Predicting wine quality

Project 3 for FYS-STK4155

Bjørn Inge Vik
Jolynde Vis

December 15, 2019

Abstract

The popularity of wine is ever increasing. Certification is an important part of the wine business and deals among others with the wine quality. Using machine learning algorithms to analyze the patterns between the quality of a wine and its physicochemical properties, and predicting a quality based on these properties, offers interesting options for the wine sector. For our analysis we used the UCI 'Wine Quality Dataset' with physicochemical properties of red and white vinho verde from the north of Portugal, which is the 9th largest wine exporting country in the world. We have applied six different machine learning algorithms to the data, and tried both classification and regression approaches. From our analysis we can conclude that the best model that fits this data is a Random Forest model, for both the classification and regression case. The model was able to achieve a classification accuracy of around 67% for both datasets. A model simply predicting the most frequent class would be roughly 43% and 45% accurate on the red and wine wine data sets, respectively. The results are promising for the wine sector and show that there are interesting possibilities for classifying wine quality using machine learning algorithms.

Contents

1 Introduction

Wine went from having a luxury status to being a treat for everyone in society, not only the upper class. Production and consumption is ever increasing, reaching a new record level in 2018 [1]. Wine is both consumed and produced all over the world. Portugal was the ninth largest export country in 2018, with a market share of 3% [2].

In addition to determining wine quality, certification and quality assessment play an important role in preventing adulteration and potential health hazards. Wine certification is based on physicochemical and sensory tests [3], where the physicochemical tests determine the chemical properties of the wines, while the sensory tests rely on human experts and are concerned with taste, i.e. human senses.

To be able to stay in the top 10 of wine exporting countries, it is important to produce high quality wines efficiently. It is of great importance to know what the chemical properties of a high quality wine are. Artificial intelligence (AI) is becoming widely popular around the world and adds intelligence to products, as well as enables us to analyse much larger datasets and get the most out of our data [4]. As a subfield of AI, machine learning algorithms such as neural networks and random forests enable us to find patterns in large datasets. Applying such machine learning algorithms to recognize patterns between the physicochemical properties of wine and its quality is of enormous interest to the wine sector. Firstly, since taste is subjective and one of the least understood human senses [5], it introduces a way to potentially make the quality assessment more objective. Also, it could save time and money as less wine tastings/tasters are needed to assess the quality of a wine. Because the prices of wines are mainly based on the quality of the wine, it could help the wine sector to put a more objective price on the wine. Lastly, many of the physicochemical properties of wines can be controlled in the production process. If the wine producers know which property, or combination of properties, ensures a higher quality of the wine, tweaking parts of the production process could improve the overall quality and increase exports and profit, especially as high quality wines can be priced higher. However, the relationships between the physicochemical and sensory analysis are complex, which makes wine classification a hard task [6].

In this report we aim to discover patterns between the physicochemical properties of a wine and its quality as determined by human tasters. For this analysis we use the 'Wine Quality Dataset' from UCI [7], a dataset concerned with wine samples from Portugal. The report is structured as follows; first we explain the theory and methods used and we shortly discuss the code implementation and some assumptions that were made. Before the analysis we first have a look at the dataset to see if any preprocessing is necessary. We then perform the analysis by implementing six different machine learning algorithms to find the best one to fit this dataset. Lastly we conclude our findings and state the limitations and

future research options in this area.

2 Methods

This section explains the theory and methods for the algorithms used in the analysis. We will discuss both classification and regression methods for relevant models. We leave out the theoretical parts for linear regression, logistic regression and neural networks as they have been discussed in the reports of Project 1 and Project 2.

2.1 Support Vector Machines

The SVM (Support Vector Machine) is a discriminative algorithm that makes use of a hyperplane to separate classes. The points that lie closest to the hyperplane, are the most difficult points to classify, and they have the most influence on the optimum location of the hyperplane [8]. The vectors describing the position of these points are so called 'support vectors'.

2.1.1 Classification

The SVM finds the optimal hyperplane by maximizing the margin around it. Figure 1 shows a two-dimensional classification case, where the black middle line is the hyperplane (the 'separation' line).

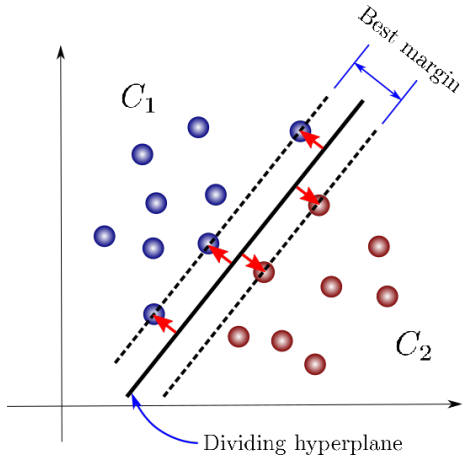


Figure 1: Support Vector Machine. [9]

In this 2-dimensional case the hyperplane becomes a line defined by

$$b + w_1 x_1 + w_2 x_2 = 0, \tag{1}$$

where x_1 and x_2 are the variables, b is the intercept and w_1 and w_2 are the vector elements of a vector that is orthogonal to the separation line. We can rewrite the equation to vector

form:

$$\boldsymbol{x}^T\boldsymbol{w} + b = 0 \tag{2}$$

We want to find the largest possible margin M, which is defined by

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq M||\boldsymbol{w}|| \; \forall i \tag{3}$$

We can scale the equation so that $||\boldsymbol{w}|| = 1/M$, so that we have to find the minimum of the norm, i.e. $\boldsymbol{w}^T\boldsymbol{w} = ||\boldsymbol{w}||$.
Using the Lagrangian Multipliers theory, we can find the coefficients of the hyperplane

$$\boldsymbol{w} = \sum_i \lambda_i y_i \boldsymbol{x}_i \tag{4}$$

and the intercept

$$b = \frac{1}{N_s} \sum_{j \in N_s} \left(y_j - \sum_{i=1}^{n} \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x}_j \right) \tag{5}$$

where N_s is the number of support vectors.

The above case assumes that the classes are well separated and do not have any overlap. In that case, we need a soft classifier, which allows misclassification of some points. To do so, so-called *slack* variables are added, which gives the following new equation

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1 - \xi_i, \tag{6}$$

where $\xi_i \geq 0$.
By bounding the sum $\sum_i \xi_i$ with a value C, we bound the total number of misclassifications.

In the case where the separation line or hyperplane is non-linear, the SVM introduces so-called *kernel (K) transformations* to still obtain a separation that is (almost) linear. In this case the basis is changed to $x \to z = \phi(x)$.
A few common used transformations are: [10]

- Linear: $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T\boldsymbol{y}$,
- Polynomial: $K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y} + \gamma)^d$,
- Gaussian Radial Basis Function: $K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma||\boldsymbol{x} - \boldsymbol{y}||^2\right)$,
- Tanh: $K(\boldsymbol{x}, \boldsymbol{y}) = \tanh\left(\boldsymbol{x}^T\boldsymbol{y} + \gamma\right)$,

2.1.2 Regression

The SVM uses the same principles for regression as for classification. The difference is that regression has unbounded, real numbers as output. To SVM still tries to seek and optimize the generalization bounds given for regression. In this case a loss function is defined that ignores errors which are situated within the certain distance of the true value: the epsilon intensive loss function. Figure 2 shows an example with the epsilon intensive band. The variables measure the cost of the errors on the training points, which are zero for all the points inside the band [11].

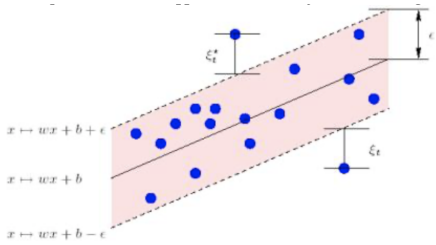


Figure 2: Support Vector Machine Regression. [11]

The goal of the SVM is now to minimize the following function

$$min\frac{1}{2}||w||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \begin{cases} y_i - f(\boldsymbol{x}_i, w) \leq \epsilon + \xi_i^* \\ f(\boldsymbol{x}_i, w) - y_i \leq \epsilon + \xi_i \\ \xi_i, \xi_i^* \geq 0, i = 1, ..., n \end{cases} \tag{7}$$

Kernels are again used to fit the data if the regression is non-linear.

2.1.3 Parameters

The SVM functionality in Scikit-Learn has three main parameters, which relate to the parameters explained above, as specified in table 1.

Parameter	Description	Value
kernels	Transformation of the plane	'linear' 'poly' 'rbf' 'tanh' (and more)
C	Penalty parameter; how much misclassification is allowed	$C > 0$ The higher C, the smaller the margin and the less misclassifications are allowed
γ	Kernel coefficient; sensitivity to differences in feature vectors	$\gamma > 0$ The higher γ, the less points that are far away from the separation line are considered (high sensitivity)

Table 1: SVM parameters in Scikit-Learn

2.2 Decision trees

A decision tree is a tree-like structure that divides data into subsets. The idea is to make rules to divide the data into bins consisting of data points with similar target values. At each so called node in the tree, a decision is made to split the data according to the value of a specific feature. Two branches emerge from each node, each branch representing a choice. Following the nodes and branches in a flow-like way, one ends up in the leafs. Each leaf should contain data points that share the same or similar target variable. The decision tree model will then predict the same class for all data points in the same leaf, or the same value if it is a regression problem.

Decision trees can be constructed in an automatic way. In case of classification, the Classification And Decision Tree (CART) method searches for the best split in each node by minimizing

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}. \tag{8}$$

J is a function of the variable k that the split is performed on and the threshold of the split t_k. m is the total number of data points in the node and m_{left} and m_{right} are the number of data points in the left and right split respectively. The gini factor G is defined by

$$G = 1 - \sum_{i=1}^{C} p_i^2, \tag{9}$$

where p_i is the proportion of instances in class i in one direction of split, i summing over C classes. The gini factor measures impurity of each subset created from a left/right split. One can see from eq. 9 that if a split leads to a branch in which all data is in one class, the gini factor becomes zero. Eq. 8 tells us to split the data in a way so that the weighted gini factors of the two resulting branches are minimized, i.e. maximizing the purity of the subsets. For a CART regression tree, the gini indices in eq. 8 are typically replaced by the variance.

Decision trees are prone to overfitting. To mitigate this, one can for example put an upper limit on the number of splits before reaching the leaves (max depth) or put a lower limit on the number of data points in each leaf.

2.2.1 Random forest

Random forest is an ensemble method that combines the predictions from multiple decision trees (hence the name). The node splits in each tree is typically determined by evaluating only a random subset of the features in the data set, for example equal to the square root of the total number of features. Bootstrapping is also normally performed so that each individual tree is constructed using a different data set. Together, this ensures enough variation between the trees and avoids wasting resources by generating near-identical trees determined by a few dominant features.

A random forest algorithm is inherently more robust against overfitting compared to single decision trees. Still, like for individual trees, one can experiment with limits on depth and number of data points in each leaf. Another key hyper parameter is the number of trees in the forest. It should be large enough to compensate for the tendency by individual trees to overfit, and thus it depends on e.g. the tree depths.

2.2.2 Gradient boosting

Gradient boosting is another ensemble method that combines the predictions of a range of simpler models, called base models. Decision trees of low complexity (e.g. depth = 1) are commonly used as base models, but other models may also be used. The starting point is a simple model predicting a constant value equal to the average of the target variables, or just zeros:

$$F_0 = 0. \tag{10}$$

The base model h (typically a shallow decision tree) is then fitted to the gradient of the loss function evaluated at the position determined by eq. 10. An improved version of the starting model becomes

$$F_1 = F_0 - \alpha h_1(\mathbf{x}, \nabla_{F_0} L(\mathbf{y}, F_0(\mathbf{x})), \tag{11}$$

where α is the learning rate. This process is repeated M times, i.e. using gradient descent to iteratively add corrections to the model. The final model is given by

$$F_M = F_0 - \sum_{m=1}^{M} \alpha h_m(\mathbf{x}, \nabla_{F_{m-1}} L(\mathbf{y}, F_{m-1}(\mathbf{x})). \tag{12}$$

As the number of estimators M increases, overfitting can become a problem. This can be compensated by choosing a smaller learning rate α and reduce the impact of each individual tree. Setting $\alpha < 1$ (referred to as shrinkage) has been shown to give better results [12]. One can also experiment with the simplicity and architecture of the base model.

2.2.3 XGBoost

Extreme gradient boosting (XGBoost) is yet another ensemble learning method. It uses decision trees or linear models as base models. Similar to gradient boosting, it is an iterative model with the prediction $\hat{y}$ at iteration t given by

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{13}$$

where f being individual tree models. Like for gradient boosting, a learning rate or shrinkage parameter η (left out above) can also be used to scale each tree boosting. Whereas standard gradient boosting fits each model to the gradient of the loss function at each iteration, XGBoost aims to fit each base model to the error given by the combination of the previous models. The loss at iteration t

$$\sum_{i=1}^{n} l(y_i, \hat{y}_i^t) \tag{14}$$

can be approximated using a second order Taylor expansion around the loss at iteration $t-1$:

$$\sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + \text{constant}, \tag{15}$$

where

$$\begin{aligned} g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \end{aligned} \tag{16}$$

and Ω is a regularization term punishing the complexity of the tree defined by

$$\Omega(f) = \alpha T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{17}$$

where T is the number of leaves in the tree, w_j is the score (prediction) for leaf j. α and λ are constants controlling the regularization strength. Removing the constant terms from eq. 15 results in the following minimization objective at each step t:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t). \tag{18}$$

The XGBoost algorithm combines regularization, tree pruning and sophisticated and effective ways of generating each tree f_t described in detail by Chen and Guestrin [1]. Besides regularization and limitations on tree depth, one can also make the algorithm more conservative by setting a threshold γ for the improvement in loss required when splitting a node.

2.3 Model evaluation

2.3.1 Classification

For classification the models are evaluated on their accuracy score,

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(\hat{y}_i = y_i)}{n}. \tag{19}$$

In addition we use look at the confusion matrix to evaluate classification results.

2.3.2 Regression

For regression the models are evaluated on their mean squared error

$$\text{MSE}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2, \tag{20}$$

and since we will compare our results with the report by Cortez et al. [6], we also evaluate the models according to their mean absolute deviation (MAD). The MAD score as used in the paper by Cortez et al. [6] is defined as

$$\text{MAD}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} |x_i - \hat{y}_i|, \tag{21}$$

where $\hat{y}_i$ is the predicted value for the i-th 0data point. MSE will penalize gross errors more than the linear MAD. In our case we choose to put a higher penalty on scores further away from the true value, so our main regressopm metric is MSE, while we use MAD to compare our results with the findings of Cortez et al. [6].

Because the outcome variable of the dataset contains only whole numbers, we will use an calculate an accuracy score for the regression as well. This accuracy score is defined as the share of points correctly predicted within a tolerance $T = 0.5$, i.e. rounding the predicted regression score to the nearest integer and calculating the accuracy.

3 Code implementation

In this section we elaborate more on how we implemented the different methods. The python files with the code can be found on

https://github.com/bingovik/FYS-STK4155/tree/master/Project3

3.1 Implementation

We have tested six different models (linear regression (OLS), logistic regression (LR), neural network (NN), support vector machines (SVM), random forest (RF) and XGBoost (XG)) on both the red and white wine data sets. For the neural networks we used Keras/Tensorflow, for the other models we used Scikit-Learn.

For classification, the models are assessed on their accuracy score, and for regression the models are assessed on their MSE and MAD score. We regard the linear and logistic regression models in some ways as baseline models, with the aim and expectation is that the other models, when optimized, will outperform these 'basic' algorithms. The models are trained, optimized and cross-validated on 80% of the data, whereas the other 20% is kept apart for a final test.

The classification goal is to predict the class label of a new data point, as one of the classes of the observed outcome variable: 3, 4, 5, 6, 7, 8 or 9. The aim of the regression is to predict a real number that is as close as possible to the observed target variable.

The code is set up in such a way that when it does a parameter search or grid search it automatically implements the best parameters found. The same code can be run for both datasets by simply switching a wine type variable to 'red'/'white'.

4 Data preprocessing

For this report we used the *'Wine Quality Data Set'* from UCI [7], which contains data of different red and white vinho verde samples from the north of Portugal. The outcome variable is the quality of the wine, on a scale from 0 to 10. According to Cortez, P. et al. [6], the wine preferences come from objective analytical tests that are available at the certification step, i.e. the quality scores of the wines in these datasets are 'objective'. The red wine dataset contains 1599 entries and the white wine dataset 4898 entries, and they both have the same 11 independent variables. The datasets are described in table 2.

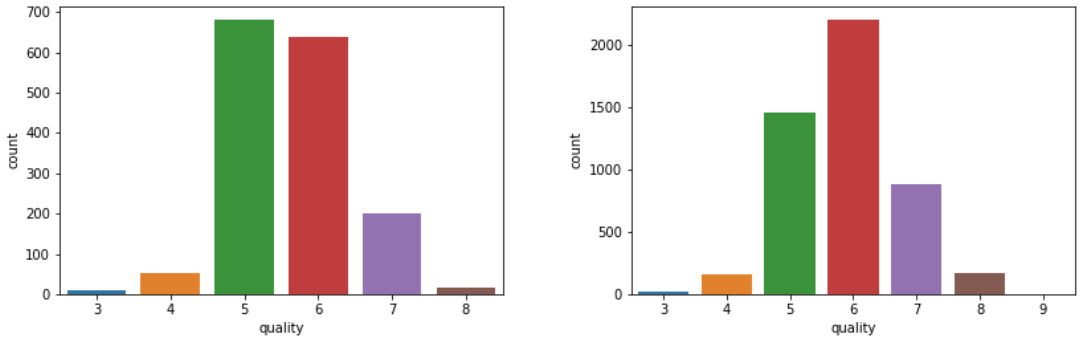
	Red wine			White wine		
Variables	min	max	mean	min	max	mean
fixed acidity	4.60	15.90	8.32	3.80	14.20	6.85
volatile acidity	0.12	1.58	0.53	0.08	1.10	0.28
citric acid	0.00	1.00	0.27	0.00	1.66	0.33
residual sugar	0.90	15.50	2.54	0.60	65.80	6.39
chlorides	0.01	0.61	0.09	0.01	0.35	0.05
free sulfur dioxide	1	72	16	2	289	35
total sulfur dioxide	6	289	46	9	440	138
density	0.99	1.00	1.00	0.99	1.04	0.99
pH	2.74	4.01	3.31	2.73	3.82	3.19
sulphates	0.33	2.00	0.65	0.22	1.08	0.49
alcohol	8.40	14.90	10.42	8.00	14.20	10.51

Table 2: Variables of the wine quality dataset

The outcome variable is distributed as shown in figure 3. We see that there are much more wines with an 'average' quality score between 5-7 than wines with a very high or very low quality. We also observe that in the red wine dataset there are no wines that have a quality score of 1, 2, 9 or 10, and in the white wine dataset there are no wines that have a quality score of 1, 2 or 10.

4.1 Correlations

We show the correlation matrix for both datasets in appendix A. There are few strong correlations between variables. One exception is a significant negative correlation between density and alcohol content, which is not surprising given alcohol's low density of 0.79 kg/liter. Alcohol also has the strongest correlation with the outcome variable 'quality'. Also unsurprising, the wines' pH value and citric acid content are correlated.



(a) Red wine dataset

(b) White wine dataset

Figure 3: Distribution of the outcome variable 'quality'

4.2 Outlier analysis

We analyse the different variables to see whether there are outliers or divergent distributions. The plots are in appendix A. There is no missing data, no outliers and no divergent distributions in the variables. There is no need to clean the data for outliers.

4.3 Feature selection

To see which features are most important we have used the feature importance function of a random forest classifier, based on which variables are split early and often in the trees (see eq. 8). Figure 4 shows the weighting of the different variables. Alcohol is the most important feature, especially for the red wine dataset. However, all the importance scores of the variables are fairly equal. Since we have only 11 variables and they seem to be (roughly) equally important we will use all of them, and not implement any feature selection or dimensionality reduction.

4.4 Feature scaling

We have used Scikit-Learn's *StandardScaler* to scale features so that their distributions have a mean of zero and a standard deviation of one.

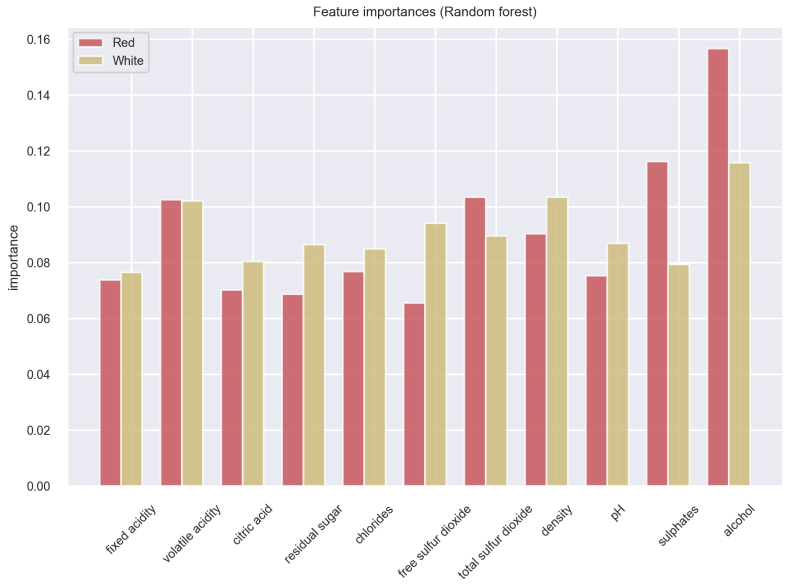


Figure 4: Feature importance of the variables for the red and white wine dataset.

5 Analysis

In this section we first elaborate on how the models are optimized. For more clarity and compactness of the report, the optimization is only shown for the red wine dataset. However, we used the exact same code with the same grid search and parameter optimization techniques for the white wine dataset. For the cases where the white wine dataset gave very interesting or very different results than the red wine dataset, we've included the results in the appendix and refer to them in the text. After the optimization we we show the final scores for the optimized models in table 4, on both the red and white wine dataset. We end this section with an analysis of the scores given in table 4.

5.1 Linear regression

We tested Ridge regression with different regularization λ in the range (0.1, 100). Figure 14 in appendix B shows little impact; linear regression models are likely not complex enough when applied straight forwardly on the 11 predictive features. A minor positive effect with $\lambda \approx 50$ was seen for the white wine set, slightly surprising since this is the larger data set.

5.2 Logistic regression

For the logistic regression we used the default parameters in Scikit-Learn, which include the inverse regularization parameter $C = 1$. As for linear regression, tests with varying regularization provided little benefit and our results presented below represent a model using the default $C = 1$.

5.3 Neural Network

For the neural network there is a fair amount of hyperparameters to tune. We ran a grid search over the number of training epochs, layer architecture and L2 regularization to find the optimal hyperparameters. We fixed the batch size at 32 and used ReLU activation on the hidden layers. The results of the grid search are shown in the heatmap in figure 5.

There doesn't seem to be an obvious structure in the heatmap. One can extract that training the network for 30 epochs may be too little to maximize accuracy. The high scores are a bit scattered. This reflects the fact that different combinations of hyperparameters are important, but it is also manifestations of noise caused by training. Figure 6 shows the learning curves for loss and accuracy for the best model (hidden layers (256,128,64,32) and $\lambda = 0.0003$). The accuracy score jumps up and down in a range of roughly 5%. Even though our 5-fold cross validation dampens this effect, one still expect some variation in the results due to this.

In training, loss improves until epoch 20 to 30 before starting to increase again as overfitting becomes a pressing issue. On the other hand, accuracy does not drop, instead it

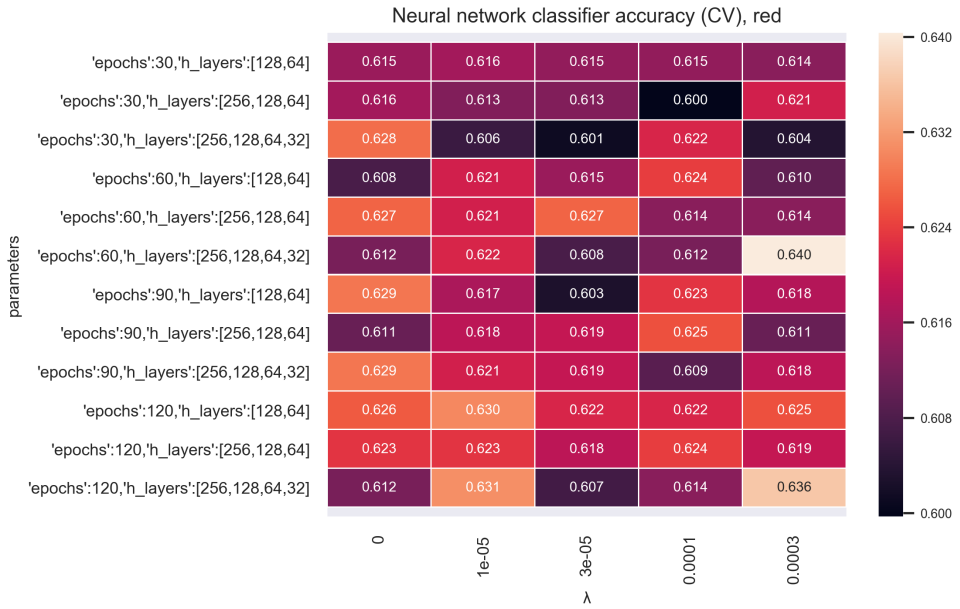


Figure 5: Neural network classifier cross-validated accuracy on validation sets

increases (slightly) with more training. Thus it is clear that selecting classification models and training solely based on accuracy could come at a slight cost with respect to the severity of misclassifications.

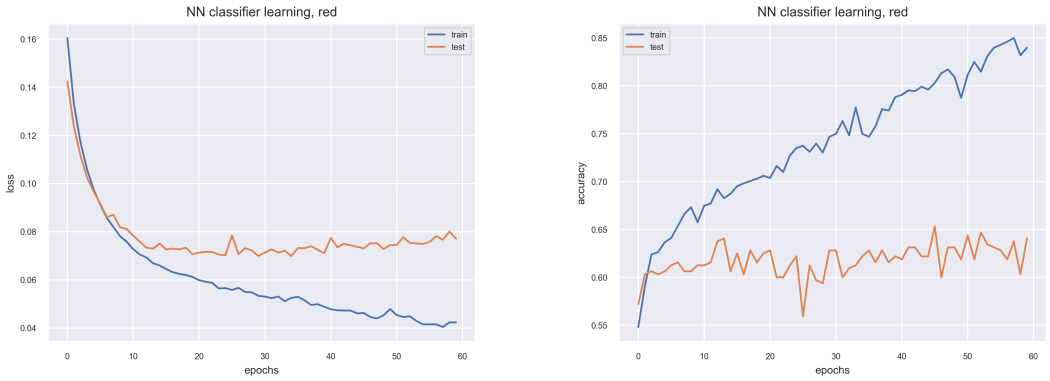
The two best models however seem to be quite deep neural networks, with some regularization added, perhaps somewhat surprising on a fairly simple data set. That said, the less complex models tested perform almost equally well.

The training accuracy evaluated using a regressor network with rounded predictions grows more slowly (see figure 15 in appendix B).

5.4 Support Vector Machine

We implemented different SVM kernels, using a trial-and-error approach to find the best one suited for this data. The default kernel is 'rbf', and this one worked significantly better than other kernels (linear', 'poly' and 'sigmoid'), for both datasets and for both classification and regression.

The other two hyperparameters that require tuning are the C-value and the γ-value. Figure 7 shows the accuracy scores obtained with different combinations of values. We see that a C-value lower than 1.0 gives the worst results. This makes sense, because the lower



(a) Learning loss

(b) Learning accuracy

Figure 6: Neural network classifier learning on the red wine data set. Hidden layers used are (256,128,64,32) and regularization $\lambda = 0.0003$.

the value of C, the larger the margin and the more misclassifications are allowed. So when C gets very small, the margin is too large and the model is underfitting the data; the classifier becomes too soft. We see that most of the C values above 0.1 give good results, as long as γ is between 0.01 and 1. For a high γ, the model gets more sensitive, meaning that less points that are far away from the separation line are considered. The fact that we are trying to classify 8 different classes here might cause the relatively lower number of λ, because it needs to take into account more points. Another reason might be that the differences are not very well defined, which means that there is no clear separation line. The heatmap for the white wine dataset looks almost the same, but has as optimal value for λ a value of 1.0 (see appendix B). This is probably due to the fact that the white wine dataset has more datapoints. For the regression case, for both datasets, a C-value of 1 with a λ-value of 0.1 gives the lowest MSE-score.

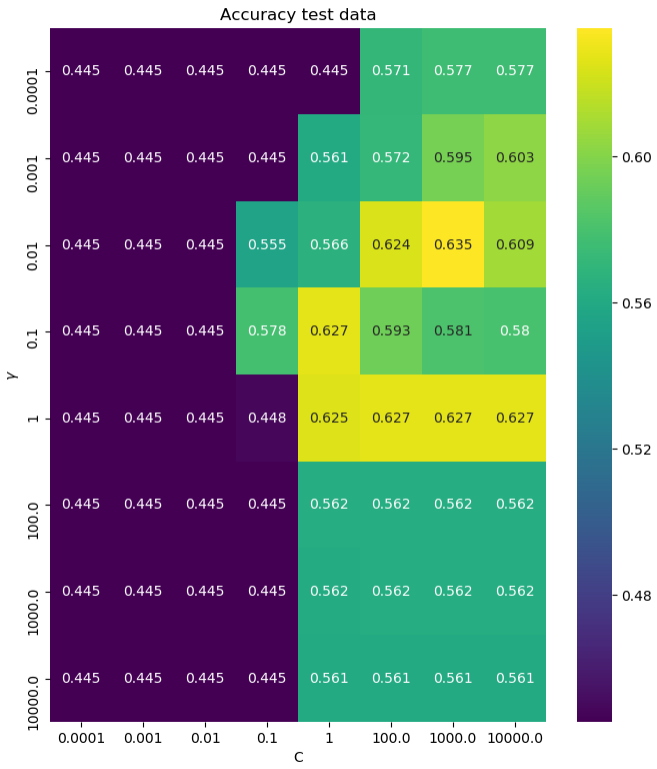


Figure 7: SVM accuracy score for different C- and γ-values

5.5 Random Forest

We tested random forest models with different limitations on the number of trees, the maximum number of instances in each leaf and the tree depths. The parameter grid is listed in table 3.

The cross-validated results for the classifier model on the white wine data set in figure 8 indicate that the best results are achieved with the tree depth limited to 12. However, this is obviously a very weak limitation since this allows for up to $2^{12} = 4096$ leaves which is more than the number of training samples in the red wine data. And no limit on the tree depth, which is the default settings in Scikit-Learn, gives roughly the same results. In addition, the model performs best with no limitation on the minimum number of samples in each leaf. It is remarkable that the nature of the algorithm, with many different trees constructed using different data (bagging) and with a random subset of features evaluated at each node, prevents too much suffering from overfitting. (Of course, the model overfits hard on the training data, but still performs well on the test data.) 100 trees are enough to achieve this effect for this data set.

Parameter	**Description**	**Values**
'n_estimators'	Number of trees	[100,500,1000]
'min_samples_leaf'	Minimum number of instances in each leaf	[1,2,3]
'max_depth'	Maximum number of splits before reaching a leaf	[10,11,12,13,None]

Table 3: Grid search parameters for random forest.

5.6 XGBoost

For XGBoost, we ran a grid search over different shrinkage parameters η, thresholds to split nodes γ as well as max tree depths. The classifier results on the red whine set, provided in figure 9, shows that highest accuracies are achiveved using $\eta = 0.3$, $\gamma = 0$, which are the default settings, and a max depth of seven. Like random forest, XGboost gives very good results that are not so sensitive to the hyperparameter settings in the range tested. Cross validation results for the white wine set are shown in figure 19 in appendix B. The best results are achieved with a higher maximum depth (10 or 11), which is likely due to the white set being larger.

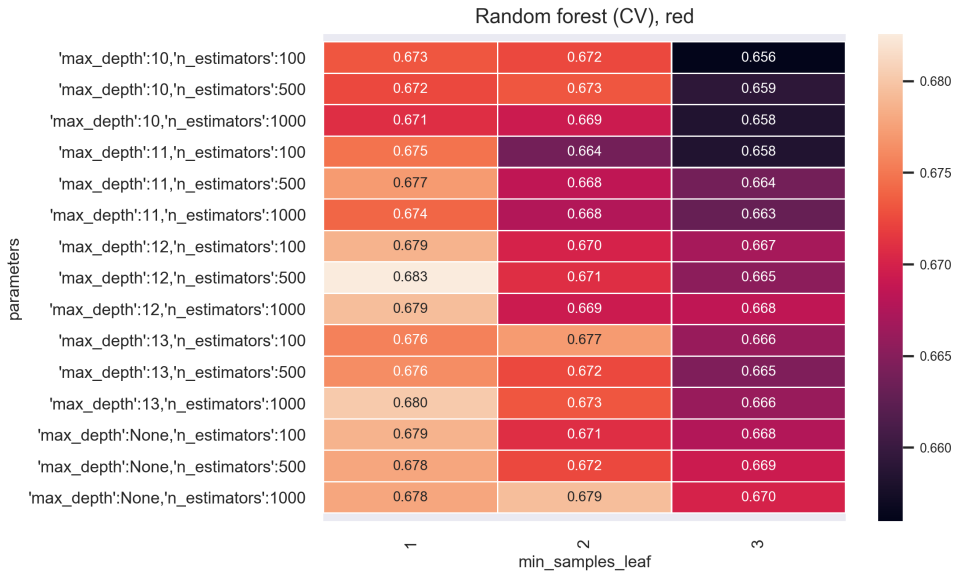


Figure 8: Random forest cross-validated accuracy on validation sets using the red wine data set. Limiting the maximum tree depth to 12 yields the best results. However, predictions are also good with no limit. It seems like 100 trees are enough for the algorithm to avoid bad results due to overfitting.

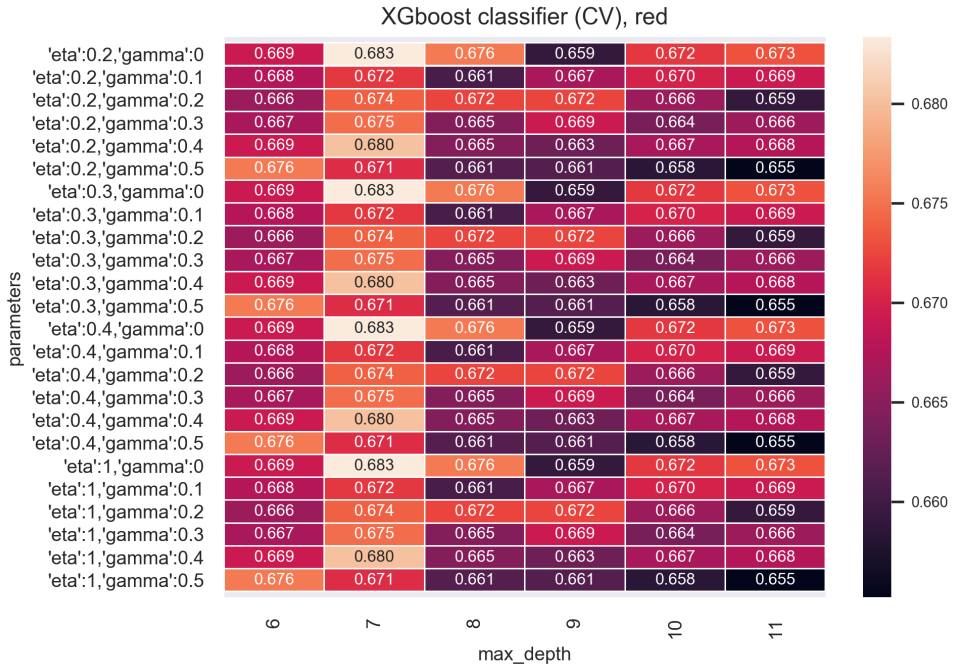


Figure 9: XGBoost classifier cross-validated accuracy on validation sets using the red wine data set.

5.7 Final results

Table 4 shows the scores for each optimized model, for both regression and classification approach and for both the red and white wine dataset. The optimized regression models were the models that had the lowest MSE, meaning they are not necessarily optimal with regards to minimizing the MAD or maximizing the accuracy. The first (upper) score is the *5-fold cross-validated* score of the optimized model on the training data, including a $\pm 2\sigma$ uncertainty range. The second (lower and *italic*) is the score on the holdout data. The best cross validated scores for each metric are in bold.

The uncertainty range presented by the cross-validation is the range in which most of the test-scores should fall, as the separate test set and the individual k-fold validation sets are fairly similar in size. For most models we observe this, although some fall out of the range. A test score below the range indicated by the cross validation could be a sign of overfitting, which can be a consequence of picking the best models according to their validation scores within the training data. We see this for the logistic regression on the white wine set, but attribute this to statistical flucatuations rather than a problem of overfitting; afterall it is one of the simpler models tested and we use only the default settings of Scikit-Learn. Believing that overfitting by hyperparameter selection from cross validation is less of a problem than the variability of the test scores, we focus on the mean cross validation scores when comparing models.

		Regression			Classification
		MSE	MAD	Accuracy ($T = 0.5$)	Accuracy
Red wine dataset	OLS	0.433 ± 0.026 *0.382*	0.513 ± 0.020 *0.475*	0.581 ± 0.043 *0.634*	-
	LR	-	-	-	0.576 ± 0.059 *0.604*
	NN	0.438 ± 0.033 *0.447*	0.504 ± 0.018 *0.501*	0.612 ± 0.028 *0.609*	0.636 ± 0.025 *0.634*
	SVM	0.418 ± 0.077 *0.388*	0.464 ± 0.044 *0.463*	0.639 ± 0.037 *0.613*	0.635 ± 0.035 *0.600*
	RF	**0.350** ± 0.022 *0.319*	**0.433** ± 0.019 *0.406*	0.665 ± 0.050 *0.725*	0.683 ± 0.058 *0.719*
	XG	0.368 ± 0.029 *0.328*	0.434 ± 0.024 *0.404*	**0.672** ± 0.048 *0.734*	**0.683** ± 0.042 *0.697*
White wine dataset	Ridge	0.574 ± 0.050 *0.543*	0.585 ± 0.019 *0.586*	0.522 ± 0.022 *0.506*	-
	LR	-	-	-	0.550 ± 0.032 *0.513*
	NN	0.500 ± 0.056 *0.500*	0.545 ± 0.032 *0.564*	0.560 ± 0.043 *0.519*	0.618 ± 0.019 *0.639*
	SVM	0.460 ± 0.080 *0.404*	**0.415** ± 0.060 *0.437*	0.641 ± 0.035 *0.676*	0.635 ± 0.028 *0.682*
	RF	**0.393** ± 0.055 *0.358*	0.450 ± 0.030 *0.436*	**0.662** ± 0.034 *0.658*	**0.666** ± 0.045 *0.688*
	XG	0.407 ± 0.062 *0.372*	0.444 ± 0.028 *0.428*	0.654 ± 0.023 *0.669*	0.656 ± 0.047 *0.683*

Table 4: Regression and classification scores on the wine datasets

5.7.1 Regression

For regression models, we focus on the MSE and accuracy. We use the MAD score to compare our scores with the paper by Cortez et al. [6].

Our results seem to favour the random forest model, with a MSE of 0.350±0.022 and 0.393±0.055 on the red and white data sets, respectively. This is closely followed by the XBGoost model (MSE 0.368±0.029 and 0.407±0.062). That said, taking into account the statistical uncertainties, it is hard to say firmly that one is better than the other. Because both are ensemble algorithms using decision trees as base models, it might make sense that their scores are similar. The SVM model falls behind on the MSE, although it performs reasonably well on accuracy, indicating that the misclassifications are more severe. This is confirmed by the confusion matrices, shown for the white data set in figure 22. The SVM regressor model classifies 40 data points wrong by 2 quality points, compared to the random forest's 19. The results for the neural network are rather unimpressive. One reason might be that we are working with relatively small data sets that does not give the network enough information to work with. Also, tuning a network is a complex process, often involving trial and error. Although we did extensive testing, it might still be possible to improve the models a bit more.

Next we will compare our scores with the scores achieved by Cortez et al. [6]. They implemented a multiple regression (ML), a neural network (NN) and a support vector machine (SVM) model, with the default parameters of the libraries they used (in R). We see that the results for linear/multiple regression are similar. We achieved slightly higher results for the SVM, but not statistically significant. We also report somewhat better results for the neural network, especially for the white wine set. We achieved a MAD of 0.545 and an accuracy of 0.560 vs a MAD of 0.58 and accuracy 0.526 reported by Cortez et al. This could be expected as we put in quite some effort into optimizing our network, while they used a relatively simple model with default hyperparameters. That said, considering the amount of time and effort we've put into optimizing the neural network, including the computational time for running the grid search, the results are not that much better.

5.7.2 Classification

Random forest and XGBoost are the best classification models as well, both obtaining very similar accuracy scores. For the classification case we see that overall we achieve a higher accuracy, and except from the SVM, all models get higher accuracy scores for classification as compared to the accuracy scores for regression. This should not be a surprise since the regression models are not selected for best accuracies. On the other hand, the classification model hyperparameters were selected to optimize accuracy only, and whether they miss by 1 or 3 classes in their predictions is not a factor.

6 Conclusion

Predicting wine quality based on its physicochemical properties is of immense interest to wine producers. Establishing relationships between chemical properties and quality enables producers to focus on these properties in order to make sure their wine has a better quality in the end. A machine learning algorithm makes 'objective' predictions, and saves time and money. It can also help with determining the prices of wines. High quality wines can typically be higher priced.

In this research we aimed to predict the quality of Portugese wines, based on their physicochemical properties. We implemented six different models in order to find the one best able to fit the data best. We looked at both regression and classification.

We have found that the best models are the random forest or XGBoost, for both classification and regression. The random forest algorithm, which got the best scores in our runs (tied with XGBoost on the red whine set), achieved a classification accuracy of 68.3% on the red wine dataset and 66.6% on the white wine dataset. For the regression case an MSE-score of 0.350 was achieved for the red wine dataset and 0.393 for the white wine dataset. Lastly we looked at the accuracy for the regression, and we saw that the best model is able to predict 67.2% of the classes with $T = 0.5$ precision for the red wine dataset and 66.2% for the white wine dataset.

By using the feature importance functionality of the random forest algorithm we were able to show that the most important variable for predicting the quality is the amount of alcohol, but the other variables followed closely.

We compared our results with the results obtained in the paper by Cortez et al. [6]. We saw that for the corresponding models (linear regression, SVM and neural networks) our scores were similar or slightly better. What is of main interest here is that the random forest and XGBoost performed significantly better, meaning it would be interesting to do more research in the direction of decision-tree based models.

7 Discussion

This project has a few limitations and options for further research that we will discuss in this section.

The wine dataset we used for this research allows both classification and regression, and after applying both we saw that the accuracy scores for classification are higher than the accuracy scores for regression. There are a few warnings with this comparison; firstly for the classification accuracy we don't have any knowledge about how far off the actual predictions are, we only know the amount of correct predictions. For regression we have the MSE which shows how far off the predictions were. Secondly, classification categorizes the wines in hard categories, i.e. 3-4-5-6-7-8-9, while regression enables scores with a decimal, i.e. 5.4-6.2-7.5 etc. Lastly, a classification model does not have any knowledge of the class order. Since the quality is a score from 0 to 10, there is order between the classes. There are some options for classification that keep the order, such as the ordinal target function approach as introduced by [13], but this was beyond the scope of our project. For future research this could be a promising direction. This problem can be approached as both classification and regression, each with pros and cons. In the end it is up to the wine sector to establish what their main goal and objectives are, and choose a method that suits these.

In our results, we saw that the uncertainty range, for both the regression and classification accuracy scores, are fairly large. We came across this problem during our project, and it seems to be a characteristic of this dataset. The confidence interval of the accuracy score of classification for the random forest is ± 0.058, which means that the actual accuracy is estimated somewhere between 74.1% and 62.5%, which is a range of more than 12%. That is quite a big uncertainty range. For the white wine dataset the uncertainty is slightly lower, simply because the dataset is more than 3 times larger so the models have more data. This should be taken into account when using the models.

The dataset used does not possess any wines with a quality score of 1, 2 or 10. This means that our algorithms never saw these examples and is not able to train on these, i.e. for the classification case the algorithm will never be able to predict that a wine that has a quality score of 1, 2 or 10. This could be solved by updating the training data, for example by letting the algorithm use online learning, i.e. it will learn from every new datapoint. This way the model is continuously updated and improved, and when new wines are included in the dataset that have a score of 1, 2 or 10 the model will (slowly, but eventually) learn to predict these as well. To ensure that it will actually pay attention to these wines, the class weights for these categories could be increased. Since our regression models are unbounded, they could theoretically predict quality scores above 10. This is not likely to happen as the highest score in the data set used is 9 (very few examples). It would be fun to see if a perfect storm of chemical properties occurred, resulting in an extreme prediction.

Another problem with the dataset is that it is unbalanced. There are many more wines with a quality between 5-7. This leads to the fact that the algorithms get very good at predicting wines with a quality between 5-7, but not so good at predicting wines with scores outside this range. To solve this, we tried adding class weights in the neural network for classification; this led to predicting more 1-2's and 8-9's and a lower overall accuracy, as expected. For future research, this area could be explored more.

What was deemed outside the scope of this project, but an interesting option for future research, would be to investigate which combinations of chemical features tend to exist in good wines. The fact that alcohol is the most important feature (determined by the random forest algorithm) only says something about the size of the effect, but nothing about the direction and combinations with other properties. To extract some knowledge about this, we fitted shallow decision tree models to the data (see figure 20 and 21 in appendix C). Red wines with alcohol content above 11.6% and sulphate content above 0.7 are deemed the best (score=7), whereas red wines with low alcohol content (10.2%) are less than spectacular. For whites, strong wines (alcohol>12.8%) and very weak wines with low acidity are deemed the best, according to this simple model. Further effort into explaining machine learning results would help wine producers to aim for the perfect combination of physicochemical properties.

Lastly, wine quality is hard to predict. Taste is very subjective, meaning that different people will like different wines, and someone could give a wine a 10 while someone else could give the same wine a 1. The good thing about these algorithms is that they can be considered objective, but of course only as objective as the data allows. The models are trained on quality scores given by humans, and it will learn these scores and predict based on these scores. Because taste is subjective, the predictions of the algorithm can never satisfy everyone's taste. Besides, it is not our intention to render wine tasters and sommeliers jobless, afterall human tasting and evaluation is part of the attraction and soul of the wine business.

8 References

[1] P. Karlsson, "World wine production reaches record level in 2018, consumption is stable," 2019.

[2] Statistica, "Leading countries in wine export worldwide in 2018, based on volume (in million hectoliters)," 2019. Accessed: 08-12-2019.

[3] S. Ebeler, *Flavor Chemistry — Thirty Years of Progress*. Kluwer Academic Publishers, 1999.

[4] S. A. S. . Solutions, "Artificial intelligence, what it is and why it matters," 2019. Accessed: 08-12-2019.

[5] R. M. D. Smith, "Making sense of taste," vol. Special Issue 16(3), pp. 84–92, 2006.

[6] Cortez, P. et al., "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47(4), pp. 547–553, 2009.

[7] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[8] R. Berwick, "An idiot's guide to support vector machines (svms)," n.d. Accessed 27-11-2019.

[9] O. C. Carrasco, "Support vector machines for classification," *Towards Data Science*, 2019.

[10] M. Hjorth-Jensen, "Lectures notes in fys-stk4155. data analysis and machine learning: Linear regression and more advanced regression analysis," 2019. Accessed: 27-11-2019.

[11] Kernel SVM, "Support vector machine regression," 2019. Accessed: 27-11-2019.

[12] J. L. Jane Elith, "Boosted regression trees for ecological modeling," 2017. Accessed: 09-12-2019.

[13] J. Cheng and G. Pollastri, "A neural network approach to ordinal regression," *IEEE Int. Jt. Conf. Neural Networks 2008 IJCNN 2008 IEEE World Congr. Comput. Intell*, pp. 1279–1284, 2008.

[14] C. G. Tianqi Chen, "Xgboost: A scalable tree boosting system," 2016. Accessed: 09-12-2019.

A Data preprocessing

A.1 Correlation matrix

Figure 10 and figure 11 show the correlation of the red and the white wine dataset, respectively.

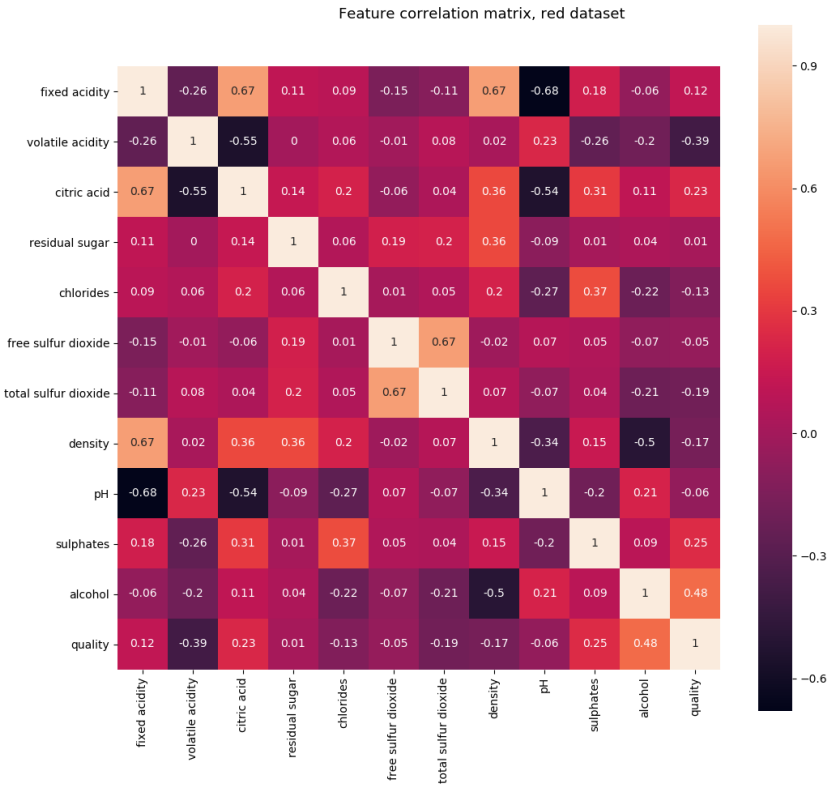


Figure 10: Correlation matrix red wine dataset

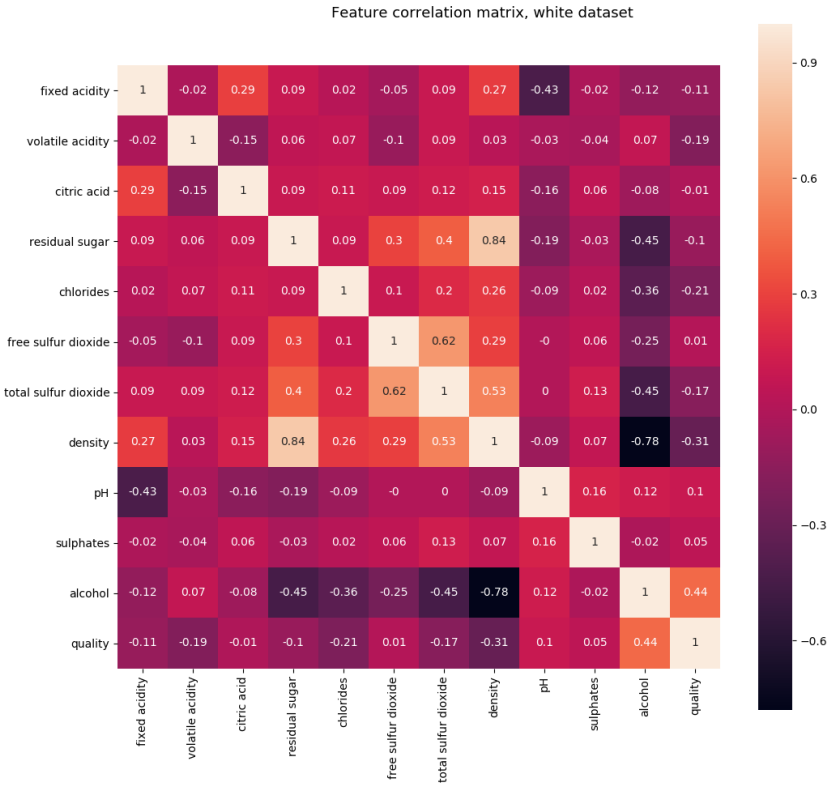


Figure 11: Correlation matrix white wine dataset

A.2 Outlier analysis

Figure 12 and figure 13 show the histograms of the variables for the red and the white wine datasets, respectively.

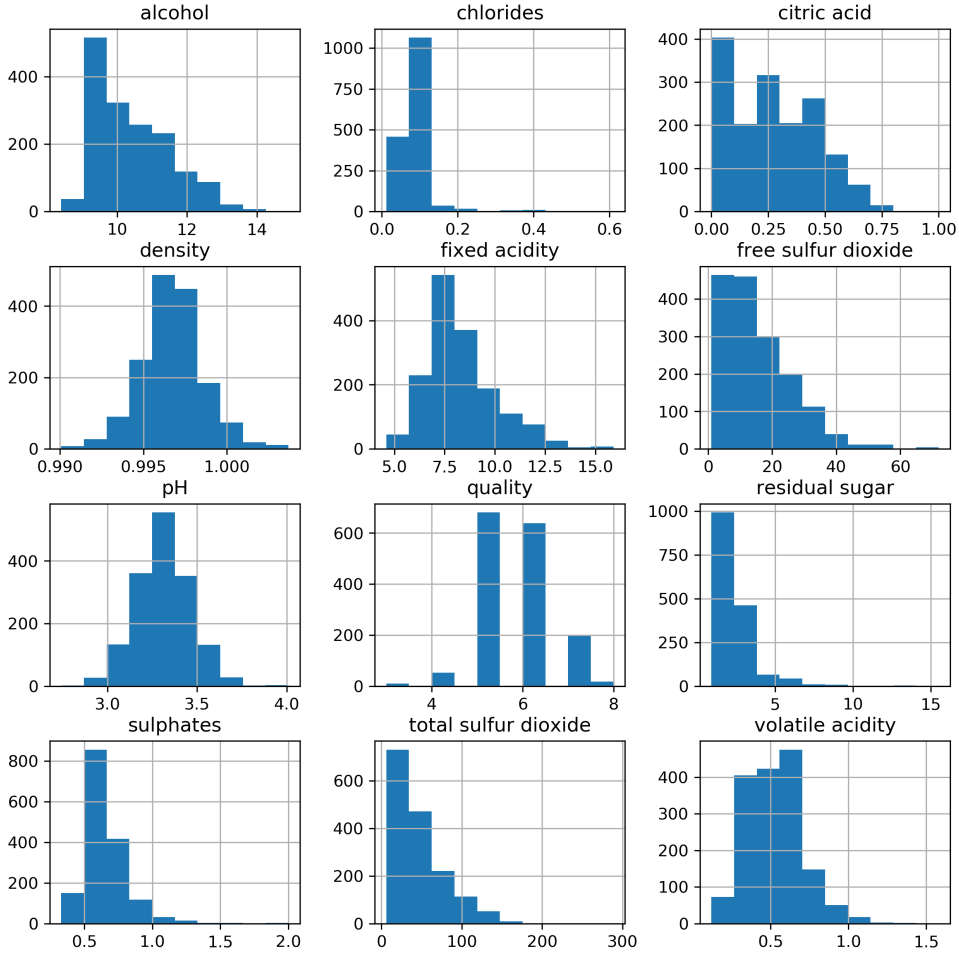


Figure 12: Histograms of the variables of the red wine dataset

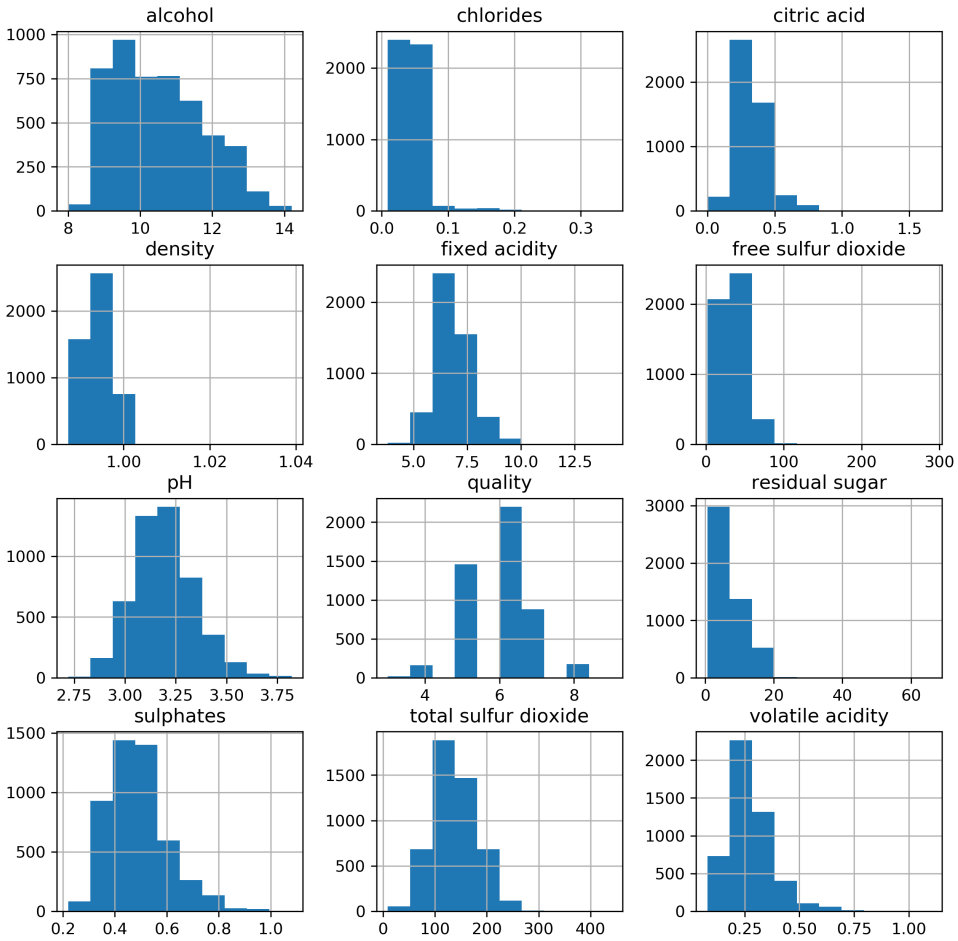
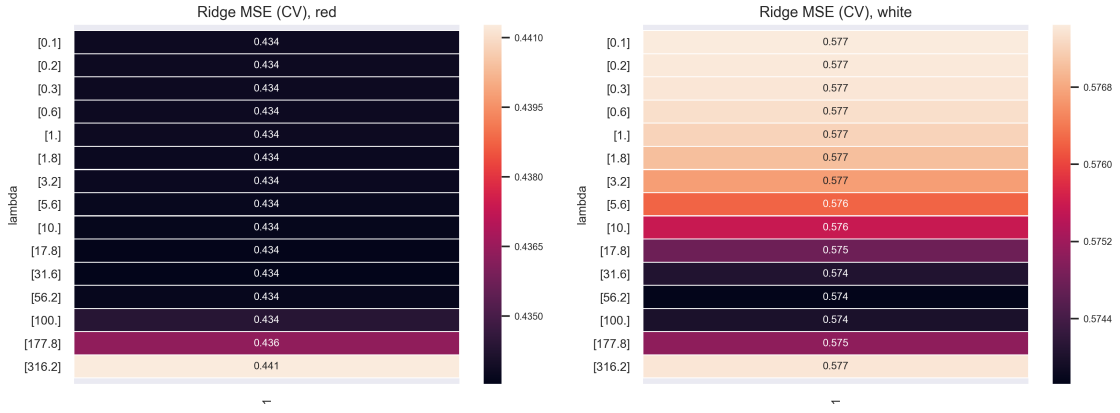


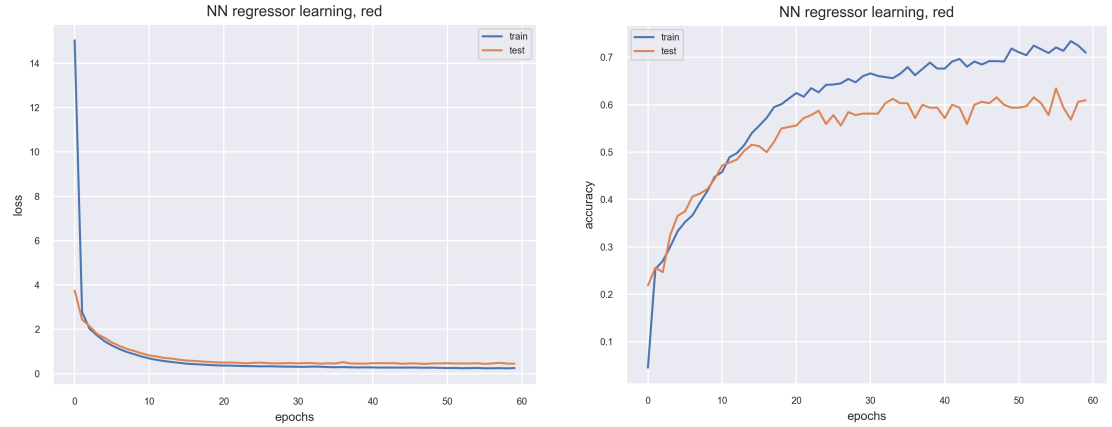
Figure 13: Histograms of the variables of the white wine dataset



(a) Red (b) White

Figure 14: Linear regression dependence on L2 regularization λ

B Model tuning/learning



(a) Learning loss (b) Learning MSE

Figure 15: Neural network regressor learning on the red wine data set.

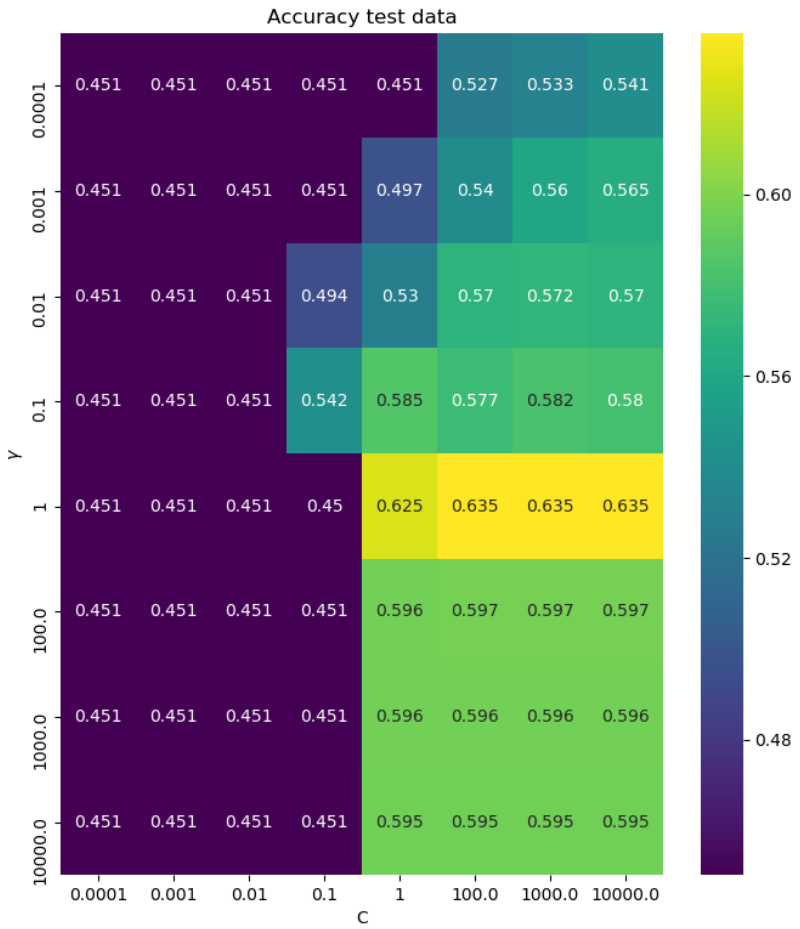


Figure 16: Accuracy of the SVM for different C- and λ-values, whites.

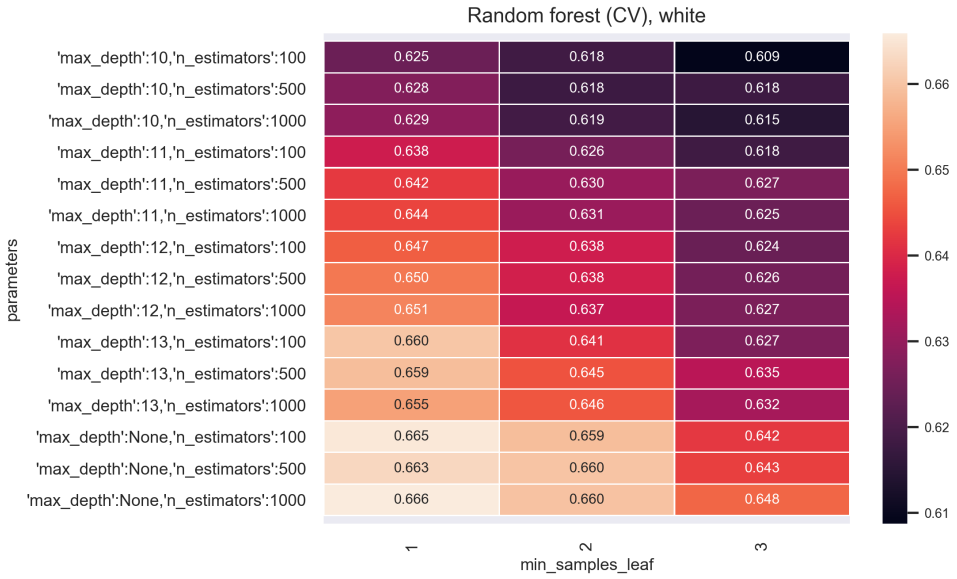


Figure 17: Random forest classifier cross-validated MSE on validation sets using the white wine data set. The best results are achieved with no limitations on the leaf sizes and on the depths of the trees.

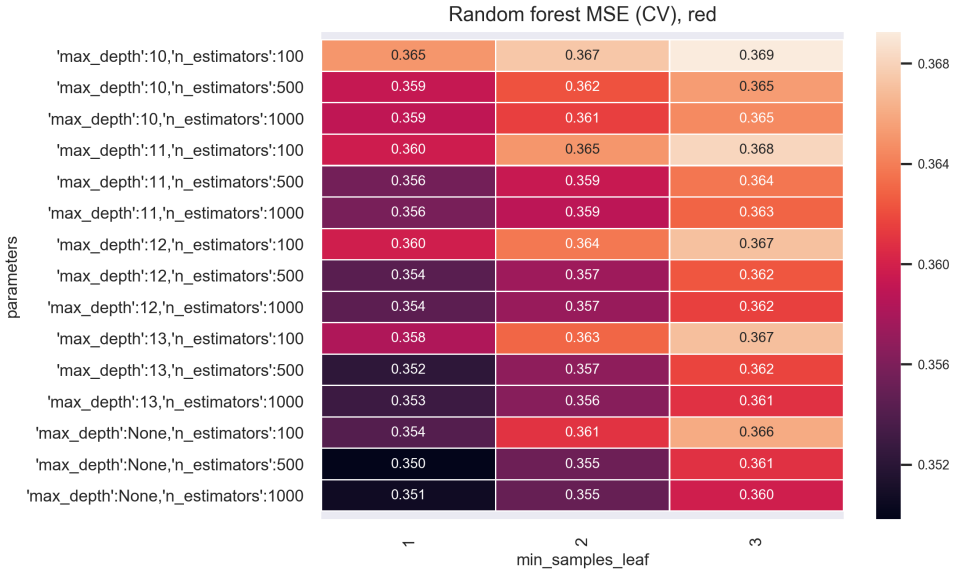


Figure 18: Random forest regressor cross-validated MSE on validation sets using the red wine data set.

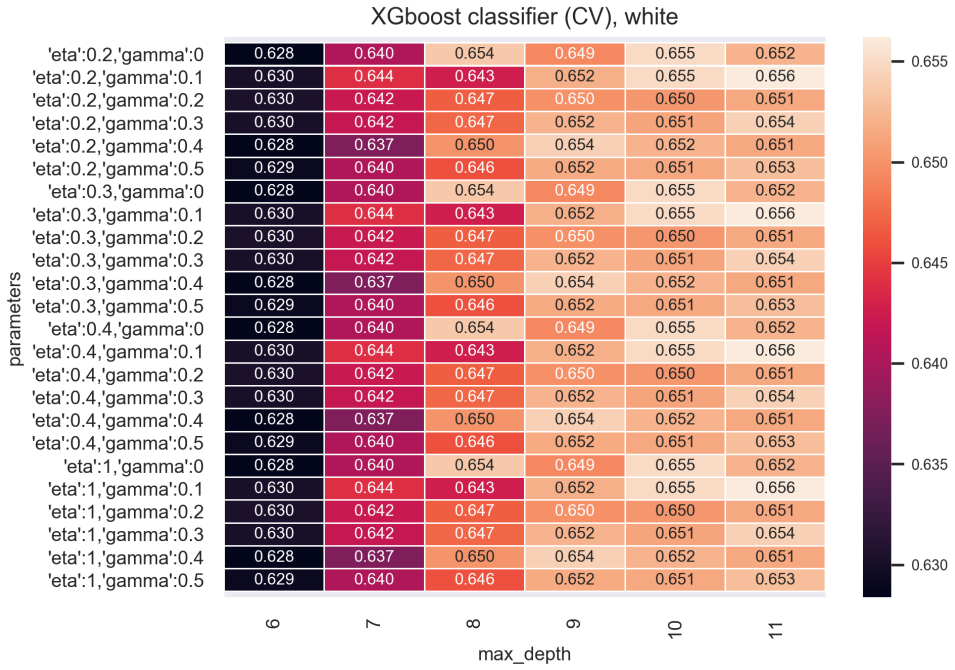


Figure 19: XGBoost classifier cross-validated accuracy on validation sets using the white wine data set.

C Interpretation: shallow decision tree

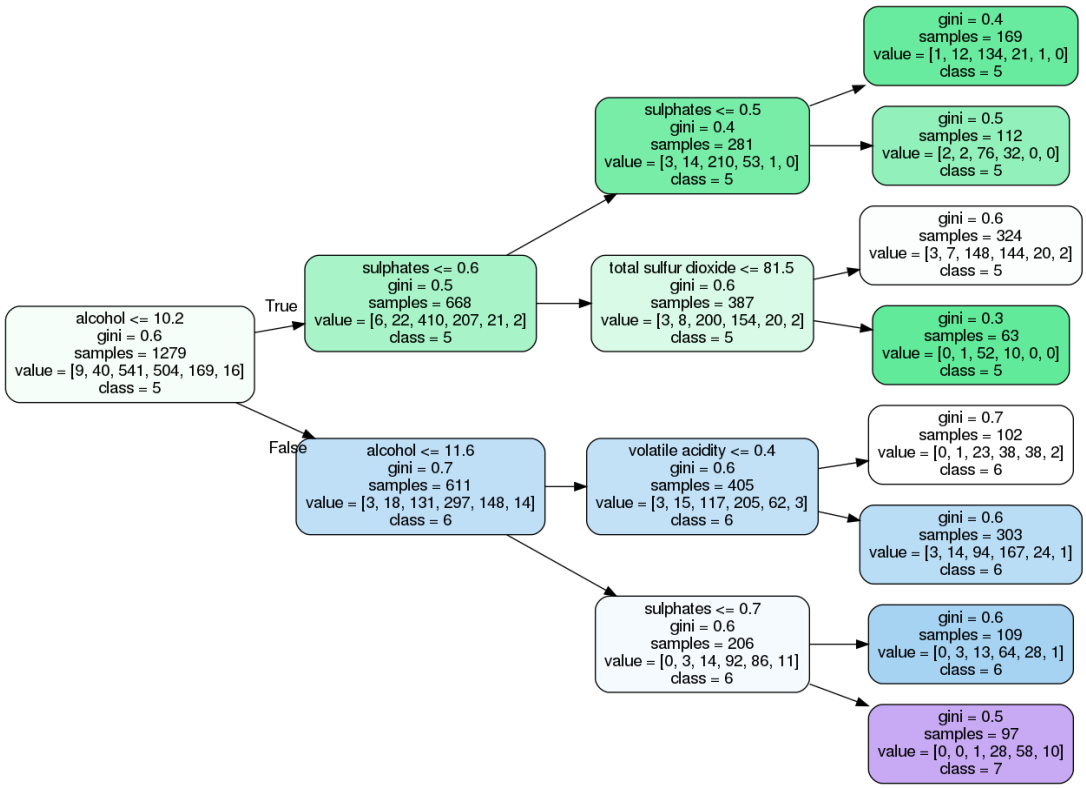


Figure 20: Shallow decision tree on the red wine set. Following the lower branches on the tree, one that see that wines with alcohol content above 11.6% and sulphate content above 0.7 are deemed the best (score=7). Red wines with low alcohol content ($\leq$ 10.2%) are less than spectacular according to this simple model.

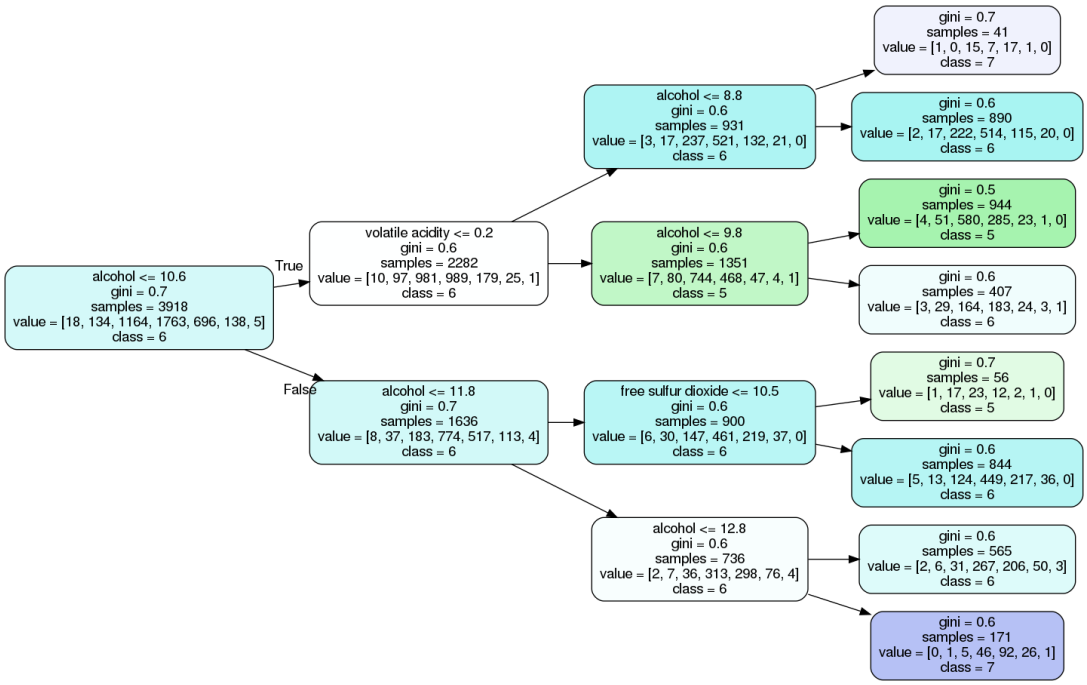


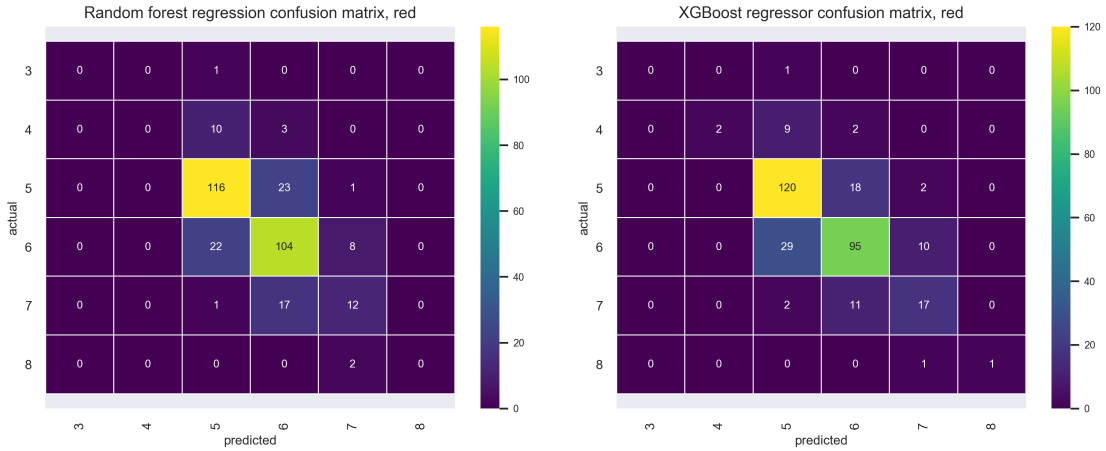
Figure 21: Shallow decision tree on the white wine set. Strong wines (alcohol $> 12.8\%$) and very weak wines with low acidity are deemed the best.

D Confusion matrices

(a) Confusion matrix RF, white

(b) Confusion matrix SVM, white

Figure 22: Confusion matrices for the random forest and the SVM regressors on the final test run on the white wine dataset. The SVM has more severe misclassifications.



(a) Confusion matrix RF

(b) Confusion matrix XGBoost

Figure 23: Confusion matrices for the random forest and the XGBoost of the final test run on the red wine dataset