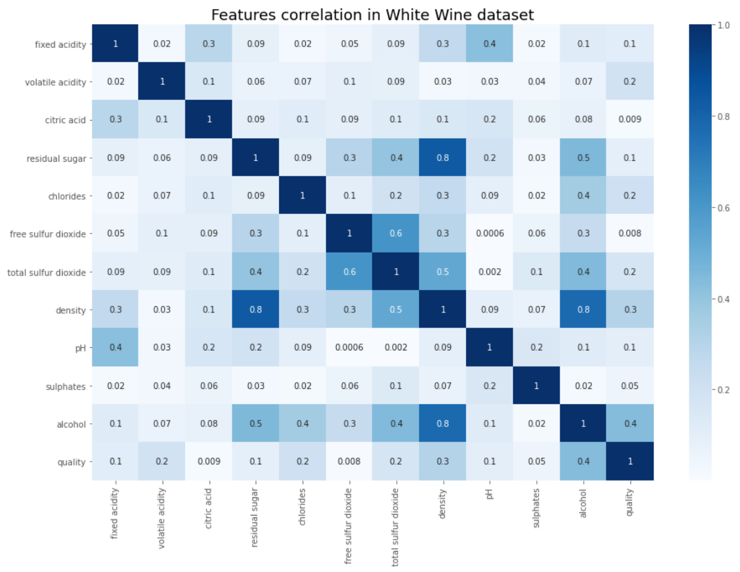
Figure [2] shows there are some low correlated features to the target variable `quality`, in both datasets. In particular, the attributes with a correlation belonging to the interval $[-0.07, 0.07]$ are dropped, so as to keep working only on relevant information.

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 2: 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?

So, 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 Components 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 point and their projection is minimized and the variance of 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 as 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 the PCs are eigenvectors of S and their eigenvalues satisfy the condition

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

so that

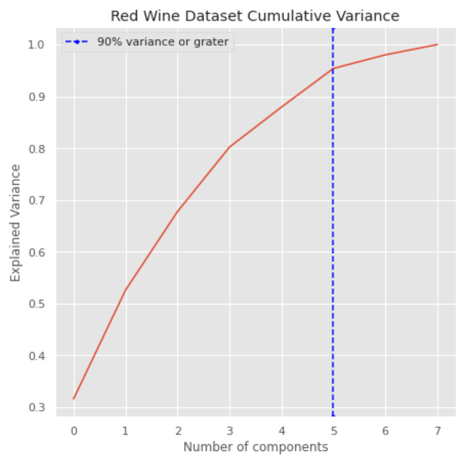
$$Var(u_1) \geq Var(u_2) \geq ... \geq Var(u_D)$$

The total variance of the data is consequently decomposed as:

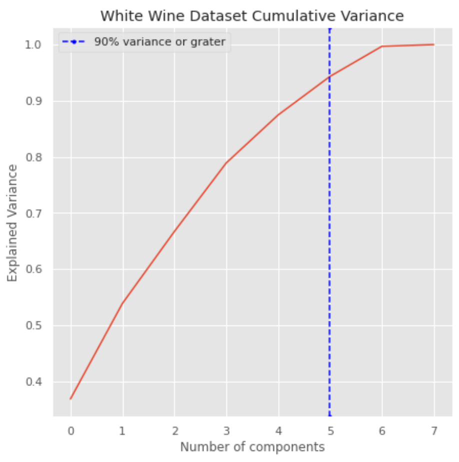
$$\sum_{i=1}^{D} Var(u_i) = \sum_{i=1}^{D} \lambda_i = \sum_{i=1}^{D} Var(x_i)$$

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 [3] shows, as for both datasets, 5 PCs were selected out of 9.



(a) *Red Wine*



(b) *White Wine*

Figure 3: Cumulative Variance of Principal Components

4.3 Oversampling with SMOTE

When dealing with imbalanced data sets, two main solutions can be applied:

1. Usage of resampling techniques
2. Change in the evaluation metrics (discussed in Chapter 5).

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 in order 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 (SMOTE, for example).

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 most widely used approach is the Synthetic Minority Oversampling Technique, or SMOTE.

SMOTE first selects a minority class instance a at random and finds its k=*5* nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b [5].

Starting from the imbalanced distribution presented in White Wine (see Figure [**??**]), the data set reach the following balanced configuration: from 1312 to 2606 *False* labels, reaching the same quantity of the *True* ones.

5 Classification

5.1 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 passing students 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** = $2 \cdot \frac{precision \cdot recall}{precision + recall}$, a balance between precision and recall, which is ideal in the case of imbalanced data.

5.2 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.

5.3 Classification Algorithms

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.

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.

Different algorithms and their results will be introduced.
For White Wine dataset we have at our disposal two different train sets: one resulting from the PCA transformation and one both reduced and then over-sampled with SMOTE. The achieved performance will be compared, so as to find the best one.

5.3.1 Logistic Regression

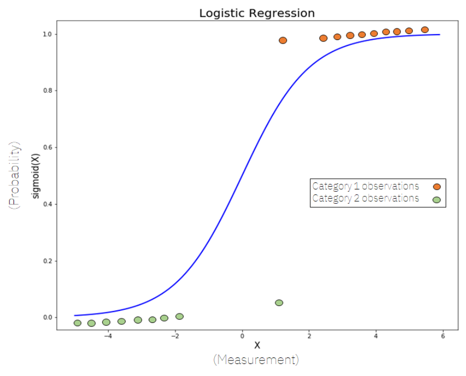


Figure 4: General Logistic Regression [6]

Logistic Regression (LR) is a binary classification model, 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, $\mathbf{w}$ 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:

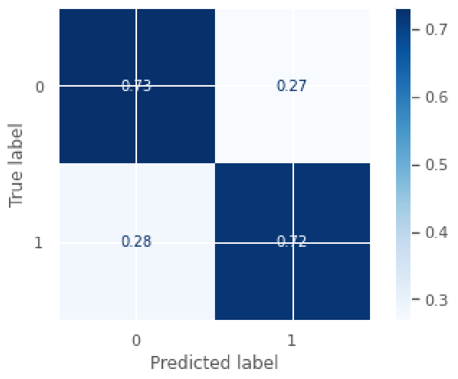
$$l(w_0, w_1) = \prod_{i;y_i=1} p(y_i) \prod_{i;y_i=0} (1 - p(y_i)) \tag{5.7}$$

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:

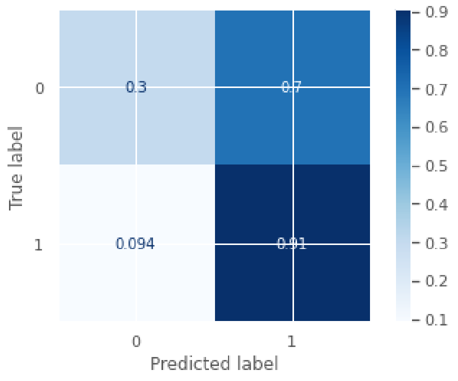
- **C**: 0.001, 0.01, 0.1, 1, 10, 100, 1000 .

Figure [5] and [6] show the confusion matrices obtained with the Logistic Regression algorithm on all data sets. The resulting accuracies can be found in Table 2 and F1-scores 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.

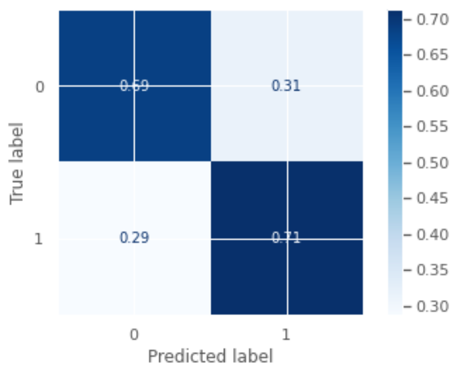


(a) *C = 0.1*

Figure 5: Confusion Matrices for Logistic Regression on *Red Wine* Dataset



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



(b) *SMOTE oversampling C = 0.1*

Figure 6: Confusion Matrices for Logistic Regression on *White Wine* Dataset

5.3.2 K-Nearest Neighbors (KNNs)

KNN is a multiclass classifier whose purpose is to find a predefined number K of training samples closest in distance to the new point, and predict the label from these. It looks at the K points in the training set that are nearest to the test input x, basing on distance metric, counts how many members of each class are in this set and returns the empirical fraction:

$$p(y = c|x, D, K) = \frac{1}{K} \sum_{i \in N_K(x,D)} \mathbb{I}(y_i = c) \tag{5.8}$$

where $N_K(x, D)$ are the indices of the K nearest points to x in D and $\mathbb{I}(e)$ is the indicator function, defined as:

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if e is true} \\ 0 & \text{if e is false} \end{cases} \tag{5.9}$$

This method is an example of **memory-based learning** or **instance-based learning**, because it does not have a learning process to construct a general internal model, but simply stores all the training samples.

The strong made assumption is that closest samples belongs to the same class, so they have to have the same labels.

The tuning of K parameter is important because if K is too small, our classifier will be sensible to noise; on the other hand, too high values of K will introduce samples belonging to other class.

Another key factor is the choice of the metric, that influences the computation of the distances between points in the given space (the KNN algorithm is based on the comparison of the distances).

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

- **K**: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21;
- **weights**: 'uniform', 'distance';
- **metric**: 'euclidean', 'manhattan'.

where **Euclidean** distance is defined as:

$$d_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d}(x_{ik} - x_{jk})^2} \tag{5.10}$$

and **Manhattan** distance is defined as:

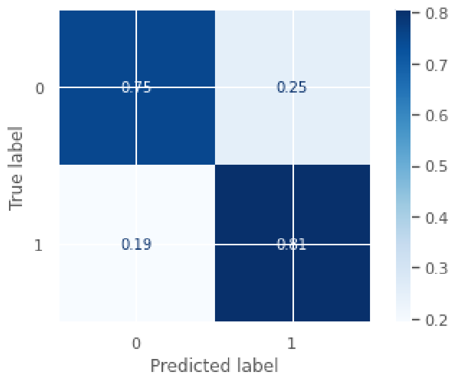
$$d_{ij} = \sum_{k=1}^{d}(|x_{ik} - x_{jk}|) \tag{5.11}$$

where d is our dimensionality and p_k and q_k are, respectively, the k^{th} attributes.

This classifier also uses a weight function for the predictions and its default value is **uniform**, which means all points in each neighborhood are weighted equally. Another weight function was tested too, the so called **distance**, which weights points by the inverse of their distance. In this case, closer neighbors will have a greater influence than neighbors which are further away.

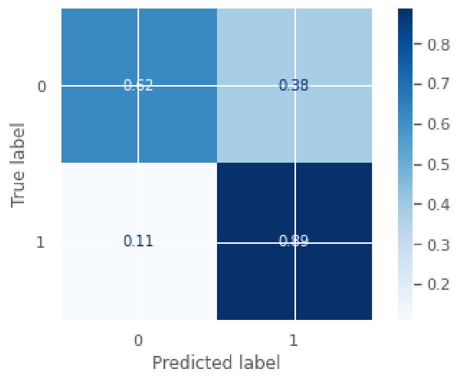
Figure [7] and [8] show the confusion matrices obtained with KNN on all data sets: the model works quite well on classifying both true and negative label. The application of the SMOTE technique on White Wine dataset makes no changes, or at least so it seems from the graphs.

The accuracy values and F1 scores can be found in Table [2] and [3]. As we can understand from the first ones, it seems the SMOTE model works worse, but this is not coherent with what we see in the confusion matrices: this means that the accuracy is not the right metric to evaluate the model. F1 score, instead, shows a more realistic view, in which performances remains more or less the same.

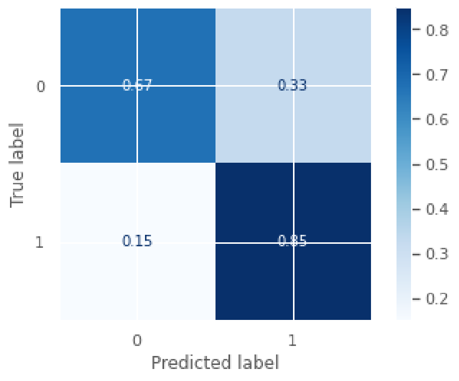


(a) *K = 21, weights = 'distance', metric = 'euclidean'*

Figure 7: Confusion Matrices for KNN on *Red Wine* Dataset



(a) *Non-oversampled dataset K = 13, weights = 'distance', metric = 'euclidean'*



(b) *SMOTE oversampling, K = 1, weights = 'uniform', metric = 'euclidean'*

Figure 8: Confusion Matrices for KNN on *White Wine* Dataset

Curse of dimensionality The KNN classifier is simple and can work quite well, provided a good distance metric and enough labeled training data. However, the main problem with KNN classifiers is that they do not work well with high dimensional inputs. The poor performance in high dimensional settings is due to the **curse of dimensionality**.

The phrase, attributed to Richard Bellman, was coined to express the difficulty of using brute force (a.k.a. grid search) to optimize a function with too many input variables [4].

This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient: the method is no longer very local. The trouble with looking at neighbors that are so far away is that they may not bee good predictors about the behavior of the input-output function at a given point.

One possible solution to this problem is to reduce the dimensionality of our dataset, as we did using PCA (see chapter 4.2).

5.3.3 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:

$$f(x) = \langle \mathbf{wx} \rangle + b \tag{5.12}$$

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 $\operatorname{sgn}(f(\mathbf{x}))$.

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

$$f(x) = \langle \mathbf{wx} \rangle + b = 1 \tag{5.13}$$

$$f(x) = \langle \mathbf{wx} \rangle + b = -1 \tag{5.14}$$

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]):

$$margin = \frac{1}{\|\mathbf{w}\|} \tag{5.15}$$

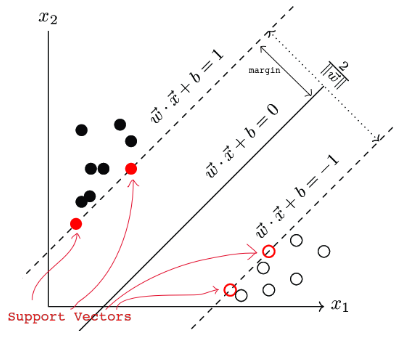

x_2
x_1
$\vec{w} \cdot \vec{x} + b = 1$
$\vec{w} \cdot \vec{x} + b = 0$
$\vec{w} \cdot \vec{x} + b = -1$
margin
$\frac{2}{\|\vec{w}\|}$
Support Vectors

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:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s. t.} \quad \forall i, y_i(\langle \mathbf{w}, x_i\rangle + b) > 1 \tag{5.16}$$

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 (5.16) a new therm:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \quad \text{s. t.} \quad \forall i, y_i(\langle \mathbf{w}, x_i\rangle + b) > 1 - \xi_i \quad \text{and} \quad \xi_i >= 0 \tag{5.17}$$

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 $\phi : \chi \rightarrow H$ 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 $K : \chi \times \chi \rightarrow \mathbb{R}$, implementing the inner product in the features space, which has lower computational cost. K is called *kernel function* and is such that:

$$K(x_i, y_i) = (\phi(x_i), \phi(y_i)) \tag{5.18}$$

In order to apply this mapping, the kernel function must satisfy **Mercer's theorem**, stating that: a symmetric function $K : \chi \times \chi \rightarrow \mathbb{R}$ 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 $x^T Gx \geq 0 \quad \forall(x) \in \mathbb{R}^n$.

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:

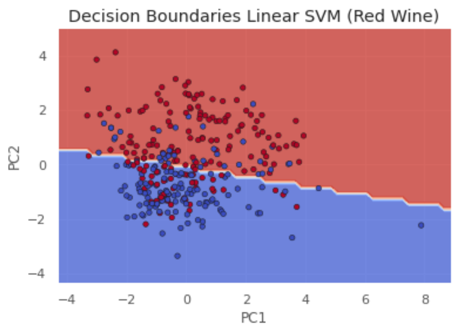
$$k(x, x') =< x, x' > \tag{5.19}$$

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:

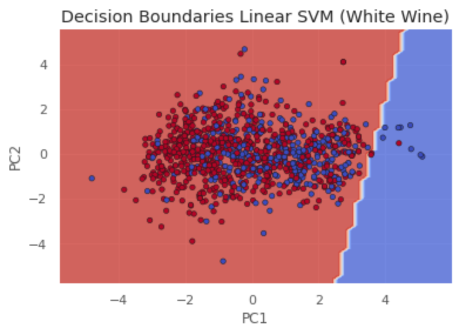
- $C = [10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3]$.

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, although the accuracy is not very high. 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 SMOTE is applied, performances improves.

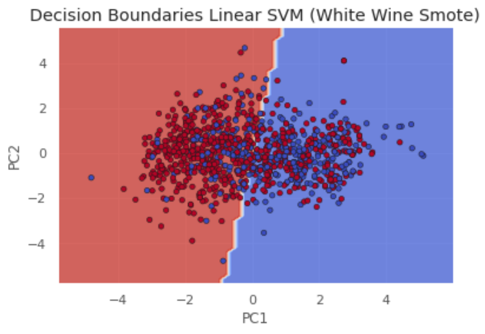


(a) *C = 0.01*

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



(a) *Non-oversampled dataset C = 10*



(b) *SMOTE oversampling C = 1000*

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:

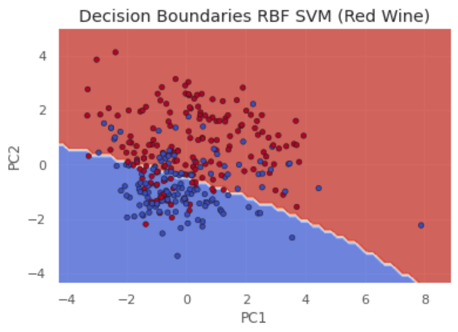
$$k(x, x') = e^{(-\gamma \|x-x'\|^2)} \tag{5.20}$$

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:

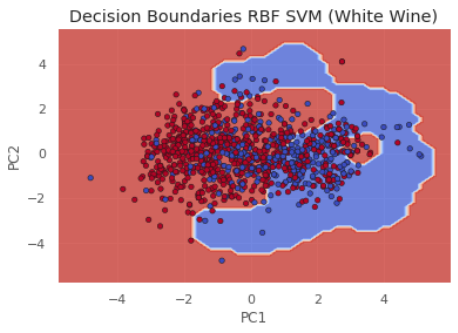
- $C = [10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3]$;
- $\gamma = [10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}, 0.5, 1, 10, 10^2]$.

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, there is no improvement.

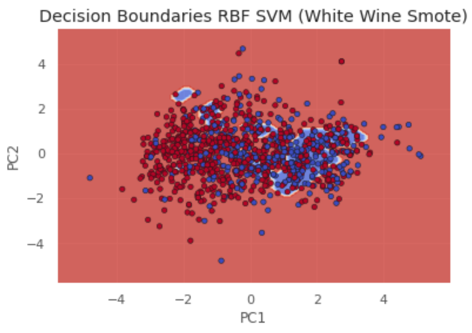


(a) C = *1000*, $\gamma = 0.001$

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)

5.3.4 Decision Trees

When using Decision Trees, the goal is to create a model predicting the value of a target variable by learning simple decision rules inferred from the data features. Their main advantage is interpretability, but they might easily lead to overfitting because of a poor generalization.

Decision Trees can be applied to both regression and classification problems. The steps to build a general DT are the following ones:

1. The set of possible input values $X_1, ..., X_p$, called the *predictor space*, is divided into J distinct and non-overlapping regions $R_1, ..., R_j$

2. For every observation that falls into R_j, the prediction is the mean of the response values for the training observations in R_j

3. The predictor space is divided into high-dimensional rectangles, called *boxes*. The goal is to find the boxes $R_1, ..., R_j$ that minimize a certain value: the *Residual Sum of Squares* (RSS) for regression problems (5.21);

the *Classification Error Rate* (E) in the other case (5.22).

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2 \tag{5.21}$$

where $\hat{y}_{R_j}$ the mean response for the training observations within the jth box,

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{5.22}$$

where $\hat{p}_{mk}$ is the portion of training observations in the mth region belonging to the kth class.

Since it is computationally infeasible to consider all possible partitions of the feature spaces into J boxes, the used approach is top-down and greedy, also known as *recursive binary splitting*: starting from the root of the tree, two or more branches are created at each split on the prediction space; at each step of the tree-building process, the best split is created, according to the chosen metrics, without taking into consideration suboptimal partitions that may lead to better results in the future.

The classification tree aims to predict the most commonly occurring class of training observations the current observation belongs to, inside its region. They are built using recursive binary splitting, based on the *classification error rate* E (5.22), the fraction of the training observations in that region that do not belong to the most common class. However, E is often not sensitive enough for tree-growing, so two other measures are preferred:

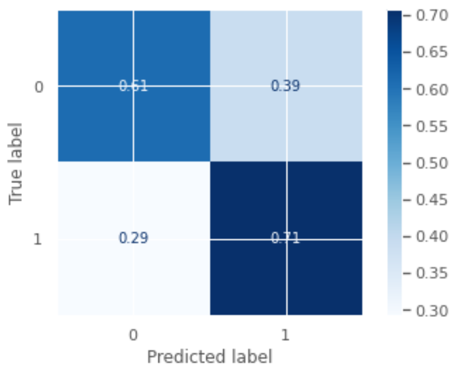
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{5.23}$$

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk}) \tag{5.24}$$

where G (5.23) is the *Gini index* and is a measure of the total variance across the K classes (often referred to as a measure of node purity), and D (5.24) is the *cross-entropy.*

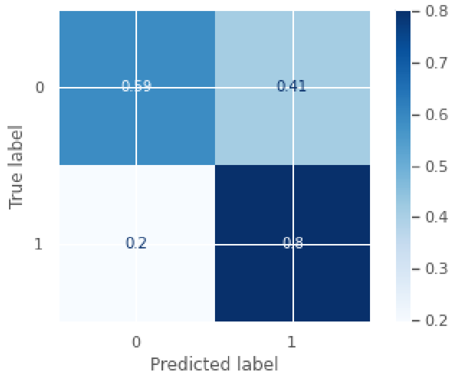
In the proposed work, the tuned parameters are: *criterion*, which states if *Gini index* or *cross-entropy* is used to classify objects as belonging to a particular class; the maximum depth of the tree; the minimum number of samples required to split a node and per each leaf.

According to the results are shown in Figures 14 and 15, the model works well on both datasets, being capable of well classify both Positive and Negative labels. The application of SMOTE oversampling seems not improves our model, also in terms of accuracy.
An example of Decision Tree can be found in Figure 16.

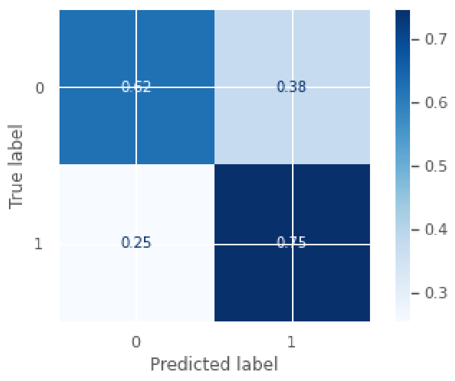


(a) *criterion: 'entropy', max depth: 16, min samples leaf: 1, min samples split: 2*

Figure 14: Confusion Matrices for Decision Trees on *Red Wine* Dataset



(a) *Non-oversampled dataset criterion: 'entropy', max depth: 21, min samples leaf: 1, min samples split: 2*



(b) *SMOTE oversampling, criterion: 'gini', max depth: 21, min samples leaf: 1, min samples split: 2*

Figure 15: Confusion Matrices for Decision Trees on *White Wine* Dataset

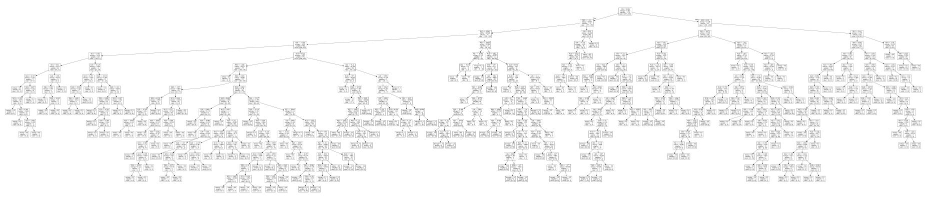


Figure 16: Resulting Decision Tree for *Red Wine* dataset

5.3.5 Random Forest

A Random Forest Classifier can be thought of as an ensemble of different Decision Trees, fit on various subsamples of the dataset, whose decisions are averaged in order to better generalise the model and therefore prevent overfitting.

Random Forests provide an improvement over *bagged trees.* In particular, *bagging* - or *bootstrap aggregation* - is a procedure for reducing the variance of a statistical learning method.

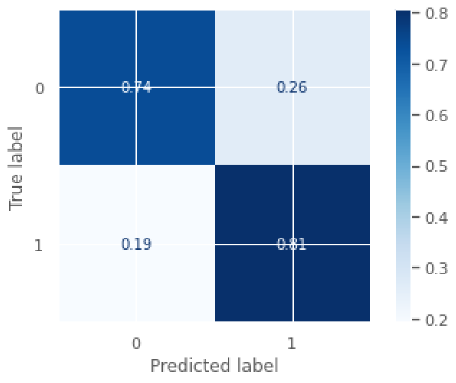
Given a set of independent observations $Z_1, ..., Z_n$, having variance σ^2, being $\bar{Z}$ the mean of those observations, the mean of $\bar{Z}$ is ${\sigma^2}/{n}$, which implies that averaging a set of observations reduces the variance. We usually don't have access to multiple trees, so bootstrap is our solution: having generated B different bootstrapped training data sets, the method is trained on the bth set and the prediction $\hat{f}^{*b}(x)$ at the point x is obtained; then, all predictions are averaged to compute the so called *bagging*, as follows:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x) \tag{5.25}$$

In the case of classification trees, the class predicted by each of the B trees is recorded and the overall prediction is the *majority vote*, the most commonly occurring class.

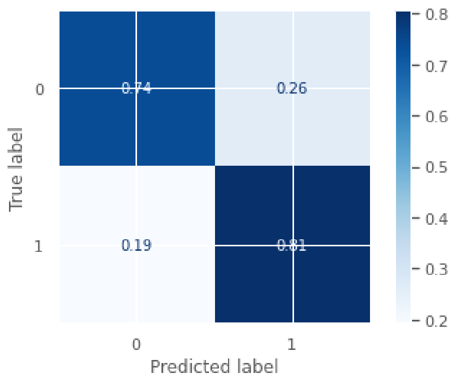
Random Forests improve bagged trees by decorrelating the trees, which results in a reduction of the variance. A number of decision trees is built on bootstrapped training samples and, each time a split has to be created, a random selection of m out of p predictors is chosen as split candidates. A fresh selection of m candidates is taken at each split. Typically, $m \approx \sqrt{p}$.

The tuned parameters are the number of predictors p and the criterion for classifying objects. According to the confusion matrices (Figure 17 and 18), the model perform very well in classify both label in both dataset. The results are very accurate and also when the model is applied on White Wine dataset (the most unblanced ones) it is able to achieves very high results. The performance drops when SMOTE oversampling is applied: apparently, the model didn't need an oversampling of the training data and it was already able to well classify data. In this case, the oversampling brings the model to have an overfitting behaviour.

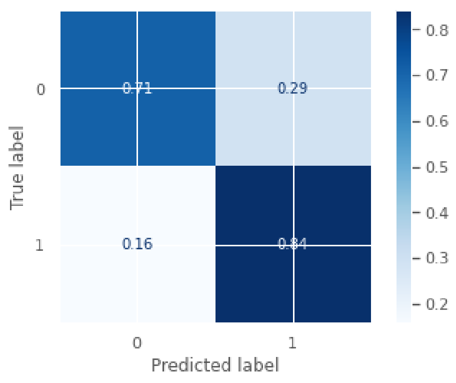


(a) *criterion: 'entropy', n estimators: 301*

Figure 17: Confusion Matrices for Random Forests on *Red Wine* Dataset



(a) *Non-oversampled dataset criterion: 'entropy', n estimators: 301*



(b) *SMOTE oversampling, criterion: 'entropy', n estimators: 601*

Figure 18: Confusion Matrices for Random Forests on *White Wine* Dataset

5.4 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 SMOTE oversampling for White Wine dataset seems to have brought no improvement, indeed the accuracy score remained unchanged (0.69). However, this is not coherent with what we see in the table 3: the F1 score has increased significantly, from 0.24 to 0.61. This means that the accuracy is not the right metric to evaluate the model when dealing with imbalanced datasets. F1 score, in this case, shows a more realistic view. Models that performs best are Random Forests, which obtained the highest F1 score and KNN, which obtained the highest accuracy

score with White Wine dataset.
An overall view on the application of the SMOTE technique shows improvements in about half of the cases.

	Red Wine	*White Wine*	*White Wine SMOTE*
Logistic Regression	0.72	0.70	0.70
KNN	0.78	0.88	0.78
Linear SVM	0.73	0.69	0.69
RBF SVM	0.74	0.76	0.78
Decision Tree	0.68	0.73	0.71
Random Forest	0.78	0.78	0.80

Table 2: Classification Results in terms of accuracy

	Red Wine	*White Wine*	*White Wine SMOTE*
Logistic Regression	0.74	0.41	0.61
KNN	0.76	0.68	0.68
Linear SVM	0.73	0.24	0.61
RBF SVM	0.73	0.60	0.58
Decision Tree	0.63	0.59	0.59
Random Forest	0.78	0.75	0.70

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

6 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 the 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 90% of its cumulative variance;
3. **Oversampling with SMOTE** 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**;
- **K-Nearest Neighbors**;
- **Support Vector Machines**;
- **Decision Trees**;
- **Random Forests**.

The most performing ones were Random Forest and KNN.

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] Yunqian Ma Haibo He. *Imbalanced Learning: Foundations, Algorithms, and Applications.* 2013.

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