Compulsory Exercise 2: Assessing red wine quality

Jakob Bergset Heide Celine Natalie Badin Olsson

18 mai, 2023

Abstract

In this analysis, we will be examining a dataset containing information about various physicochemical properties of red wine and its corresponding quality ratings. The data is related to the red variant of the Portuguese "Vinho Verde" wine. The dataset is publicly available and can be accessed from the UCI Machine Learning Repository*. We apply three classification methods to examine this data set - k-nearest-neighbours (KNN), logistic regression and random forests. The random forest model turned out to be able to predict new data very well. From the last two models, we gained some inference about the quality of the red wine, namely that there are 4 physicochemical ingredients that are essential to the quality of the red wine.

Introduction

In this project, we will investigate the physiochemical properties of red wine using the Red Wine Quality data set (https://archive.ics.uci.edu/ml/datasets/wine+quality). In this data set, there are 1599 different wines with 12 variables each, one of which is denoted as "quality", which is a score from 0-10 given each wine based on a sensory rating. As quality is a categorical variable, we consider this to be a classification problem, with quality as the response and the other variables as predictors. Our findings can be useful to wine producers, or to predict the rating of new wines, based only on their physicochemical properties. Some of the models we build can also be used by wine enthusiasts to quickly determine if a wine is likely to be a good red wine.

Our main goal is inference - but naturally, we also want to build good models in order to predict the quality of new wines. We apply different classification models of varying complexity. In addition, we also compare the models in terms of interpretability and performance.

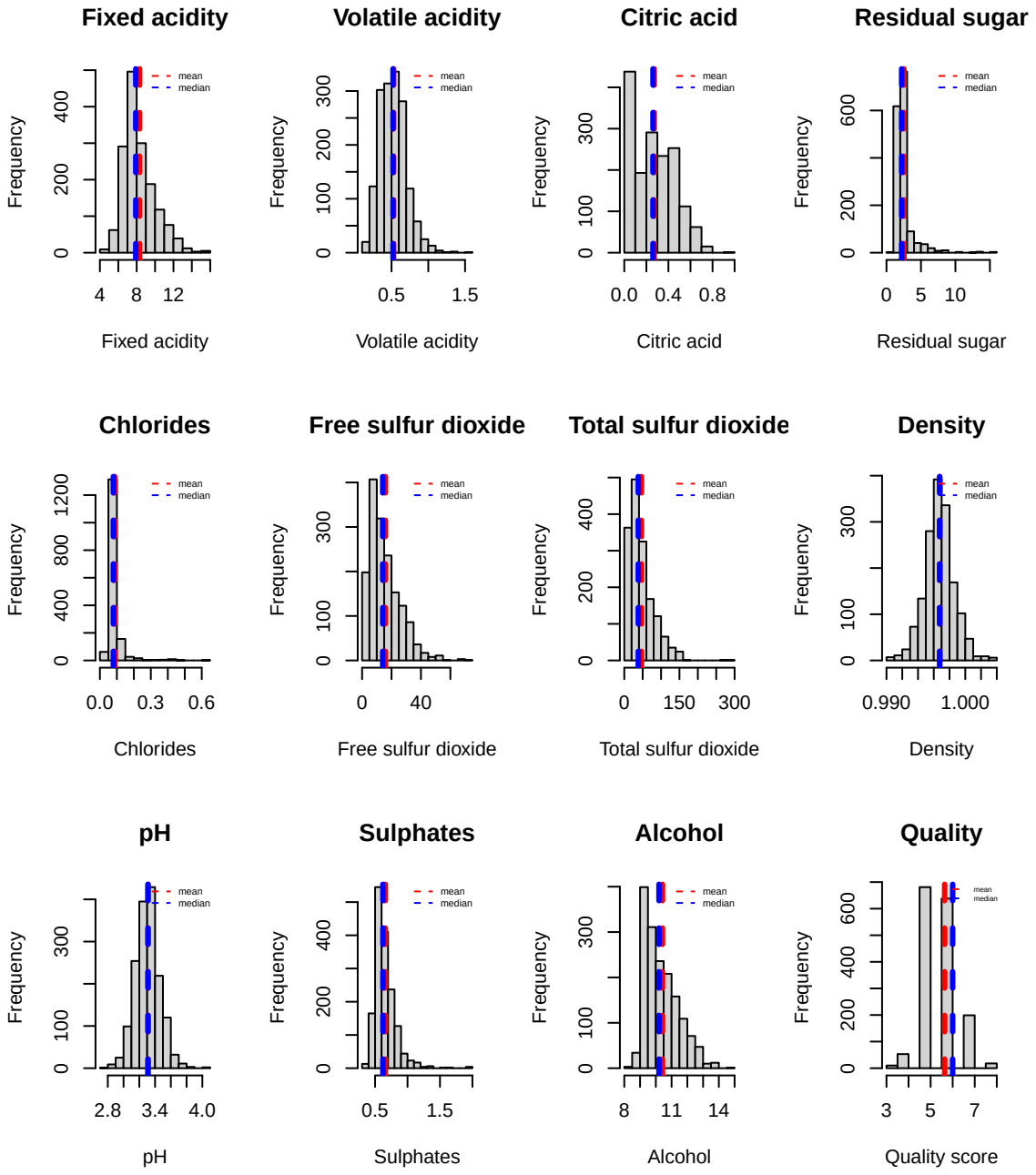
Descriptive data analysis/statistics

First we want to look at the data we are working with. As mentioned there are 12 variables in this data set each describing a different feature of the red wine:

- Fixed acidity: the concentration of non-volatile acids. It contributes to the overall acidity of the wine.
- Volatile acidity: the concentration of volatile acids, mostly acetic acid. A high concentration of volatile acids may give the wine a vinegary taste.
- Citric acid: the concentration of citric acid. It is often present in small quantities in red wine, and can provide a citric taste.
- Residual sugar: Indicates how much sugar remains in the wine after the fermentation has stopped. The higher the residual sugar content, the sweeter the wine.
- Chlorides: The quantity of salts in the red wine.
- Free sulfur dioxide: the amount of sulfur dioxide which is not bound to other compounds.
- Total sulfur dioxide: overall concentration of free and bound forms of sulfur dioxide. It is often used as a preservative and an antioxidant.

- Density: mass of the wine per unit volume. It is mostly influenced by alcohol and sugar content.
- pH: measurement of the acidity/basicity of the red wine, on a scale from 0-14. 0 is the most acidic and 14 is the most basic.
- Sulphates: compounds added to wine as preservatives and antioxidants.
- Alcohol: the percentage of alcohol in the red wine.
- Quality: the output variable ranging from 1-10 based on the tasters sensory rating.

We look at the distribution of all the different variables below.



We see that the two measurements of central tendency, mean and median, are about equal in value for all variables. A more detailed summary of the data is printed below.

```
summary(data)
```

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##        pH          sulphates         alcohol         quality
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
##  Median :3.310   Median :0.6200   Median :10.20   Median :6.000
##  Mean   :3.311   Mean   :0.6581   Mean   :10.42   Mean   :5.636
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000
```

In this summary we can see the mean and median, as well as the range of all the variables more clearly. For example, even though the quality of the wine can get a score between 0 and 10, the range of the quality in the data set only goes from 3 to 8.

We also want to look at the standard deviation and the variance for the different variables. Both are measurements of how great the dispersion of the data points are. Below the standard deviation is printed for all the different features.

```
sapply(data, sd)
```

```
##        fixed.acidity     volatile.acidity          citric.acid
##          1.741096318          0.179059704          0.194801137
##       residual.sugar            chlorides  free.sulfur.dioxide
##          1.409928060          0.047065302         10.460156970
## total.sulfur.dioxide              density                   pH
##         32.895324478          0.001887334          0.154386465
##            sulphates              alcohol              quality
##          0.169506980          1.065667582          0.807569440
```

Most of the variables have a relatively low standard deviation, which means the average distance between a data point and the mean is low. A standard deviation close to zero indicates that most of the data points are close to the mean. Two variables which stand out are free sulfur dioxide with $sd = 10,46$ and total sulfur dioxide with $sd = 32.89$. This means that the average distance between a data point and the mean is high, and that the dispersion of the data points are high.

The variance of the different features is printed below.

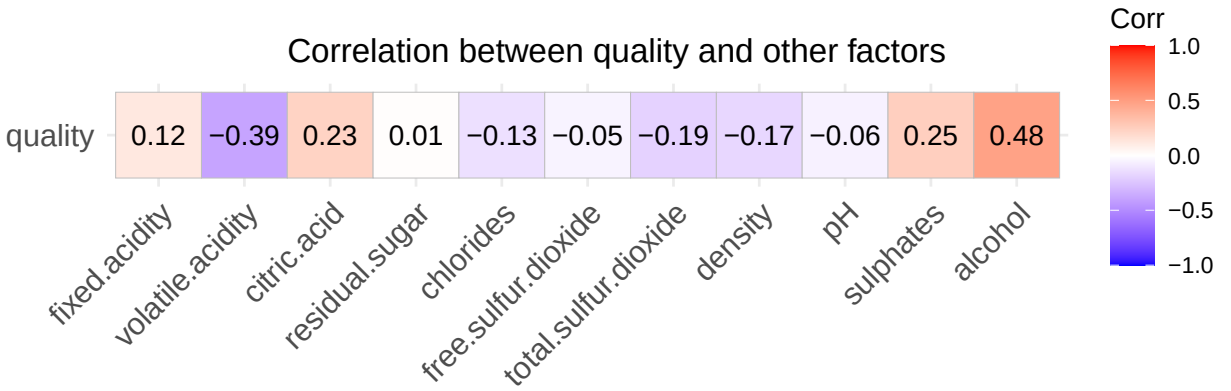
```
sapply(data, var)
```

```
##        fixed.acidity     volatile.acidity          citric.acid
##         3.031416e+00         3.206238e-02         3.794748e-02
##       residual.sugar            chlorides  free.sulfur.dioxide
##         1.987897e+00         2.215143e-03         1.094149e+02
## total.sulfur.dioxide              density                   pH
##         1.082102e+03         3.562029e-06         2.383518e-02
##            sulphates              alcohol              quality
##         2.873262e-02         1.135647e+00         6.521684e-01
```

Since the variance is the standard deviation squared we see the same results as with standard deviation, only on a smaller scale. A higher variance indicates a greater dispersion of the data points, and a lower variance indicates that the data points are closer to the mean.

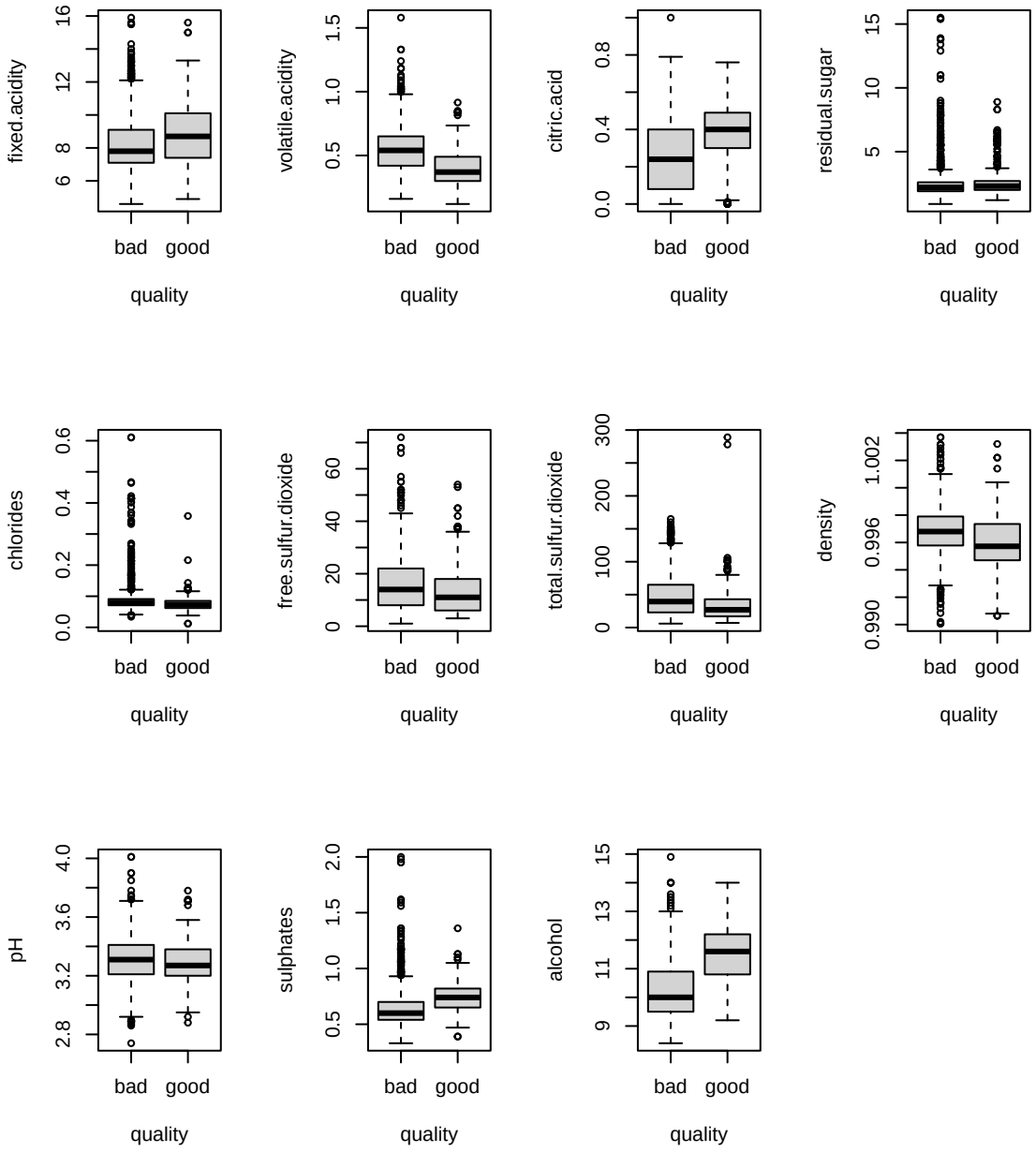
To get an idea of which factors might affect the quality of the wine we can start by looking at the correlation between the quality and the other factors. This is done by using the `cor()` function on our data set.

```
correlation <- round(cor(data), 2)

ggcorrplot(correlation[1:11, 12, drop = FALSE], lab = TRUE, title = "Correlation between quality and other factors") +
    theme(plot.title.position = "plot", plot.title = element_text(hjust = 0.5))
```



When looking at the correlation between the quality and the predictors we can assume that there is a correlation if the absolute value is approximately above 0.5. We see that the only predictor that comes close to this is the alcohol predictor.

Until now we have regarded the response variable quality as continuous. Now we want to look at the correlation between the quality and all the other predictors, but now we consider the quality variable as categorical. We give all wines with a score 6 or higher the quality "good", and all wines with a score lower than 6 the quality "bad". The correlations can be seen in the plot below.



From the boxplot above we again see that alcohol might affect the quality of the wine. So far it is difficult to say anything more about which variables have the greatest effect on the quality of the wine.

Methods

We will employ several methods to analyse this dataset, and compare their performance as well as their interpretability.

KNN

The K-Nearest Neighbours (KNN) algorithm is a non-parametric method that can be used for both classification and regression. When given a point and a chosen number of neighbours K, the KNN-method finds the K points in the training data which is closest to it, in Euclidean distance. In other words, it finds the K nearest neighbours of the given point. The method assigns the given point a value by a majority vote (classification) or by calculating the mean value of the neighbours (regression). This way it predicts the value of the given point.

But what K-value should we choose? When using a large K-value we look at a large number of neighbours and we get a smoother decision boundary. In addition, we can reduce the effect of outliers in our data set, but it can also cause the model to lose sensitivity to patterns and make the result less accurate; underfitting. On the other hand, when choosing a small K-value we look at just a few of the nearest neighbours. We get a flexible decision boundary where the model can capture more detailed patterns and variations in the data. However a small K-value makes the model more inclined to noise and outliers; overfitting. This is exactly the issue of the bias-variance tradeoff. To find the optimal value of K we use cross-validation to compare the predicted error for different values of K.

When using the KNN-method we also need to consider the curse of dimensionality. KNN works well when there are few predictors and a lot of observations, but when the number of predictors gets too high the effectiveness of the KNN-method drops drastically. This is because when dealing with high dimensions of the predictor space, the nearest neighbours tend to be far away, and the method is no longer local. For example when the number of predictors gets larger than the number of observations, the KNN-method will most likely lose effectiveness. In our analysis we have 1599 observations and just 11 predictors, and therefore steer clear of the curse.

It is also worth noting that KNN does not provide us with any inference about which predictors are important to the quality. It is thereby a limited model in the sense that we can only use it for predicting the quality of new wines.

Decision trees

A classification tree is a method used for predictive modeling. It is a decision tree that recursively partitions data into smaller subsets based on a set of splitting rules. Each split is based on the attribute that provides the most information gain in terms of improving the classification accuracy. The final result is a tree where each leaf node represents a class label. The main advantage of trees is that they are easily interpretable and illustrative, however their prediction accuracy is not necessarily so good, as we will see in the implementation later.

In our dataset, a leaf node will be a quality rating (from 0 to 10), whilst the splits will be predictors which are deemed most important (in each subset of data) to the wine quality by the tree algorithm.

Another disadvantage of decision trees is that they are prone to overfitting. To account for this, we can prune the tree back - for example by using k-fold cross validation. In the CV, we have to make a choice of measuring the error, which is typically done using the misclassification error.

Another model we will use is random trees. Compared to the single decision tree, random forest models are quite complex. It is a method that uses many decision trees in order to model the data. In short, the idea of random forests is to a) reduce the high variability of the trees by averaging a large number of trees created from bootstrapped samples (referred to as bagging), and b) only allow for a few random predictors to be used at each split. If there is a strong predictor in our dataset, then b) is especially important, since most of the trees from the bootstrapped samples would likely have this predictor as their root node and thus be highly correlated.

Logistic regression

Another method for classification is logistic regression. Logistic regression is a method which outputs the probability of an observation belonging to either class (as we will be looking at a binary class problem). The probability function p can be described as

$$p = \frac{\exp \beta X}{1 + \exp \beta X}$$

where β is the vector of coefficients and X is the design matrix. The coefficients β are estimated from the maximum likelihood. In order to convert the probability to a class, we use a cut-off value of p = 0.5 (i.e. a wine which gives p $>$ 0.5 is classified as "good").

An advantage of logistic regression is that it gives us the estimated coefficients corresponding to each predictor. These coefficients can be interpreted in the following way: if we increase the predictor x_{ij} by one unit, then the odds of $Y_i = 1$ (in our case: quality = "good") by a factor of $\exp \beta_j$. A hypothesis test for statistical significance is also performed for each predictor. A potential issue of logistic regression is that it assumes that our data is linearly separable, which is rarely the case with real-world data.

Measuring model performance

We will use several methods to compare our model performances. With most classification models, predictive accuracy is the main goal, and so we measure the accuracy of each model on the test data. This is done by first creating a confusion matrix, and taking the sum of the diagonal (which are the correctly classified observations) and dividing by the sum of the entire matrix (the total sum of observations). A model with higher accuracy is obviously preferable if we want to predict the quality of a new wine.

In order to tune our hyperparameters, we use k-fold cross validation in both the KNN and decision trees. In KNN, the hyperparameter is how many neighbours should be considered when classifying each observation. In the decision trees, we need to choose how many leaves the tree should have. Moreover, in the random trees, we need to choose the number of predictors used in each split, as well as the total number of trees in the forest. We will consider the out-of-bag error to try and find the best choices of these parameters.

In the last part of our analysis with the random forest and logistic regression, we will convert the categorical variable quality (0-10) to a binary variable ("bad"-"good"). This allows us to further analyse the model performances by calculating the ROC curves and their corresponding AUC score. The ROC (receiver operating characteristic) curve is a plot of the true positive rate against the false positive rate. It is essentially an illustration of a classification model's ability to correctly identify positive cases, as the discrimination threshold of the model is varied. The AUC (area-under-curve) score is exactly what its name suggests - the area under the ROC curve. We want the AUC score as close to 1 as possible.

Results and interpretation

We divide our data set into training and test data, and convert the response "quality" to a factor.

```
setwd("C:\\Users\\jkbhe\\OneDrive\\Documents\\TMA4268 Statlearn\\code")

wine = read.csv("winequality-red.csv")

# Dividing the data set into training and test data
set.seed(2023)
# 70% of the sample size for training set
training_set_size <- floor(0.7 * nrow(wine))
train_ind <- sample(seq_len(nrow(wine)), size = training_set_size)
train = wine[train_ind, ]
test = wine[-train_ind, ]
```

```r
# Convert the 'quality' response to a qualitative variable
train$quality = as.factor(train$quality)
test$quality = as.factor(test$quality)
class(train$quality)
```

```
## [1] "factor"
```

```r
class(test$quality)
```

```
## [1] "factor"
```

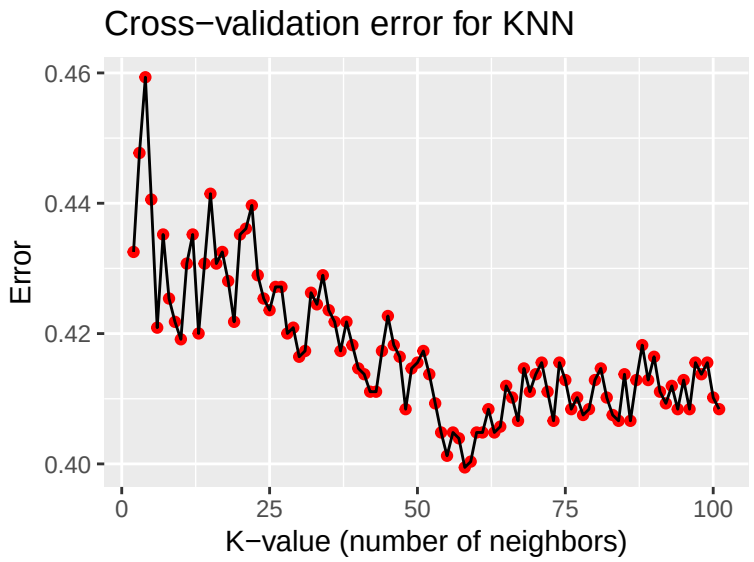
KNN

We start our analysis by using the KNN method. The KNN-method is sensitive to magnitudes, and we therefore scale all the features so they all weigh equally. Then we separate the data into training and testing data.

```r
# scaling data
scaled_data = wine
scaled_data[, 1:11] = scale(wine[, 1:11])

# separate into training and testing data
set.seed(2023)
# 70% of the sample size for training set
training_set_size <- floor(0.7 * nrow(scaled_data))
train_ind <- sample(seq_len(nrow(scaled_data)), size = training_set_size)
train = scaled_data[train_ind, ]
test = scaled_data[-train_ind, ]
train$quality = as.factor(train$quality)
test$quality = as.factor(test$quality)
```

We then use cross-validation do choose a optimal K-value. For a hundred values of K we calculate the error between the predicted values and the values from the training data. We choose the K-value with the smallest error.

```r
# cross validation for k-value
set.seed(3)
all_errors = rep(0, 100)  # vector to store the error for each k-value
dim_train <- dim(train)[1]  # how many different wines in training data
x <- 2:101  # x has 100 values

for (k_val in x) {
    KNN = knn.cv(train = train[, -12], cl = train$quality, k = k_val)  # knn-cross-validation function
    compare_table = table(KNN, train$quality)  # compares knn.cv result and the actual values from training data
    # the diagonal of compare_table is the amount of wines that were classefied
    # right.
    error = (dim_train - sum(diag(compare_table)))/dim_train  #calculating the error
    all_errors[k_val - 1] = error
}

xy_vals <- data.frame(X = x, Y = all_errors)  # make data frame with x and y values for plotting

plot <- ggplot(xy_vals, aes(x = X, y = Y)) + geom_point(color = "red") + geom_line() +
    ylab("Error") + xlab("K-value (number of neighbors)") + ggtitle("Cross-validation error for KNN")
plot
```




```
min_index <- which.min(all_errors)  # find index of k-value with smallest error
k_value <- x[min_index]
cat("K-value: ", k_value)
```

```
## K-value:  58
```

We see that the optimal K-value in our case is 58. Using this K-value we make our model using the `knn()` function. Then by making a confusion matrix we can calculate the accuracy of the prediction.

```
# Make model with found k-value
KNN_model <- knn(train = train[, -12], test = test[, -12], cl = train$quality, k = k_value)
confusion_KNN <- confusionMatrix(test$quality, KNN_model)
confusion_KNN
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   2   0   0   0
##          4   0   0  10   3   0   0
##          5   0   0 155  37   1   0
##          6   0   0  79 114   8   0
##          7   0   0   6  42  16   0
##          8   0   0   0   7   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5938
##                  95% CI : (0.5483, 0.638)
##     No Information Rate : 0.525
##     P-Value [Acc > NIR] : 0.001438
##
##                   Kappa : 0.3284
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity                NA       NA   0.6151   0.5616  0.64000       NA
## Specificity          0.995833  0.97292   0.8333   0.6859  0.89451  0.98542
## Pos Pred Value             NA       NA   0.8031   0.5672  0.25000       NA
```

```
## Neg Pred Value              NA       NA   0.6620   0.6810  0.97837       NA
## Prevalence           0.000000  0.00000   0.5250   0.4229  0.05208  0.00000
## Detection Rate       0.000000  0.00000   0.3229   0.2375  0.03333  0.00000
## Detection Prevalence 0.004167  0.02708   0.4021   0.4188  0.13333  0.01458
## Balanced Accuracy           NA       NA   0.7242   0.6237  0.76725       NA
```

We see that the accuracy of the prediction is only 59.38%. The prediction makes the most mistake by mixing the quality scores 5, 6, and 7. When the majority of the wines have a score of 5 or 6, it is hard for the model to differentiate between them since the variations between them are so small.

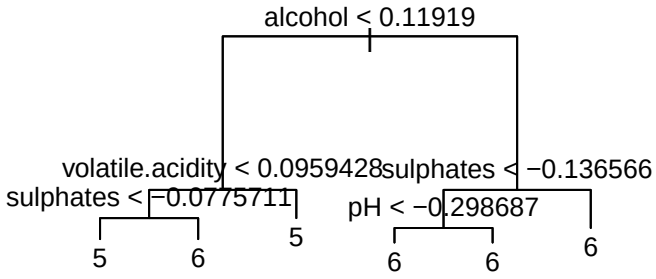
Decision trees

We now move on to another algorithm - decision trees. Our first tree is built below. Here the splitting criterion is chosen as the minimizing the cross entropy, by setting the parameter `split = "deviance"`.

```
# Building a classification tree using the tree() function
wine_tree <- tree(train$quality ~ ., data = train, split = "deviance")
summary(wine_tree)
```

```
##
## Classification tree:
## tree(formula = train$quality ~ ., data = train, split = "deviance")
## Variables actually used in tree construction:
## [1] "alcohol"          "volatile.acidity" "sulphates"        "pH"
## Number of terminal nodes:  6
## Residual mean deviance:  1.936 = 2155 / 1113
## Misclassification error rate: 0.4129 = 462 / 1119
```

```
plot(wine_tree, type = "proportional")
text(wine_tree, cex = 0.8)
```



The tree above shows why decision trees are so useful - given a wine and the relevant properties, one could quickly make a classification just by looking at the tree. But how accurate is the prediction?

```
# Calculate the accuracy of the first tree
tree_pred <- predict(wine_tree, test, type = "class")
confusion_matrix = table(test$quality, tree_pred)
confusion_matrix
```

```
##    tree_pred
##       3   4   5   6   7   8
##   3   0   0   0   2   0   0
##   4   0   0   7   6   0   0
##   5   0   0 144  49   0   0
##   6   0   0  74 127   0   0
##   7   0   0   8  56   0   0
##   8   0   0   0   7   0   0
```

```
tree_accuracy = sum(diag(confusion_matrix))/sum(confusion_matrix)
print(tree_accuracy)
```

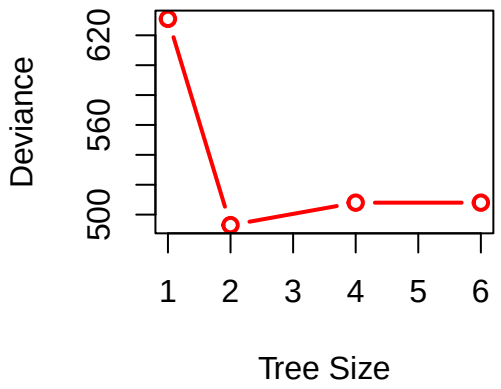
```
## [1] 0.5645833
```

We see that the tree above only predicts approximately 57% of wines correctly! This is not a very high prediction rate (even lower than for KNN, which was 59%).This might be because the data is not balanced (i.e. most of the quality ratings are 5 or 6). We also see that all of the terminal nodes to the right of the first split end up predicting the same class, namely 6. We therefore consider pruning the tree using cross validation (CV).

```
# Pruning the tree back using k-fold CV
set.seed(1)
wine_tree_cv = cv.tree(wine_tree, FUN = prune.misclass)  #CV
```

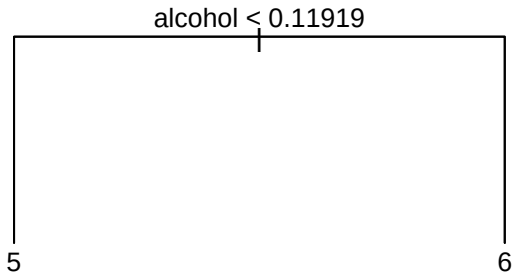
We use the misclassification error to prune the tree, by setting the parameter `FUN = prune.misclass`. The *cv.tree*() function performs a 10-fold CV by default. The results from the cross validation are shown in the following figure:

```
# Plotting the result of the CV
plot(wine_tree_cv$dev ~ wine_tree_cv$size, type = "b", lwd = 2, col = "red", xlab = "Tree Size",
    ylab = "Deviance")
```



We see that we get a reduction in deviance if we prune the tree back from 6 to 2 leaves.

```
# From the plot we should choose 2 terminal leaves/nodes
wine_tree_pruned <- prune.tree(wine_tree, best = 2)
plot(wine_tree_pruned, type = "proportional")
text(wine_tree_pruned, cex = 0.8)
```




```
# Calculate the accuracy of the pruned tree
pruned_tree_pred <- predict(wine_tree_pruned, test, type = "class")
confusion_matrix = table(test$quality, pruned_tree_pred)
pruned_tree_accuracy = sum(diag(confusion_matrix))/sum(confusion_matrix)
print(pruned_tree_accuracy)
```

```
## [1] 0.5333333
```

With the pruning we see that the accuracy of the model is slightly lowered to 54%. Interestingly, the only split is the predictor alcohol. However, we require a better model, and therefore we turn to random forests. Since we already have an indication that alcohol is a strong predictor from our previous findings, we expect the random forest model to perform better, since most of the trees from the bootstrapped samples would likely have alcohol as their root node and thus be highly correlated. Since we have a total of 11 predictors in our data set, we set `mtry` $= \sqrt{11} \approx 3$ in the `randomForest` function. This means that at each split, 3 predictors will be chosen randomly and considered in the split.

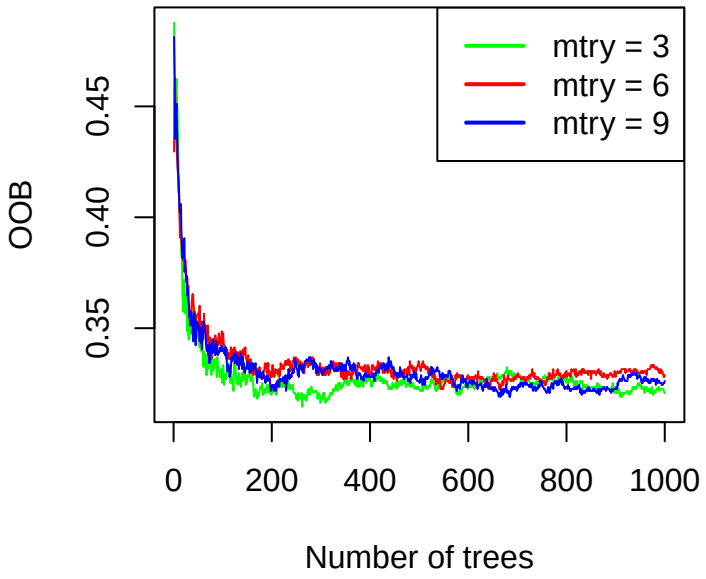
```
# Building a random forest We have 11 predictors, so for the random forest we
# should use sqrt(3) \approx 3 predictors each split (note that we are doing
# classification)
wine_rf = randomForest(train$quality ~ ., data = train, mtry = 3, ntree = 500, importance = TRUE)
# Calculate the accuracy of the random forest model
rf_pred = predict(wine_rf, test, type = "class")
confusion_matrix2 = table(test$quality, rf_pred)
rf_accuracy = sum(diag(confusion_matrix2))/sum(confusion_matrix2)
print(rf_accuracy)
```

```
## [1] 0.6854167
```

The random forest method outperforms the single decision tree with an accuracy of approximately 69%. This is a significant increase! Here we chose the parameters `mtry` $= 3$ and `ntree` $= 500$. We can investigate the parameters further by calculating the out-of-bag (OOB) error for three models with `mtry` $= 3, 6, 9$ as a function of number of trees. The results are shown in the figure below:

```
# Check different hyperparameters - number of trees and number of predictors in
# each split
set.seed(1)
model1 = randomForest(train$quality ~ ., data = train, mtry = 3, ntree = 1000)
model2 = randomForest(train$quality ~ ., data = train, mtry = 6, ntree = 1000)
model3 = randomForest(train$quality ~ ., data = train, mtry = 9, ntree = 1000)
```

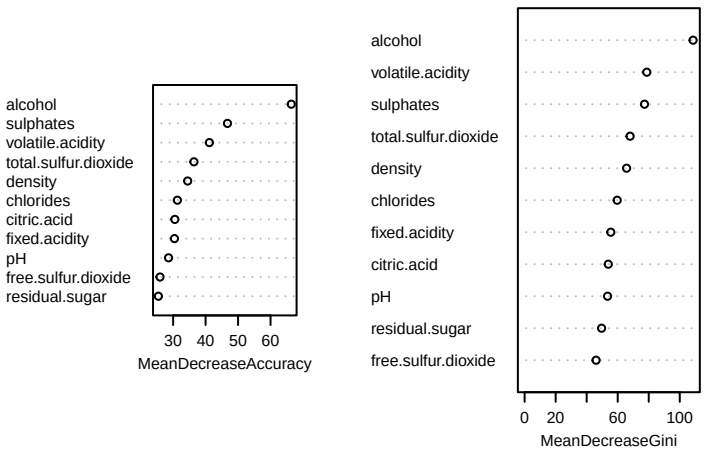
```
plot(1:1000, model1$err.rate[, 1], type = "l", col = "green", ylab = "OOB", xlab = "Number of trees")
lines(1:1000, model2$err.rate[, 1], col = "red")
lines(1:1000, model3$err.rate[, 1], col = "blue")
legend(x = "topright", legend = c("mtry = 3", "mtry = 6", "mtry = 9"), lty = c(1,
    1, 1), col = c("green", "red", "blue"), lwd = 2)
```



Using 3 predictors in each split seems to perform the best. All three models also seem to stabilize around 300-400 trees, so using 500 trees in our random forest models should be sufficient.

We can also interpret the predictor importance in the model by using the `varImpPlot()` function, which gives two plots: the left plot shows variable importance based on randomization, while the right plot is the mean decrease in the Gini index by splits of a predictor (decrease in node impurity).

```
# We can check the importance of each predictor using the varImpPlot() function
varImpPlot(wine_rf, main = "", cex = 0.5, fig.dim = c(20, 20))
```



We can see from the plots that the predictors alcohol, sulphates, volatile acidity, total.sulfur.dioxide and density seem to be the strongest predictors of quality.

We want to further measure our model fits by looking at the ROC curves and corresponding AUC score. We therefore convert our response variable "quality" to a binary variable where the class "bad" corresponds to wines with quality rating from 0 up to (and including) 6, and the class "good" corresponds to wines with quality rating from 6 up to 10.

```
# We assign the continuous response 'quality' a binary value 'good'/'bad'
wine <- read.csv("winequality-red.csv")
wine$quality <- cut(wine$quality, breaks = c(0, 6, 10), labels = c("bad", "good"),
    right = TRUE)
summary(wine$quality)  #Note the variable quality is changed
```

```
##  bad good
## 1382  217
```

```
class(wine$quality)
```

```
## [1] "factor"
```

```
train <- wine[train_ind, ]
test <- wine[-train_ind, ]
```

This allows us to calculate the ROC curve of the random forest model. The ROC curve is a plot of the true positive rate (sensitivity) versus the false positive rate (which is equivalent to 1 - specificity). We create a new random forest with the binary response, and check the accuracy of this model:

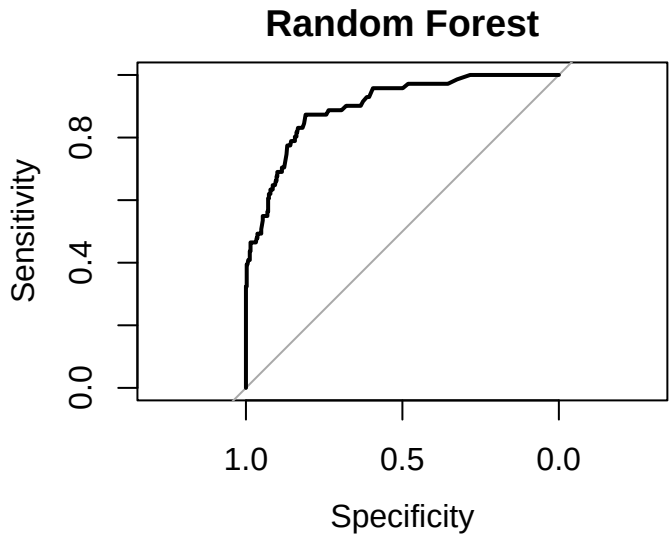
```
# We create a random forest again
set.seed(1)
wine_rf = randomForest(train$quality ~ ., data = train, mtry = 3, ntree = 500, importance = TRUE)

# Calculate the accuracy of the random forest model
rf_pred = predict(wine_rf, test, type = "class")
confusion_matrix2 = table(test$quality, rf_pred)
rf_accuracy = sum(diag(confusion_matrix2))/sum(confusion_matrix2)
print(rf_accuracy)  #Note the accuracy is higher
```

```
## [1] 0.9041667
```

The random forest model predicts the wines in our test data set to be either good or bad with an accuracy of 91%.

We can now calculate the ROC curve by the `roc` function from the pROC library. This is done by passing the argument `type = "prob"` when we make a prediction on our test data set, so we get the probability distributions of each prediction instead of a response. The probability distributions can then be used to find the ROC curve.

```
# Calculate the ROC curves with corresponding AUC
library(pROC)
rf_pred = predict(wine_rf, test, type = "prob")
rf_roc = roc(test$quality, rf_pred[, 2])
plot(rf_roc, main = "Random Forest")
```




```
rf_roc$auc
```

```
## Area under the curve: 0.9002
```

We see the AUC score of the ROC curve is approximately 0.9, which is very good. For comparison, we consider another model.

Logistic regression

Logistic regression is good for making predictions about the probability of an event occurring. In our case the probability of a wine being "good" or "bad".

We use the function `glm()` with the parameter `family = "binomial"` to create our logistic regression model. As before, we use the `predict()`function to test our model, and create a confusion matrix to show the accuracy of the model.

```
logModel <- glm(quality ~ ., family = "binomial", data = train)
summary(logModel)
```

```
##
## Call:
## glm(formula = quality ~ ., family = "binomial", data = train)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0347  -0.4075  -0.1983  -0.0983   3.2054
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)           1.938e+02  1.346e+02   1.439 0.150067
## fixed.acidity         2.616e-01  1.528e-01   1.712 0.086912 .
## volatile.acidity     -3.500e+00  1.013e+00  -3.456 0.000548 ***
## citric.acid           5.910e-02  1.064e+00   0.056 0.955712
## residual.sugar        1.908e-01  9.787e-02   1.950 0.051176 .
## chlorides            -1.017e+01  4.472e+00  -2.273 0.022999 *
## free.sulfur.dioxide   3.327e-02  1.646e-02   2.022 0.043174 *
## total.sulfur.dioxide -2.205e-02  7.027e-03  -3.137 0.001704 **
## density              -2.086e+02  1.375e+02  -1.517 0.129220
## pH                    3.702e-02  1.259e+00   0.029 0.976540
## sulphates             3.728e+00  6.892e-01   5.410 6.30e-08 ***
## alcohol               8.715e-01  1.643e-01   5.305 1.13e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 866.75  on 1118  degrees of freedom
## Residual deviance: 575.01  on 1107  degrees of freedom
## AIC: 599.01
##
## Number of Fisher Scoring iterations: 6
```

```
prediction <- predict(logModel, test, type = "response")
logPrediction = ifelse(prediction > 0.5, 1, 0)
logConf = table(test$quality, logPrediction)
logConf
```

```
##       logPrediction
##          0   1
##   bad  395  14
##   good  47  24
```

```
logAcc = sum(diag(logConf))/sum(logConf)
logAcc
```

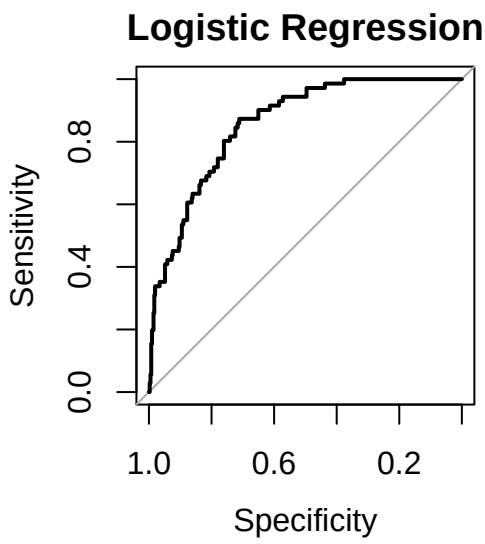
```
## [1] 0.8729167
```

When classifying the wines into "good" and "bad" the logistic regression model classifies 419 of the wines accurately, and 61 of the wines wrong, giving an accuracy of 87%.

We again see that volatile acidity, total sulfur dioxide, sulphates and alcohol are the most significant predictors at a level of 0.01.

Below the ROC and AUC are calculated.

```
logROC <- roc(test$quality, prediction)
plot(logROC, main = "Logistic Regression")
```




```
logROC$auc
```

```
## Area under the curve: 0.8568
```

The AUC has a value of 0.8568, making it a good result since we want a AUC value near 1. The random forest model achieved an AUC score of 0.9, so based on this it seems the better choice of model.

Summary

The KNN model seemed to not perform too well on our data, with a predictive accuracy of 59%. The decision tree also proved to be a bad model for our data set, with an accuracy of only 54%. When using the "quality" response as a non-binary variable, the random forest method gave slightly better results with an accuracy of 69%.

For the logistic regression method, it turned out to be the next best model with an accuracy of 87% and AUC = 0.86.

In our case the use of random forest with binary response variable proved to be the best model with an accuracy of 91% and AUC = 0.9.

Common to the two methods logistic regression and random forests, is the fact that the top 4 most significant predictors are the same - alcohol, sulphates, volatile acidity and total sulfur dioxide. So even though the methods are different and consider the predictors as different levels of importance, they both agree on the top 4 most significant to the wine quality.

Sources

- P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.