Mathematics for Machine Learning
*wine quality data-set, **tesina***

Tommaso Massaglia, S292988

Abstract

The following work will be a comprehensive analysis of the *Wine Quality Data-set* found in the UCI machine learning repository. The final goal of this experience was to train and compare different classification/regression methods on the data-set following a comprehensive analysis of the latter carried through statistical and mathematical methods with the goal of cutting out the less important attributes, augmenting or balancing the data, and overall making the subsequent prediction tasks more precise. Regarding the prediction, the target attributes chosen were 'type', which consisted in a binary classification between red/white wine, and 'quality', a measure from 1 to 10 which was carried on through regression and support vector machines. Overall the achieved precision on the task was satisfying, and some prepossessing pipelines were shown to perform admirably.

The code used to conduct the analysis can be found *here*.

I. Introduction

TO support the growth of the wine industry from luxury to generalized good, the industry began collecting several qualitative and descriptive measures during the production process of it to produce certifications based on them, which quickly became a key item in guaranteeing the safety and quality of the product. Wine certification is usually assessed through physio-chemical measurements performed in a laboratory environment on values such as density, alcohol or pH values, and sensory tests that rely mainly on an expert's evaluation, making wine classification a difficult task.

The data-set that will be analyzed in the following paper contains data on a total of 6497 different wine samples, and was used for a study carried on in 2009 [1] which had the goal of developing a machine learning model able to predict the human sensory classification of a wine from the laboratory-found values, ultimately creating something that's *'useful to support the oenologist wine tasting evaluations and improve wine production'*.

The analysis will be carried with the following order: section II will cover the exploratory of the data-set, more qualitative in nature, section III will cover the available preprocessing methods and their influence on the data-set, section IV explains the statistical and mathematical models that were trained in the experiments, and finally section V will analyse and compare between models/pipelines the result of the training.

II. Data Exploration

A. Data-set description

The data-set, found *here*, is provided as two separate sheets, one for red wines and one for white ones; for the purpose of the analysis I decided to merge the two data-sets in a single one while adding a 'type' column which differentiates between white wines (0) and red wines (1).

TABLE I
DATA-SET SUMMARY

attributes	**units**	min	max	mean
fixed acidity	$g_{tartaric\_acid}/dm^3$	3.8	15.9	7.2
volatile acidity	g_{acetic_acid}/dm^3	0.08	1.58	0.33
citric acid	g/dm^3	0.0	1.6	0.31
residual sugar	g/dm^3	0.6	65.8	5.44
chlorides	$g_{sodium\_chloride}/dm^3$	0.009	0.611	0.056
free SO2	mg/dm^3	1.0	289.0	30.5
total SO2	$mg/dm,^3$	6.0	440	115.7
density	g/cm^3	0.987	1.038	0.994
pH	abs	2.72	4.01	3.21
sulphates	$g_{potassium\_sulphate}/dm^3$	0.22	2.0	0.531
alcohol	% vol.	8.0	14.9	10.49
quality	scale 1-10	3.0	9.0	5.81
type	categorical	/	/	/

By just looking at the summary it's hard to determine relationships or the importance of certain attributes, to aid in exploring the data-set, the use of plots will be necessary to highlight said characteristics.

The following plots were generating using the python library *seaborn* which provides many tools for visualization.

The first of the target attributes, we can see that *quality* is mostly normally distributed, with mostly middle grade values; also, despite being a range that goes from 1 to 10, no wine goes under 3 or over 9. The plot has different colors depending on the quality of the wine as well. We can see that there is a larger number of white wines compared to the red ones, which becomes evident in the second plot of figure 1; there are in fact **1599** red wines and **4898** white ones, making the data-set unbalanced for the binary classification task, while it won't have much influence in the prediction of quality.

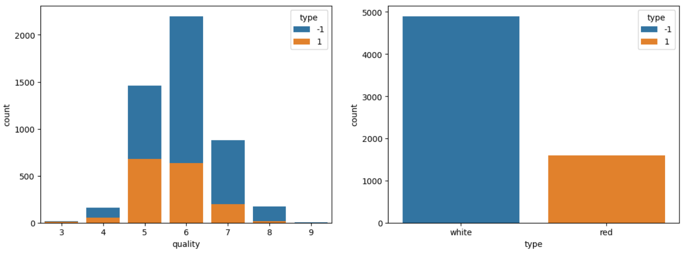

Fig. 1. count-plots for the quality (on the left) and type (on the right) attributes

One powerful visualization tool that *seaborn* offers is the so called *pair plot*: this graph plots all available pairwise relationships in the specified data-set, while the diagonal plots are uni-variate distribution plots drawn to show the marginal distribution of the data in each column.

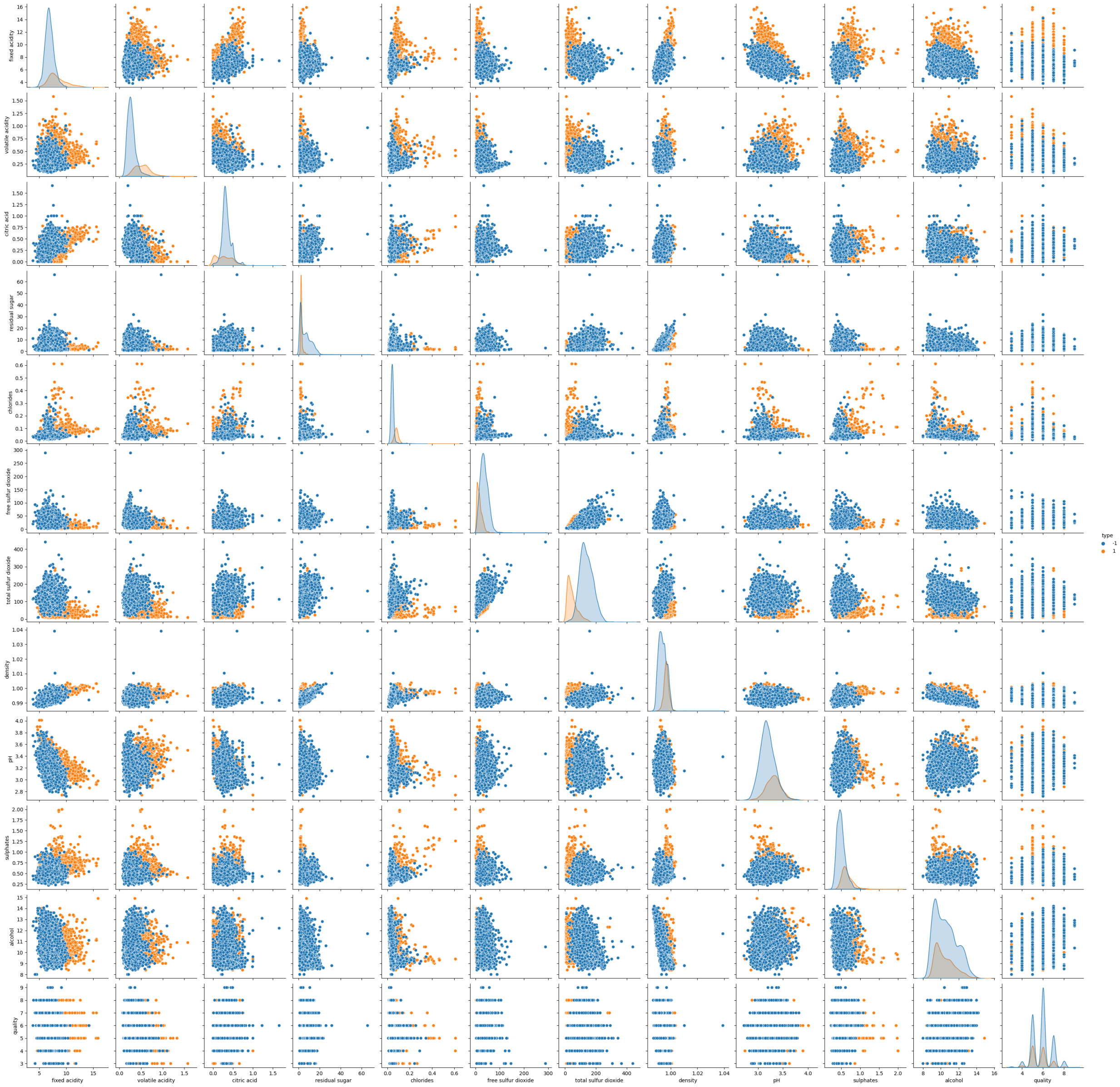

Fig. 2. Pairplot for all attributes

Considering the pairwise relationships not much can be inferred from the plots, no particular distribution comes to the eye, while looking at the uni-variate distribution plots we can say that all the attributes are roughly normally distributed.

B. Correlation Analysis

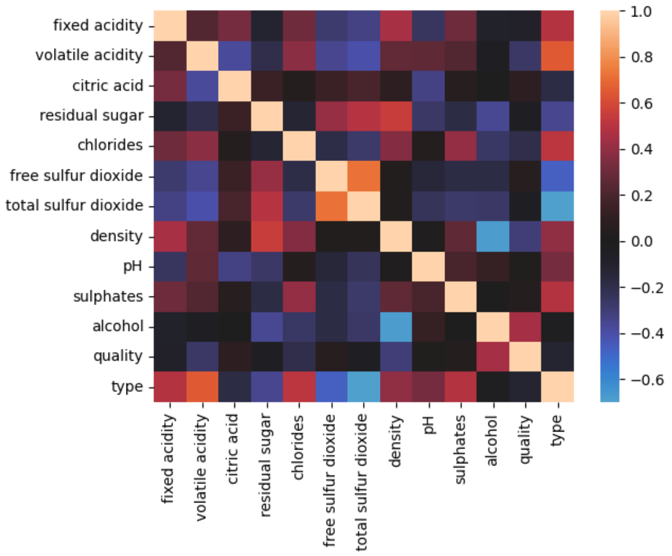

Fig. 3. correlation heat-map

The best way to quickly infer the correlation between the target attribute and the other ones is without doubts a heat-map whose values are given by a correlation matrix. Figure 3 is the output of this process, and from it we can deduce that:

- Type is strongly correlated with volatile and fixed acidity, total and free SO2, and residual sugar
- Quality has no strong correlations, but is influenced by alcohol and density the most
- Type and Quality (the target attributes), have little to no correlation

How strongly or weakly an attribute is correlated with one another is a double-edged sword in machine learning; if an attribute has too weak of a correlation it can be safely cut from the prediction variables, at the same time, a too strong correlation could lead to false predictions or to a ’monopoly’ by the model as it would be seen as the only meaningful one. Considering the heat-map, I can infer that the binary classification between red and white wine will be a task with a high level of success, while the prediction of quality will achieve a less optimal result.

C. Target attribute focus

To further show the correlation between the target attribute and the rest, I decided to use box-plots of the target attribute vs the rest:

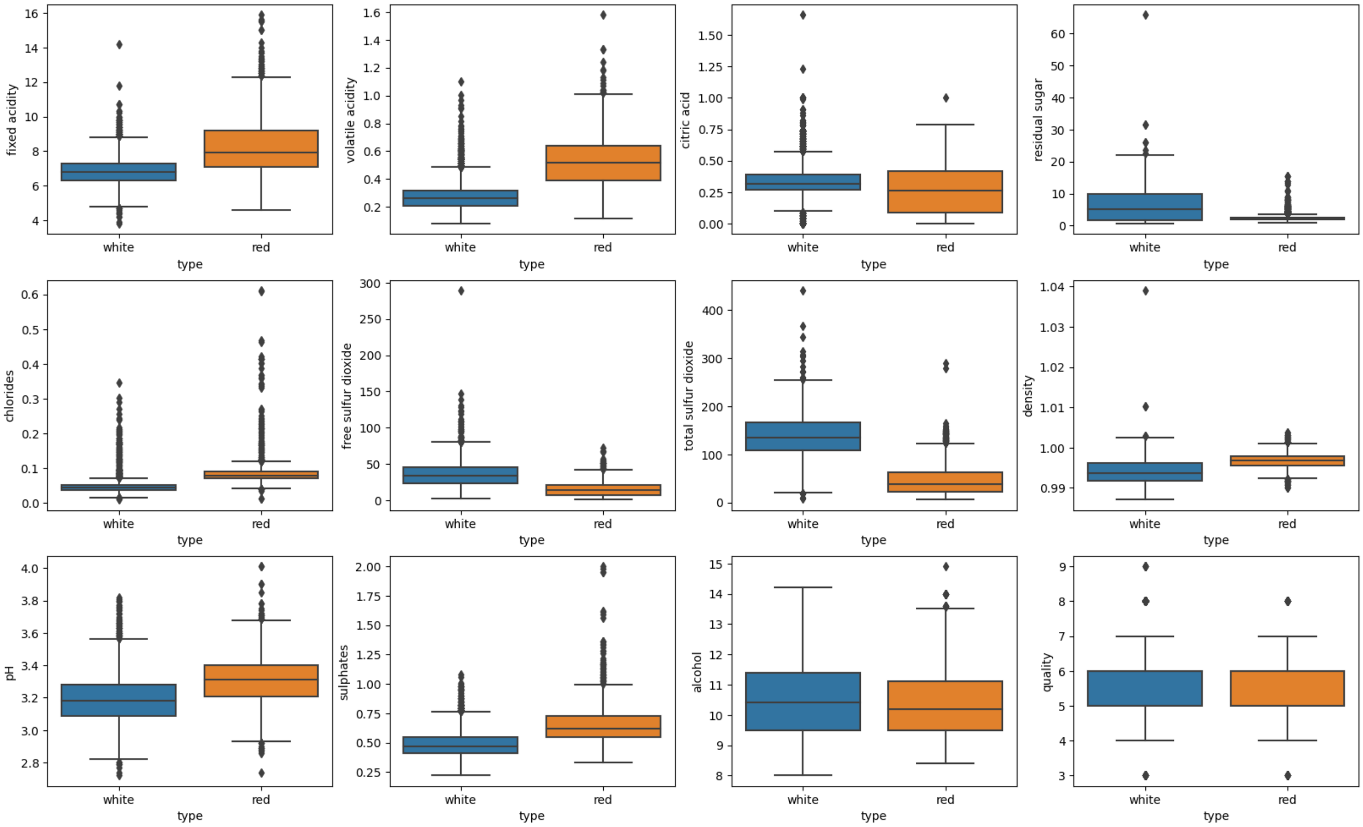

Fig. 4. type box-plots

By looking at the box-plots in figure 4 we get a confirmation of what we found in figure 3, where the distance between boxes is larger the type is more influenced by the attribute we are considering, confirming that acidity and SO2 are the most relevant in this classification.

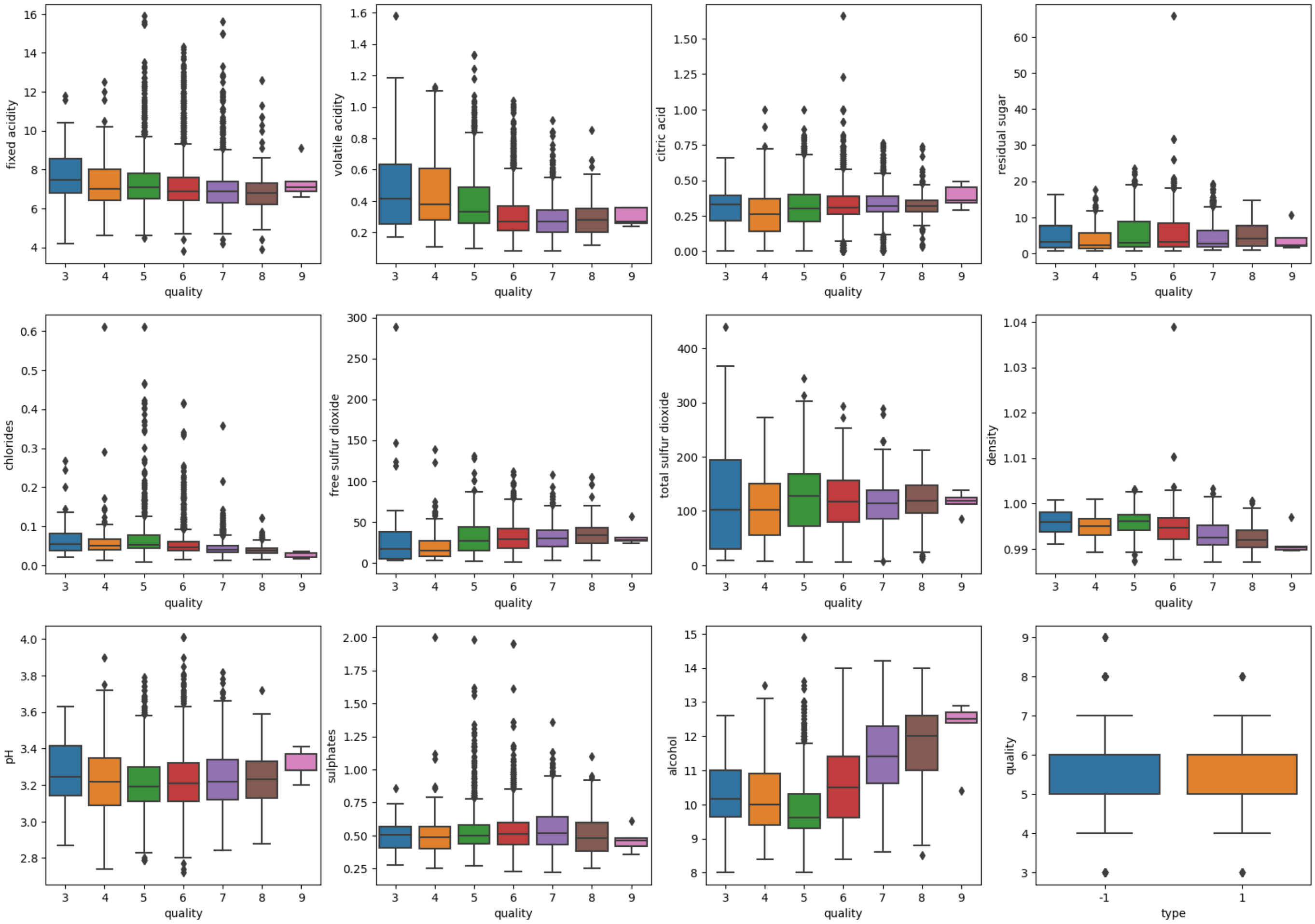

Fig. 5. quality box-plots (the last plot is the same as before for visualization reasons)

Looking at figure 5 we can notice some outlier data for certain attributes, but the most evident aspect is how the type of wine we're considering does not help in determining its quality.

The main takeaways of this preliminary analysis are:

- Quality can be removed as input attribute in the prediction on type and vice-versa
- The data-set is unbalanced toward white wines, this can affect the outcome of the type classification
- While type is correlated more or less to every attribute, quality is not; the dimensionality of the data-set could be easily reduced

III. Preprocessing

This section will focus on giving a mathematical definition of the implemented preprocessing techniques as well as their effect on the data when applied.

A. Feature Scaling

Feature scaling is one of the most important preprocessing steps when dealing with machine learning. Machine learning models profit greatly by having the values they are fed scaled (which usually means standardized); the advantage of training a model with scaled data is that the resulting model doesn't give more weight to larger values than it does to other, and vice versa for smaller values. Two scaling techniques were considered, *min-max normalization* and *standardization*.

1) Min-max Normalization:

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)} \tag{1}$$

Min-max normalization re-scales the values of a feature it is applied to to have distribution value between 0 and 1.

2) *Standardization:*

$$X_{new} = \frac{X_i - \mu}{\sigma} \tag{2}$$

Standardization re-scales the values of a feature to have a distribution with mean 0 (or close to) and variance 1.

B. *Outlier Detection*

An outlier is an object (or a value in our case) that deviates significantly from the rest of the objects; the cause for which outliers exist is usually measurement or execution error. Most data mining methods discard outliers noise and exception to create a sturdier model, this is not true only when the outlier is the main focus of the analysis such as in fraud detection. Outlier detection can be carried through statistical methods -such as **z-score detection**, and *inter quartile range*- as well as using machine learning approaches -such as *isolation forest classification*, *elliptic envelope*, and *K means clustering*-.

1) *Z-Scores:*

The Z-Score of an observation can quantify its *unusualness* when our data follows the normal distribution. Z-Scores are the number of standard deviations above and below the mean that each value falls; the Z score of a value is computed as $Z = \frac{X-\mu}{\sigma}$. The further away an observation Z-Score is from 0, the more unusual it is; standard cutoff values for finding outliers are -3/+3:

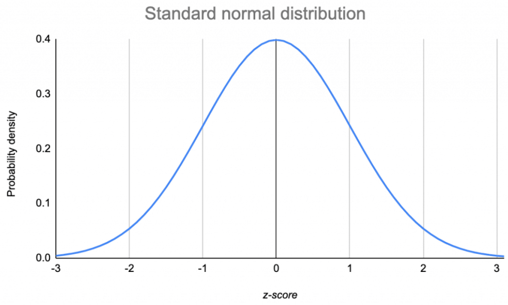

Fig. 6. In a population that follows the normal distribution, Z-Score values more extreme than +/-3 are 1/370 observations

2) *Isolation Forest:*

The Isolation Forest algorithm is an ensemble outlier detection method; the main difference from other outlier detection approaches is that Isolation Forest 'begins' detecting anomalies without learning the normal pattern first.

At the basis of the algorithm there is the tendency of anomalous instances in a data-set to be easier to separate from the rest of the sample (isolate), compared to normal points. To perform the isolation, the algorithm recursively generates partitions on the sample by randomly selecting a split value for the attribute between the minimum and maximum values allowed for that attribute.

From a mathematical point of view, recursive partitioning can be represented by a tree graph known as *Isolation Tree*, where the number of partitions necessary to isolate a point are seen as the length of the path to reach a terminating node starting from the root (the longer the path the less likely a point is isolated).

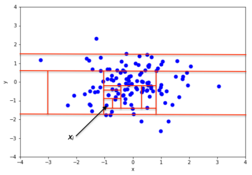

Fig. 7. example of a non anomalous point isolation

C. *Principal Component Analysis*

Principal Component Analysis, in short PCA, is the process of computing the principal components and using them to perform a change of basis on the data, usually using the first few principal components and ignoring the rest. The main advantage of using PCA lies in the ability to reduce the dimensionality of the data-set while preserving as much variability (i.e. statistical information) as possible.

def: The principal components that are computed in PCA are the a sequence of p unit vectors where the $i - Th$ vector is the direction of a line that best fits the data while being orthogonal to the first $i - 1$ vectors.

PCA is commonly used when many of the variables are highly correlated between them, and it is desirable to reduce their number to an independent set.

D. *One Hot Encoding*

One Hot Encoding is a technique usually used in machine learning to work with categorical data. As ML models need numerical data to learn and predict, one possible solution is to split the categorical attribute into n columns -where n is the possible number of values the categorical attribute can have- and then assign to each column the value 1 if the categorical attribute had that value and 0 if it didn't. More modern approaches use a sparse representation of the split to prevent one of the main issues that one hot encoding has, which is adding too many columns for a categorical attribute that has many values.

The table below depicts a visual representation of one hot encoding being applied to the categorical attribute 'fruit'.

Fruit		Orange	Apple	Kiwi
Orange		1	0	0
Apple	→	0	1	0
Kiwi		0	0	1

E. Re-sampling

An unbalanced data-set is characterized by an uneven distribution of observation for the target class. Put simply, the main issue with having unbalanced data is how accurately we manage to learn to predict the target class as our classifier will be biased towards the more frequent class. One of the many possible solutions to this issue is to *re-sample* the data -either by over-sampling or under-sampling it- making the data-set balanced again.

1) Synthetic Minority Over Sampling:

Synthetic Minority Over Sampling, SMOTE in short, works by oversampling the minority class and has been thoroughly described in [4]. The algorithm works as follows:

Algorithm 1: SMOTE

```
Select a minority class instance at random
Find the instance $K$ nearest neighbors
Choose one of the $k$ nearest neighbors
Connect the two points in the feature space
Generate a synthetic instance as a convex combination of the two chosen instances
```

The result is that plausible samples are created, that is, they are relatively close in the feature space to the existing data.

2) Random Under-sampling:

Under-sampling does not require as much work as oversampling does; one common approach is to randomly choose sample to remove from the majority classes until a more balanced data-set is obtained.

F. Pre-processing Pipelines

The final goal of preprocessing is to prepare the data-set for training. The different techniques I described in this section can be combined in a preprocessing pipeline to take advantage of all the benefits of them. To get an idea of how the different approaches influence the model performance, many different pipelines were tested for each model/task, which are, other than the single method being tested alone:

	classification		**regression**
pipeline1	isolation forest min-max normalization	*pipeline1*	isolation forest min-max normalization
pipeline2	isolation forest PCA	*pipeline2*	isolation forest PCA
pipeline3	SMOTE min-max normalization	*pipeline3*	min-max normalization PCA

As the data-set was unbalanced only for the classification task, re-sampling techniques were not considered when dealing with regression.

IV. Model training

In this section I'll go over the models that I applied applied for the different tasks.

A. Random Forest Classifier

Random Forests are a class of ensemble models that operate by constructing a multitude of decision trees at training time. The output of the model is different for classification and regression, and in the first case it is the class most elected by the trees. Compared to a standard decision tree, random forests reduce the danger of over-fitting on the training data.

To explain what happens in a random forest classifier, first we have to describe what a decision tree is. A decision tree is a predictor, $h : X \rightarrow Y$ that predicts the label associated with an instance x by traveling from a root node of a tree to a leaf.

A tree is built by splitting the source set, constituting the root node of the tree, into subsets, which constitute the successor children's. To navigate the tree, each node represent an independent classification task on one of the attributes of the input, with each node having its own splitting rules.

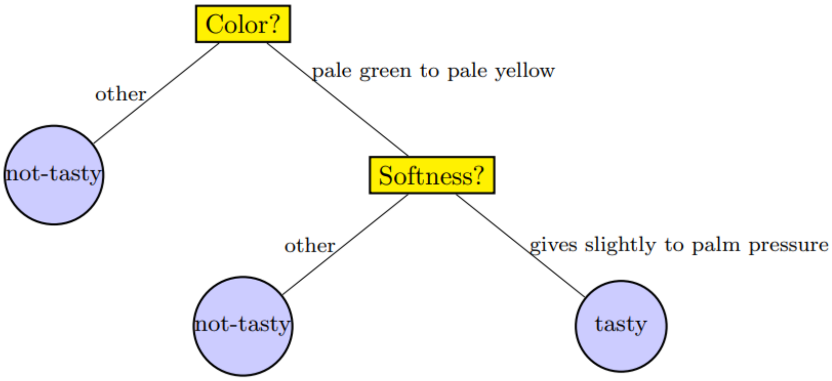

Fig. 8. An example of a simple decision tree, the decision is made by moving from node to node, choosing the path that fits the input correctly

An example of how to build a decision tree is **TDIDT**, which stands for Top Down Induction Decision Tree; the algorithm is composed of a growing phase, in which the decision tree is built, and a pruning phase, in which the complexity of the decision tree is reduced by removing the sub-trees and leafs that increase the overall error estimate of the tree.

Algorithm 2: Grow-DT (*examples*)

```
$N \leftarrow$ a new node
$N.class \leftarrow$ most common class in examples
$N.test \leftarrow$ best attribute (or test)
if N.test is not good enough
    then mark N as a leaf and return N
for each value $v_j$ of N.test
    $examples_j \leftarrow examples$ with $N.test = v_j$
    if $examples_j$ is empty
        then $N.branch_j \leftarrow N.class$
        else $N.branch_j \leftarrow$ Grow-DT(examples)
return N
```

Which in short means we:

1) Find the best attribute
2) Partition the example based on the attribute values
3) Apply the method for each partition

Generally, smaller trees are preferred over larger ones as, despite increasing the error, they reduce over-fitting and make the output trees more generalizable, making *pruning* -the act of actively removing branches and leafs that increase the estimated error rate of the tree- an important step in growing trees.

One other way of reducing the bias, rather than by growing smaller trees, is by generating an ensemble of trees and outputting the majority vote of the forest as prediction. Each tree is trained on a random sample taken from the original data-set, thus creating a Random Forest, this approach is commonly known as *bagging* or *bootstrap aggregating*:

Algorithm 3: Random Forest

```
for $b = 1, ..., B$
    1. Sample with replacement $n$ examples from $X, Y$, call these samples $X_b, Y_b$
    2. Grow a tree on $X_b, Y_b$
```

The subsets can be built not only on data but also on on attributes. This second approach is preferred as, by restricting the available attributes to a subset of them, we grow by design smaller trees that reduce even more the risk of over-fitting.

B. K-NN Classifier

the K Nearest Neighbour algorithm is a non-parametric supervised learning method that can be used for both classification and regression. In both cases, the input consists of the k closest training examples in a data-set, while the output, in the specific case of classification, is a class membership that is decided by plurality vote -the assigned class is the most common among the k nearest neighbour-.

```
Algorithm 4: k-NN
input: a training sample $S = (x_1, y_1), ..., (x_m, y_m)$
output: for every point $x \in X$,
    return the majority label among $\{y_{\pi_i(x)} : i \leq k\}$
```

k-NN classifiers are especially influenced by the so called *curse of dimensionality*: as the number of dimensions increase, the size of the data-set should increase exponentially and its sparsity becomes too high to find patterns or make sensible decisions. Techniques such as the aforementioned *PCA* or *feature selection* help greatly in avoiding this issue, and are commonly applied when using k-NN with data-sets that have ≥ 10 features.

C. Linear Regression

Linear Regression is a common statistical tool used to model the relationship between a *scalar response* and one or more *explanatory variables*, since in our case we have a multitude o explanatory variables, the regression is specifically a *multiple linear regression*.

In linear regression, the relationship between input and target are modeled using *linear predictor functions* (3)

$$\mathcal{H}_{reg} = L_d = \{x \to \langle w, x\rangle + b : w \in \mathbb{R}^d, b \in \mathbb{R}\} \tag{3}$$

The unknown parameters of this set of functions are estimated from the data during training, to do so, a loss function is defined to estimate how far are our linear prediction parameters from the truth, common loss functions are metrics such as the *squared loss function*, of which its empirical risk function is the *mean squared error* (4)

$$\ell(h, (x, y)) = (h(x) - y)^2 \to L_S(h) = \frac{1}{m}\sum_{i=1}^{m}(h(x_i) - y_i)^2 \tag{4}$$

Linear regressors used for prediction are trained -fitted- to an observed data-set of values of the response and its explanatory variables; after such a model is developed, if additional explanatory variables are collected without an accompanying response, the fitted model can be used to make a prediction of such response.

Least Squares (5) is the algorithm that finds the optimal parameters -solves the hypothesis of the linear regression predictors-. Given a training set S and using the homogeneous version of L_d:

$$\underset{w}{argmin}\, L_S(h_W) = \underset{w}{argmin}\, \frac{1}{m}\sum_{i=1}^{m}(\langle w, x_i\rangle - y_i)^2 \tag{5}$$

D. Support Vector Machine Regressor

To describe what a Support Vector Machine Regressor does we first have to introduce a couple of concepts. First, we define a training set of examples to be linearly separable if there exist a *halfspace* (w, b) such that $y_i = sign(\langle w, x_i\rangle + b)\,\forall\, i$.

The goal of an SVM is to pick the best halfspace, which is the one that maximises the *margin* -the minimum distance between a point in a training set and the hyperplane- with respect to the existing data points; this is usually referred to as a *hard-SVM* (5).

```
Algorithm 5: Hard-SVM
input: $(x_1, y_1), ..., (x_m, y_m)$
solve
        $(w_0, b_0) = \underset{(w,b)}{argmin}||w||^2 \; s.t. \forall i, y_i(\langle w, x_i\rangle + b) \geq 1$
output: $\hat{w} = \frac{w_0}{||w_0||}, \hat{b} = \frac{b_0}{||w_0||}$
```

The assumption that the training set considered is linearly separable is a strong one: *soft-SVMs* (6) are introduced as a way to make support vector machines work in non-linearly separable spaces by introducing non-negative slack variables ξ that allow for the constraint $y_i(\langle w, x_i\rangle + b) \geq 1$ to be violated, changing it to $y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i$. The slack variable ξ_i becomes then a measure of how much the hard constraint is being violated; a second new parameter λ controls the trade-off between respecting and violating the constraint.

Algorithm 6: soft-SVM

```
input: $(x_1, y_1), ..., (x_m, y_m)$
parameter: $\lambda > 0$
solve
    $\min_{w,b,\xi}(\lambda||w||^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i) \qquad s.t.\ \forall i,\ y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i\ and\ \xi_i \geq 0$
output: $w, b$
```

Support vector machines were first introduced as a mean to solve a classification task, as the description given so far shows, but regression is also possible with this approach. First introduced in [5], *Support Vector Regressors*, or in this specific case *Epsilon-Support Vector Regressors*, allow us to fit a plane to our training data with the goal of finding a function $f(x)$ that deviates from y_n by a value no greater than ϵ for each training point x, and at the same time is as flat as possible. As seen before when introducing soft-SVM, to assume that we can fit all the available data within a margin of size ϵ is not feasible; as such, by relaxing the constraint and allowing for data to be outside the margin -e.g. by using slack variables ξ- we make the learning problem easier and more able to fit a model closer to the desired output.

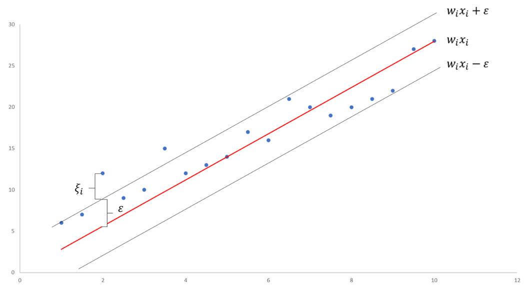

Fig. 9. Example of SVR with Slack Variables

Algorithm 7: Epsilon-Support Vector Regression

```
objective: $min\ \frac{1}{2}||w||^2 + c\sum_{i=1}^{n}|\xi_i|$
s.t. $y_i - w_i x_i| \leq \epsilon + |\xi_i|$
```

The hyper-parameter c found in the objective function allows us to tune the error tolerance, the greater is c, the higher is the tolerance for points to be outside the margin.

1) The Kernel Trick: The Kernel Trick is an important tool available when training a support vector machine: sometimes, data can be more easily separable if 'seen' from a higher dimension then the one we found it in -such as translating 2D values into a 3D space-. By deciding a *kernel function* to apply to our data points and by applying it to our data points we manage to operate in a higher dimensional space without having to compute the coordinates in the new space, making the complexity go from $O(n)$ rather than $O(n^2)$.

We can define a generic kernel function that maps the data in the input space X to the space V $\phi : X \to V$ as:

$$k(x, x') = \langle\phi(x), \phi(x')\rangle v \tag{6}$$

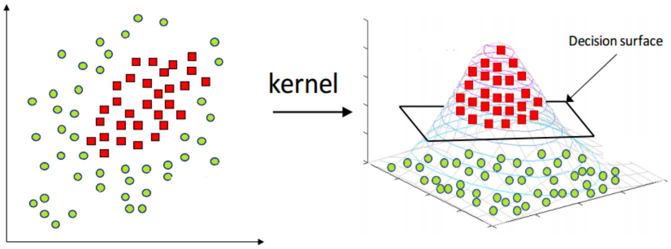

Fig. 10. Visual representation of data being translated to a higher dimensional space

The SVR I applied uses a *Radial Basis Function* (7) kernel which are the most widely used form of kernels in machine learning due to their similarity to the Gaussian distribution; the kernel has 2 parts:

1) $||x - x'||^2$ which is the euclidean distance between the two points x and x'
2) σ which is a variance hyper-parameter we learn during training

$$k(x, x') = exp(-\frac{||x - x'||^2}{2\sigma^2}) \tag{7}$$

V. Results

A. Evaluation Procedure

In order to test the performance of the obtained model, the usual procedure is to split the data-set -with a varying percentage, which in my case was 33%- randomly into *train* and *test* data. The purpose of the test data is to keep it ordered and with its original label to then feed the label-less data into the trained model and compare the predicted outcome with the expected one. If the model were to be tested on the same data it was trained on, the probability of over-fitting would rise.

Another meaningful parameter when using packages such as *sklearn* is the random seed. The model training and the aforementioned train-test-split phases have a certain degree of random sampling, the seed for all experiments was set at the beginning to a fixed number -42- to 'level the playing field'.

B. Evaluation Metrics

To compare the effect that different preprocessing steps and pipelines, as well as the use of different models, had in the classification and regression task, evaluation metrics are used. There exist many different kinds of evaluation metrics, and in the experiment I used the following:

1) Accuracy: The accuracy of a model summarizes the performance of a classification model as the number of correct predictions divided by the total number of predictions. It is a very basic metric that gives an immediate overlook of the model performance. Given N predictions, the accuracy of a model is given by:

$$accuracy = \frac{\sum correct}{N} \tag{8}$$

This metric is well suited for a classification task, as it is not very meaningful when predicting a continuous value.

2) Confusion Matrix: The confusion matrix of a set of predictions is a special type of contingency matrix with two dimensions -actual and predicted-, and identical et of classes in both dimensions. In the case of binary classification, where the output is either positive or negative, the matrix has four cells that cover the four possible cases:

- True Positive and True Negative are the correct outcomes, a 1 is predicted as a 1 and so on
- The False Negative cell is the number of type 2 errors committed during prediction
- The False Positive cell is the number of type 1 errors committed during prediction

With these four values -TP, TN, FP, FN- its possible to compute a large number of metrics that emphasize different aspects of the results, the two we are most interested in are *sensitivity* (9), the rate of correct positives among all positives, and *specificity* (10), the rate of correct negatives among all negatives. These two give an idea of how well the model is predicting each kind of wine.

		Predicted Condition	
	Total Population P+N	Positive (PP)	Negative (PN)
Actual Condition	Positive (P)	True Positive (TP)	False Negative (FN)
	Negative (N)	False Positive (FP)	True Negative (TN)

$$sensitivity\,(TPR) = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{9}$$

$$specificity\,(TNR) = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{10}$$

3) MSE: When dealing with continuous values and regression, rather than being interested in the number of correct predictions we are more interested in how close we were to the correct result. The *Mean Squared Error* measures the average of the squares of the errors and it is one of the most used metrics as it is an estimator of the true average loss on the actual population distribution. The lower the average error, the lower the MSE.

If a vector of n predictions is generated from a sample of n data points on all variables, and Y is the vector of observed values of the variable being predicted, with $\hat{Y}$ being the predicted values, then the within-sample MSE of the predictor is computed as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{11}$$

4) R^2 Score: Also called the *coefficient of determination*, R-squared is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted. R^2 is a comparison between the residual sum of squares with the total sum of squares.

given:
a data-set with n values marked $y_1, ..., y_n$ -or $[y_1, ..., y_n]^T$ each associated with a fitted value $f_1, ..., f_n$
defined the **residuals** as $e_i = y_i - f_i$

if:
$\hat{y}$ is the mean of the observed data $\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

then:
the *sum of squared residuals* is defined as $SS_{res} = \sum_i e_i^2$
the *total sum of squares* is $SS_{tot} = \sum_i (y_i - \hat{y})^2$
and $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$

C. Type Classification

TABLE II
SUMMARIZATION OF THE RESULTS OBTAINED BY TESTING THE OBTAINED MODELS ON 30% OF THE ORIGINAL DATASET

modelname	**input_db**	**accuracy**	**sensitivity**	**specificity**
KNeighborsClassifier()	**pipeline_1**	**0.997**	0.991	**0.998**
	pipeline_3	0.991	**0.995**	0.987
	standard_norm	0.99	0.978	0.995
	minmax_norm	0.989	0.978	0.993
	benchmark	0.931	0.802	0.976
	pca	0.931	0.802	0.976
	isoforest_out	0.93	0.778	0.974
	smote	0.928	0.953	0.902
	zscore_out	0.926	0.774	0.97
	random_undersample	0.891	0.87	0.911
	pipeline_2	0.878	0.602	0.966
RandomForestClassifier()	**isoforest_out**	**0.996**	0.983	**0.999**
	pipeline_3	0.995	**0.994**	0.995
	pipeline_1	0.994	0.983	0.998
	zscore_out	0.993	0.978	0.998
	smote	0.993	0.992	0.994
	standard_norm	0.992	0.978	0.997
	benchmark	0.992	0.976	0.997
	minmax_norm	0.992	0.976	0.997
	random_undersample	0.99	0.989	0.991
	pca	0.986	0.962	0.995
	pipeline_2	0.938	0.749	0.998

Table V-C shows a summary of the results obtained by training and testing the models described in IV-A and IV-B on the pre-processed data-sets specified in III-F along with a benchmark that had no preprocessing applied.

Starting with the k-NN results, we can see that the most effective preprocessing step was *feature scaling*: all the models trained on a data-set that was previously scaled achieved an accuracy of ≈ 0.99, with no particular difference between the classification of red and white wines -as we can see from *specificity* and *sensitivity*-. Another interesting result is the equivalence in performance we can see between *benchmark* and *PCA*: as the number of features wasn't that high, PCA didn't change the output model performance, which is in line with the concept of PCA. Lastly, we can see the positive effect that *SMOTE* and *Random Under-Sampling* had in balancing the prediction preference, with sensitivity and specificity scores close to each other -we can see benchmark predicting a white wine way more often due to the unbalanced nature of the data-set, leading to many false positives despite a achieving higher accuracy-.

Regarding Random Forest we can see that feature scaling had much less of an effect on the results with pretty much identical results between feature scaled models and the benchmark one. What greatly improved performance was outlier removal, with *isoforest*, *z-score* and *pipeline 1* in the top 4 scores. PCA negatively impacted the performance of the model, highlighting the affinity that Random Forest has for data-sets with a large number of attributes. Lastly, re-sampling techniques had little to no effect on the performance, suggesting that Random Forest has a high degree of tolerance for unbalanced datasets.

Overall the results were satisfactory, with high scores even for non pre-processed models, suggesting that with the collected data the difference between red and white wines is easily identifiable.

D. Quality Regression

TABLE III
Summarization of the results obtained by testing the obtained models on 30% of the original dataset

name	pipeline	r2_score	mse
LINEAR	isoforest_out	**0.291**	0.546
	zscore_out	0.29	**0.514**
	minmax_norm	0.28	0.54
	standard_norm	0.28	0.54
	benchmark	0.28	0.54
	pipeline_1	0.276	0.525
	pca	0.275	0.544
	pipeline_3	0.275	0.544
	pipeline_2	0.011	0.756
SVM_RBF	standard_norm	**0.377**	**0.467**
	minmax_norm	0.353	0.486
	pipeline_3	0.351	0.487
	pipeline_1	0.351	0.47
	pca	0.218	0.587
	benchmark	0.144	0.643
	zscore_out	0.138	0.623
	isoforest_out	0.132	0.669
	pipeline_2	-0.003	0.767

Table V-D shows a summary of the results obtained by training and testing the models described in 3 and 7 on the pre-processed data-sets specified in III-F along with a benchmark that had no preprocessing applied; re-sampling was left out as the data-set was not unbalanced for this task.

Starting with the Linear Regressor, we can see that the best performance was obtained when *outlier removal* was applied to the data-set before training: linear regression is highly influenced by outliers as they tend to negatively impact the found function by deviating too much from the rest of the data. Feature Scaling had no effect to the output model because of the applied estimation technique -ordinary least squares-. Lastly, PCA not worsening the results too much suggests the importance that the method can have in real-time inference: by reducing the dimensionality of the data it is possible to compute the prediction faster -and the bottleneck usually lies in the model not in the PCA process- without loosing too much performance.

Regarding Support Vector Regression with a Gaussian kernel, we can see the best results in the feature-scaled models, the hyperplanes found by a feature scaled Support Vector Machine are less sensitive to specific attributes, making the output model more robust to changes in data as well as faster at finding the hyperplane itself -training is much faster in a feature scaled SVM-. We can see the importance of feature scaling by looking at the other results as well: the MSE increases by over 0.1 and the R^2 lowers by the same amount when feature scaling is not applied, suggesting that it is a mandatory step when training a Support Vector Machine.

Overall, considering the top 4 results for both methods, we cans see SVR outperforming Linear Regression as expected, SVR is a much more complex and robust algorithm that is able to create models which are fit to the data they are trained with, while the output of Linear Regression is a much more rudimental one. Looking at the MSE, the task of predicting the quality of the wine could be considered complete as the results were very close to the expected outcome, confirming the idea that such a model could be created proposed in [1].

VI. Conclusion

In this work I went over some of the many available pre-processing and data-mining techniques while describing their importance and applications. The models obtained showed great promise, especially for the much more difficult task of predicting the quality of a given wine from its physio-chemical attributes: as [1] suggests, such a model can be very useful during production as it's easier to have a computer output an expected quality rather than have an expert give a non necessarily objective score to the wine, both cost and time wise.

References

[1] Paulo Cortez et. al.,*Modeling wine preferences by data mining from physicochemical properties*, Elsevier, 2009
[2] Shalev-Shwartz, Shai, and Shai Ben-David, *Understanding machine learning: From theory to algorithms*, Cambridge university press, 2014
[3] Li Canchen, *Preprocessing Methods and Pipelines of Data Mining: An Overview*, Seminar Data Mining, 2019
[4] N. V. Chawla, et. al., *SMOTE: Synthetic Minority Over-sampling Technique*, Journal Of Artificial Intelligence, 2011
[5] Vapnik, V., *The Nature of Statistical Learning Theory*, Springer, New York, 1995.
[6] Suykens, Johan A. K.; Vandewalle, Joos P. L.; *Least squares support vector machine classifiers*, Neural Processing Letters, Jun. 1999