Wine Type and Quality Prediction

.

1 Introduction

This project uses the wine quality data set from UCI repository of white and red wines. Two datasets were created, using red and white wine samples. There are much more normal wines than excellent or poor ones.

Input variables used based on physicochemical tests:

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. alcohol

Output variable based on sensory data:

12. quality (score between 0 and 10)

2 Methods and Analysis

Predicting wine quality and identifying the wine type, red or white, are classification problems. This is because the desired outcomes are grouped in classes or categories. Quality has 10 classes, numbered from 0 to 9, and wine type has two classes, red and white. The methodology used in this document apply machine learning classification techniques.

The evaluation of machine learning algorithms consists in comparing the predicted value with the actual outcome. The confusion matrix displays the predicted and observed values in a table and provides additional statistics summary. There are other metrics that go beyond the scope of this document.

2.1.1 Confusion Matrix

The confusion matrix is a simple table with the cross tabulation of the predicted values with the actual observed values.

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

All values are in absolute numbers of observations and predictions, so for example the *True Positive* is the number of predicted values that are exactly the same as the actual values.

The meaning of the values in the table are:

True Positive (TP): Predicted *positive* for an actual *positive* value.

True Negative (TN): Predicted *negative* for an actual *negative* value.

False Positive (FN) or Type 1 Error: Predicted *positive* for an actual *negative* value.

False Negative (FN) or Type 2 Error: Predicted *negative* for an actual *positive* value.

Some useful statistic metrics can be calculated from the confusion matrix.

Accuracy: the proportion of correct predictions for both *positive* and *negative* outcomes, i.e., the ability to correctly predict a *positive* and *negative*. High accuracy with a large difference in the number of positives and negatives becomes less meaningful, since the algorithm loses the ability to predict the less common class. In this case, other metrics complements the analysis.

$$\text{Accuracy} = TP+TN/TP+FP+TN+FN$$

Sensitivity: the proportion of *positive* values when they are actually *positive*, i.e., the ability to predict *positive* values.

$$\text{Sensitivity} = TP/TP+FN$$

Specificity: is the probability of a predicted *negative* value conditioned to a *negative* outcome.

$$\text{Specificity} = Pr(Y\hat{}=\text{Negative}|Y=\text{Negative})$$

In other words, specificity is the proportion of *negative* values when they are actually *negative*, i.e. the ability to predict *negative* values.

$$\text{Specificity} = TN/TN+FP$$

Prevalence: how often the *positive* value appears in the sample. Low prevalence may lead to statistically incorrect conclusions.

$$\text{Prevalence} = TP+FN/TP+FP+TN+FN$$

Precision: is the probability of an actual *positive* occurs conditioned to a predicted *positive* result.

$$\text{Precision} = Pr(Y=\text{Positive}|Y\hat{}=\text{Positive})$$

Precision can be written as the proportion of *positive* values that are actually *positive*.

$$Precision=TP/TP+FP$$

Recall: is the same as sensitivity and is the probability of a predicted *positive* value conditioned to an actual *positive* value.

$$Recall=Sensitivity=Pr(Y\hat{}=Positive|Y=Positive)$$

$$Recall=TP/TP+FN$$

2.1.2 F1 Score

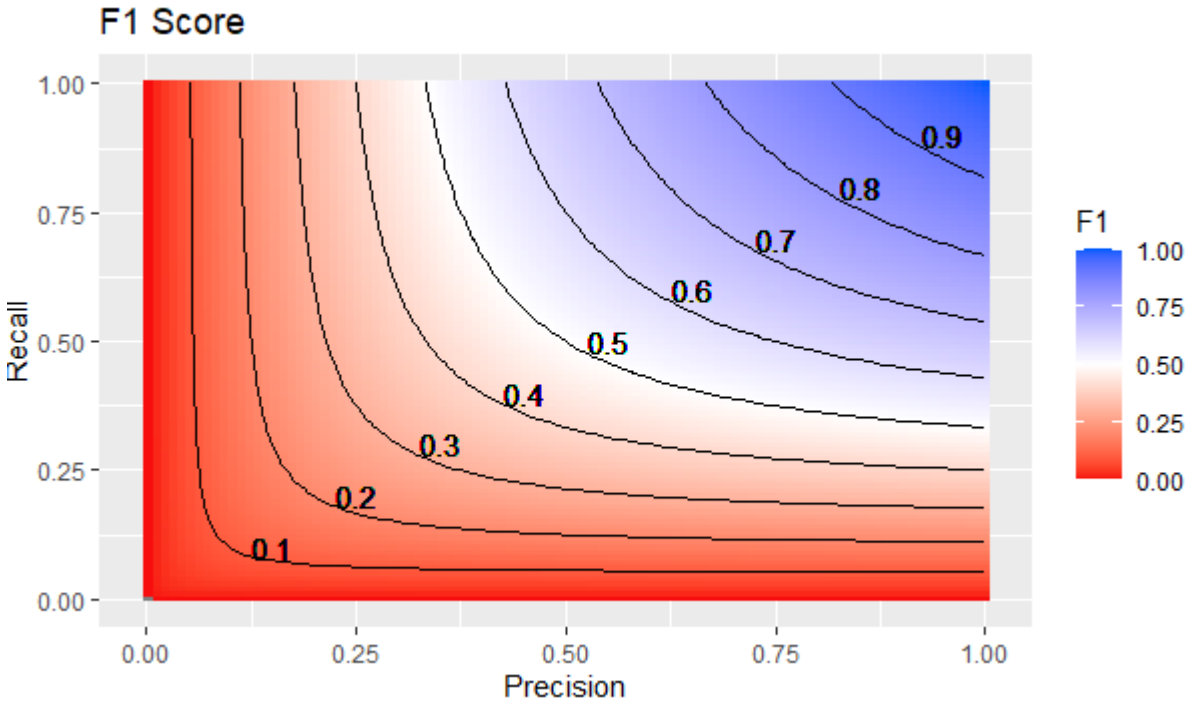
The F1 score is a measure of accuracy for classification problems with binary outcome. It is computed as the harmonic mean between precision and recall.

$$F1\ Score=1/1/2(1/recall+1/precision)$$

The formula can be simplified to:

$$F1\ Score=2\times(precision\times recall)/(precision+recall)$$

Since precision and recall vary from 0 to 1, we can make a plot of both values with the F1 score variation in color gradient. The black lines indicate the same F1 score values for different precision and recall combinations.



2.1.3 ROC and AUC

The Receiver Operator Characteristic (ROC) curve is the plot of True Positive Rate (TPR) and False Positive Rate (FPR) at different classification thresholds. It is commonly used in the evaluation of predictions of classification outcomes in machine learning.

$$TPR=Sensitivity=TP/TP+FN$$

$$FPR=1-Specificity=FP/TN+FP$$

AUC	Classification
0.90 - 1	excellent (A)
0.80 - 0.90	good (B)
0.70 - 0.80	fair (C)
0.60 - 0.70	poor (D)
0.50 - 0.60	fail (F)

The area under the ROC curve (AUC) measures the probability that a model will rank a positive value higher than a negative one.

AUC ranges from 0 to 1. A model with 100% wrong predictions has an AUC of 0, a model with 100% right predictions has AUC of 1 and a model with random prediction has an AUC of 0.5.

A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system.

2.2 Data Preparation

In this section, we download and import the datasets in R. In the UCI repository there is one file for each wine type and one file with the dataset description, total of three files.

The type column is added to each dataset to define the type of wine we want to predict with the factor variable type. This is variable used as the outcome that will be predicted.

Then, both datasets are combined in the wine dataset.

```
# Set number of significant digits
options(digits = 3)
# The 'load_lib' function installs and loads
# a vector of libraries
load_lib <- function(libs) {
  sapply(libs, function(lib) {

    # Load the package. If it doesn't exists, install and load.
    if(!require(lib, character.only = TRUE)) {

      # Install the package
      install.packages(lib)

      # Load the package
      library(lib, character.only = TRUE)
      }
})}
```

```r
# Load the libraries used in this section
libs <- c("tidyverse", "icesTAF", "readr",
          "lubridate", "caret")

load_lib(libs)

# Download the datasets from UCI repository
if(!dir.exists("data")) mkdir("data")
if(!file.exists("data/winequality-red.csv"))
  download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red
.csv", "data/winequality-red.csv")
if(!file.exists("data/winequality-white.csv"))
  download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-wh
ite.csv", "data/winequality-white.csv")
if(!file.exists("data/winequality.names"))
  download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality.na
mes", "data/winequality.names")

# Import the datasets.
# 'red' is the red wine dataset
# 'white' is the white wine dataset.
red   <- read_delim("data/winequality-red.csv",
                    delim = ";",
                    locale = locale(decimal_mark = ".",
                                    grouping_mark = ","),
                    col_names = TRUE)
white <- read_delim("data/winequality-white.csv",
                    delim = ";",
                    locale = locale(decimal_mark = ".",
                                    grouping_mark = ","),
                    col_names = TRUE)

# Set column names
cnames <- c("fixed_acidity", "volatile_acidity", "citric_acid",
            "residual_sugar", "chlorides", "free_sulfur_dioxide",
            "total_sulfur_dioxide", "density", "pH",
            "sulphates", "alcohol", "quality")

# Columns used for prediction are all columns
# except 'quality'.
xcol <- c("fixed_acidity", "volatile_acidity", "citric_acid",
          "residual_sugar", "chlorides", "free_sulfur_dioxide",
          "total_sulfur_dioxide", "density", "pH",
          "sulphates", "alcohol")

colnames(red)   <- cnames
colnames(white) <- cnames

# Add the column 'type' to define the type of wine
red   <- mutate(red,   type = "red")
white <- mutate(white, type = "white")

# Join 'red' and 'white' datasets
wine <- rbind(red, white)
wine <- mutate(wine,
               quality = as.factor(quality),
               type = as.factor(type))
levels(wine$quality) <- paste0("Q", levels(wine$quality))
```

Now we split the dataset in two parts, one for training and one for testing. The model building and tuning is done in the training set, and then we use the test set to predict new values and evaluate the results. In this project the model won't be evaluated in another dataset.

We create a pair of training and testing sets for each prediction problem. train_set and test_set will be used for identifying wine type, and train_set_red and test_set_red will be used for red wine quality.

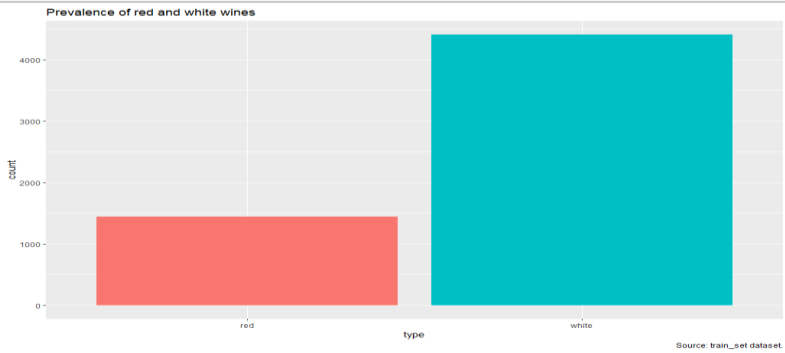
```
# Test set will be 10% of the entire dataset
set.seed(2020, sample.kind = "Rounding")
test_index <- createDataPartition(y = wine$type,
                                  times = 1,
                                  p = 0.1,
                                  list = FALSE)

# Train and test sets for wine type
train_set <- wine[-test_index,]
test_set  <- wine[test_index,]

# Train and test sets for red wine quality
train_set_red <- train_set[which(train_set$type == "red"),]
test_set_red  <- test_set[which(test_set$type == "red"),]

train_set_red$quality <- factor(train_set_red$quality)
test_set_red$quality  <- factor(test_set_red$quality)
```

2.3 Data Exploration and Visualization

2.3.1 Red and White Prevalence

The goal is to identify red and white wines, so we check the distribution of both types. The summary above provides the figures; now let's check the relative amount.

```
# Distribution of red and white wines
ggplot(data = train_set) +
  geom_bar(aes(type, fill = type)) +
  labs(title = "Prevalence of red and white wines",
       caption = "Source: train_set dataset.") +
  theme(legend.position = 'none')
```



The prevalence of red wine in this dataset is 24.6% and of white wine is 75.4%.

Let's compare this prevalence against the total production of red and white wines. This dataset was created in 2009 and there's no information about the date of each sample, so I'm assuming the most recent information is from 2008.

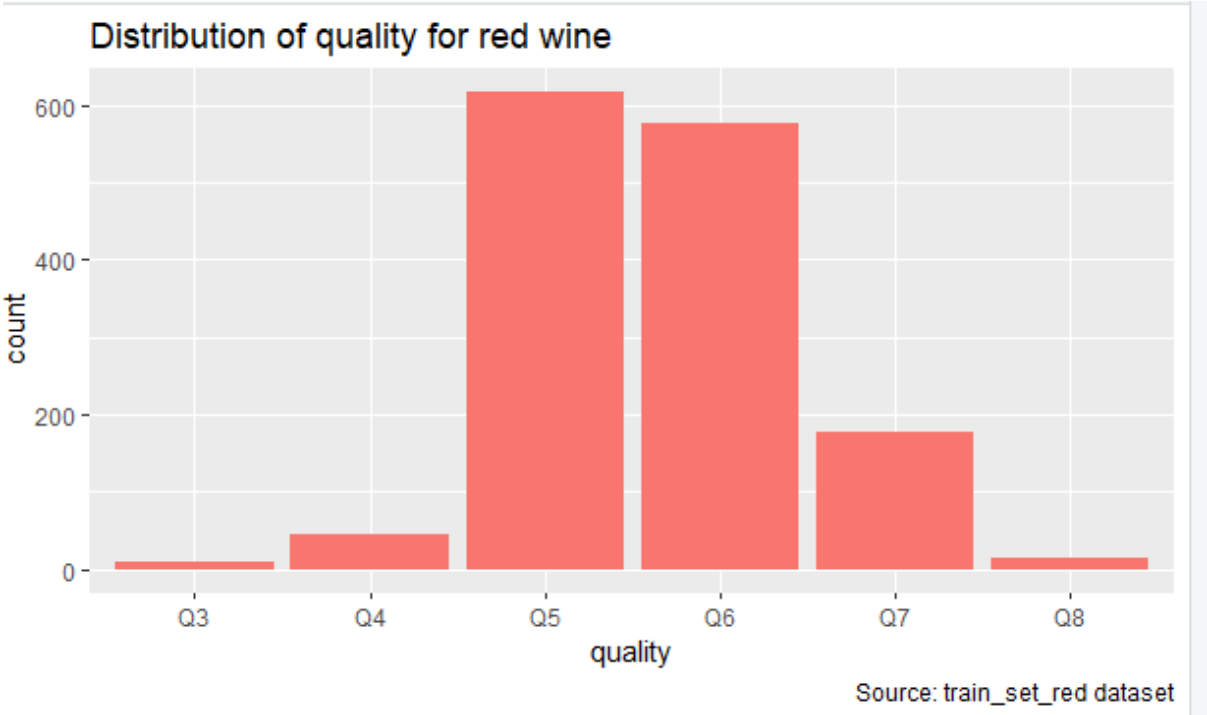
The commission of vinho verde producers provides annual statistics of production for each wine type.

We import the statistics and calculate the prevalence of red wine production.

The prevalence of the production of red wine is higher than observed in the dataset. There is no information available about production by brand or grape variety.

2.3.2 Quality distribution

Before predicting the quality of red wines, we need to understand the quality distribution. Notice that the distribution is highly disproportional: there are many more wines with qualities 5 and 6 than 3,4 and 8.



2.3.3 Variable importance

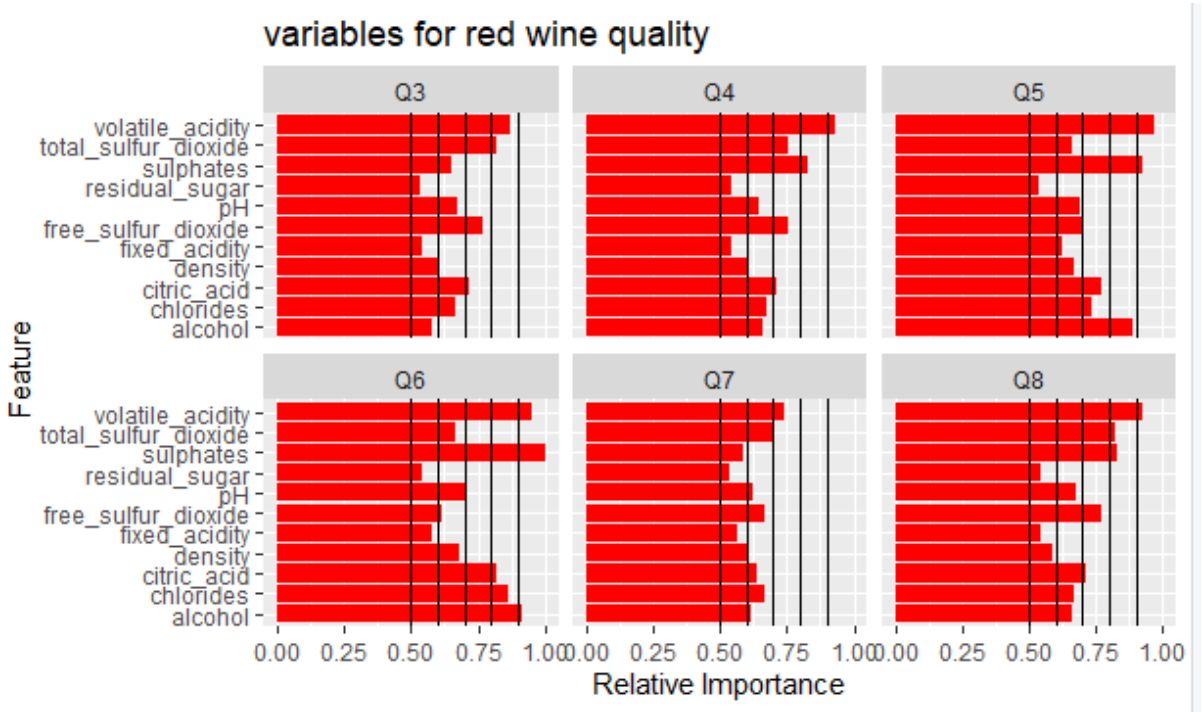
There are 11 variables, or features, that may be used as predictors, but not all of them may be useful in predicting the wine type. We use the R function filterVarImp to estimate the importance of each variable. For two class outcomes, such as predicting whether the wine is red or white, the area under the ROC curve is used as a measure of variable importance[7].

Total sulfur dioxide, chlorides and volatile acidity are the most important for predicting wine type, while alcohol, citric acid and residual sugar are the least important.

For multi-class outcomes, such as predicting the wine quality in a scale of 1 to 9, the problem is decomposed into all pair-wise problems and the area under the curve is calculated

for each class pair (i.e. class 1 vs. class 2, class 2 vs. class 3 etc.). For a specific class, the maximum area under the curve across the relevant pair-wise AUC's is used as the variable importance measure.

Volatile acidity, sulphates and alcohol have very high AUC values for red wine quality.



In practice, the distribution of features isn't so clear, overlapping for different outcomes. In this case, machine learning models use two or more features for prediction. Also, highly correlated features provide low predictive power.

So, let's look at each feature distribution for each wine type and quality of red wines, then we check the correlation among the features.

2.3.4 Data visualization

It's easier to identify the distribution using data visualization. First, we load some packages and define a function used in this section.

```
# Install and load the libraries used for visualization
# The 'load_lib' function was defined earlier.
load_lib(c("gridExtra", "ggridges", "ggplot2",
           "gtable", "grid"))


# The 'grid_arrange_shared_legend' function creates a grid of
# plots with one legend for all plots.
# Reference: Baptiste Auguié - 2019
# https://cran.r-project.org/web/packages/egg/vignettes/Ecosystem.html
grid_arrange_shared_legend <-
  function(...,
           ncol = length(list(...)),
           nrow = 1,
           position = c("bottom", "right")) {
```

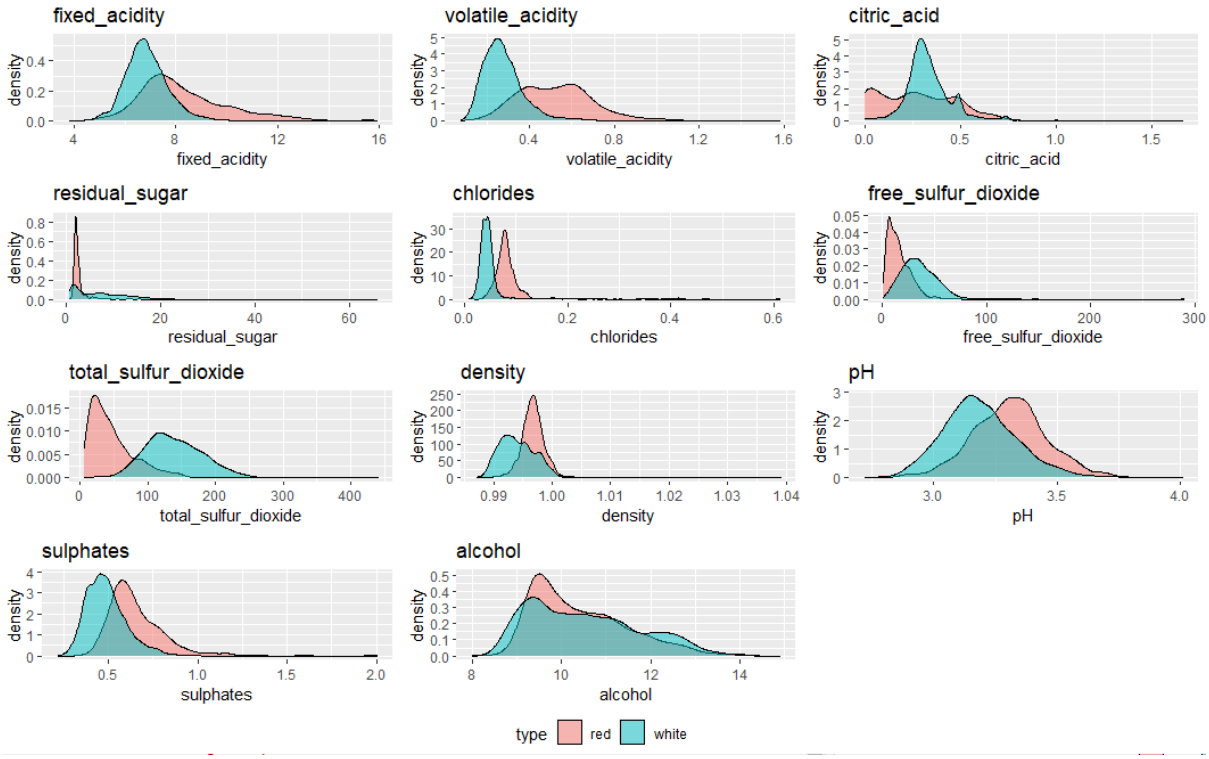
```
  plots <- list(...)
  position <- match.arg(position)
  g <-
    ggplotGrob(plots[[1]] + theme(legend.position = position))$grobs
  legend <- g[[which(sapply(g, function(x)
    x$name) == "guide-box")]]
  lheight <- sum(legend$height)
  lwidth <- sum(legend$width)
  gl <- lapply(plots, function(x)
    x + theme(legend.position = "none"))
  gl <- c(gl, ncol = ncol, nrow = nrow)

  combined <- switch(
    position,
    "bottom" = arrangeGrob(
      do.call(arrangeGrob, gl),
      legend,
      ncol = 1,
      heights = unit.c(unit(1, "npc") - lheight, lheight)
    ),
    "right" = arrangeGrob(
      do.call(arrangeGrob, gl),
      legend,
      ncol = 2,
      widths = unit.c(unit(1, "npc") - lwidth, lwidth)
    )
  )

  grid.newpage()
  grid.draw(combined)

  # return gtable invisibly
  invisible(combined)

}
```

2.3.5 Density plots

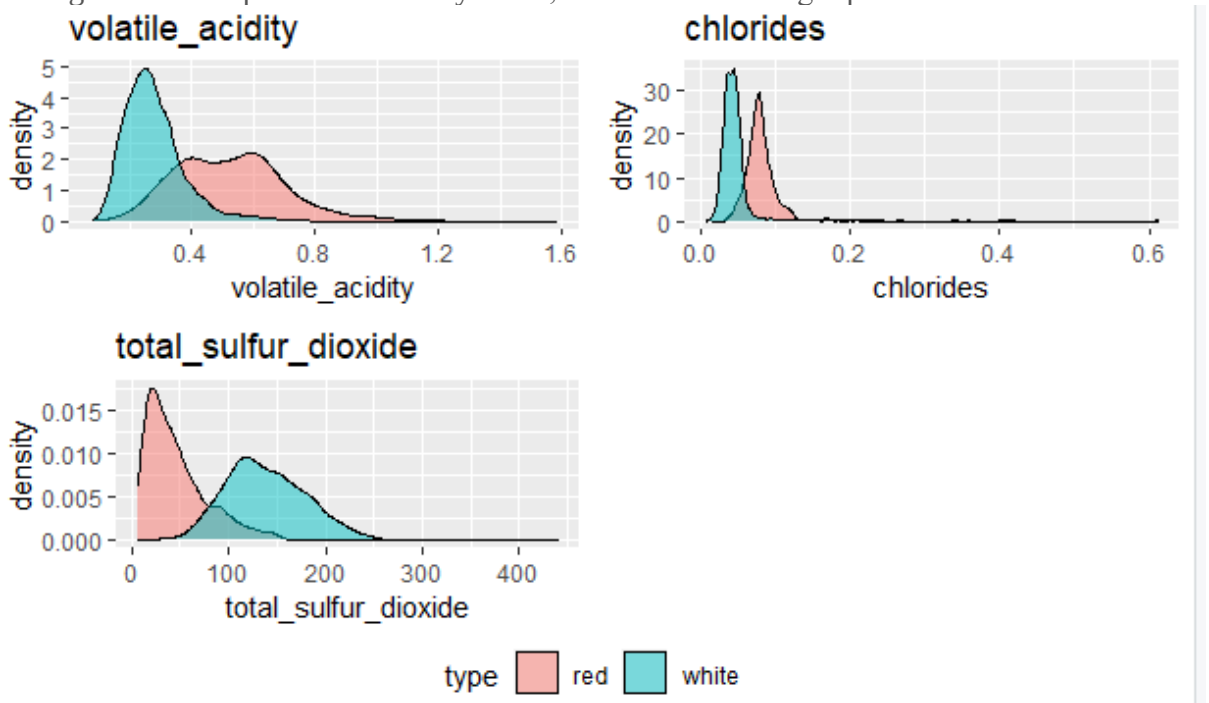
The density plots show each predictor distribution grouped by wine type. Features with little or no overlap make better prediction with higher accuracy, sensitivity and specificity.

The distribution of alcohol is very similar for red and white wines, meaning that alcohol is a bad predictor for wine type. On the other hand, the distribution of volatile acidity, chlorides and total sulfur dioxide is clearly shifted for both types. Sulphates, citric acid and pH have large overlap. The distribution on the density plots confirms the findings of variable importance.

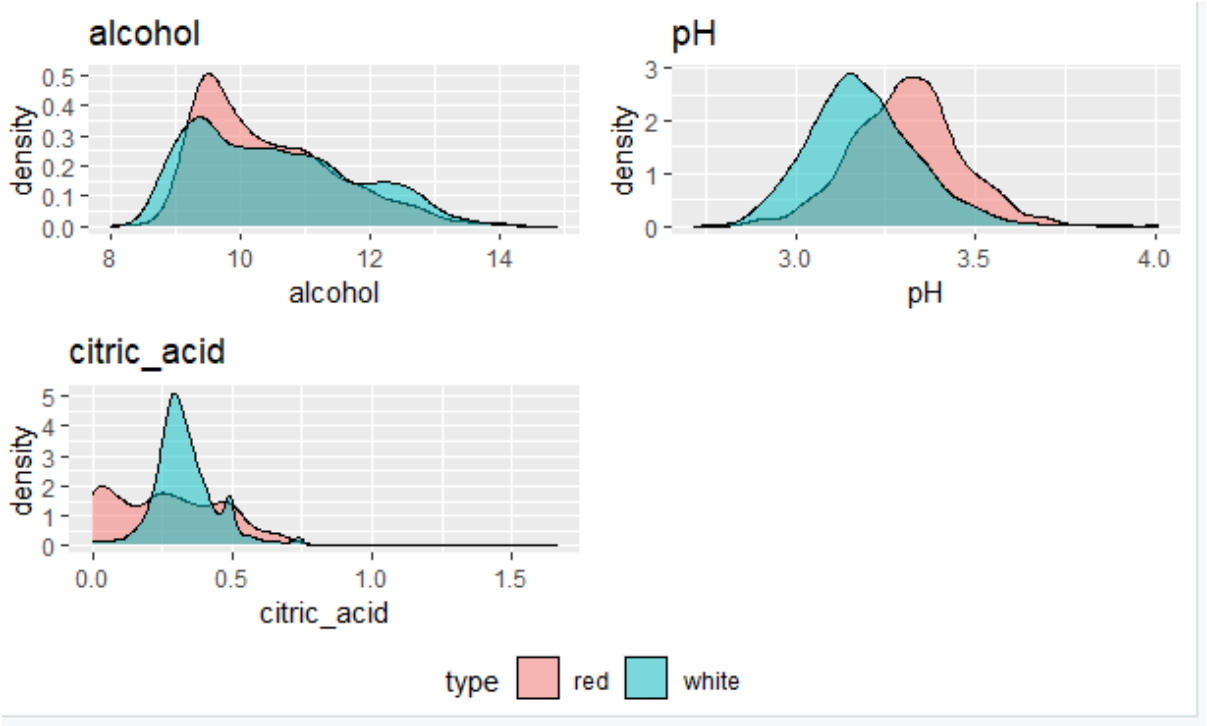
```
# Density grid
dens_grid <- lapply(xcol, FUN=function(var) {
  # Build the plots
  ggplot(train_set) +
    geom_density(aes_string(x = var, fill = "type"), alpha = 0.5) +
    ggtitle(var)
})
do.call(grid_arrange_shared_legend, args=c(dens_grid, nrow = 4, ncol = 3))
```



We notice that volatile acid, chlorides and total sulfur dioxide have distinct peaks for red and white wines. These predictors may help distinguish both wine types. These three variables distributions are shifted to the left side of the xx axis, indicating possible outliers in the right side. The plot above is very small, so let's make a larger plot for them.



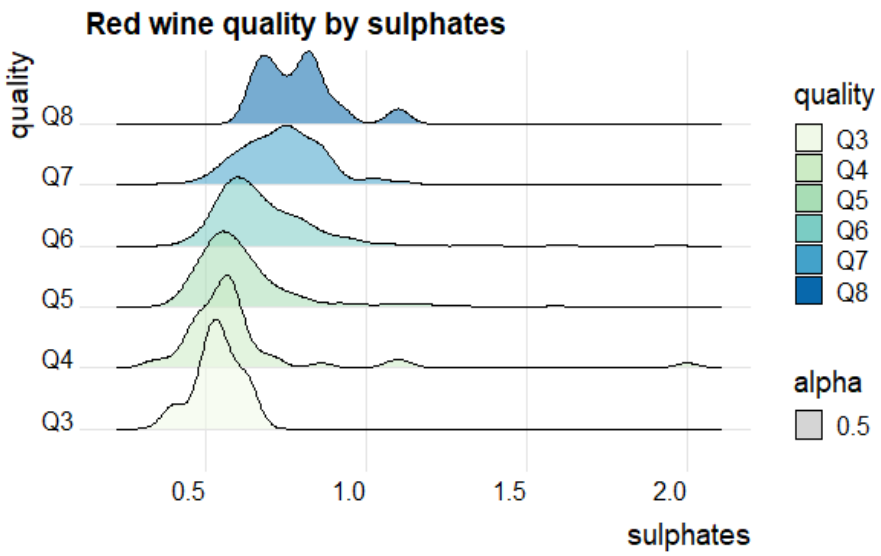
The distribution of alcohol, pH and citric acid overlaps in both wine types.

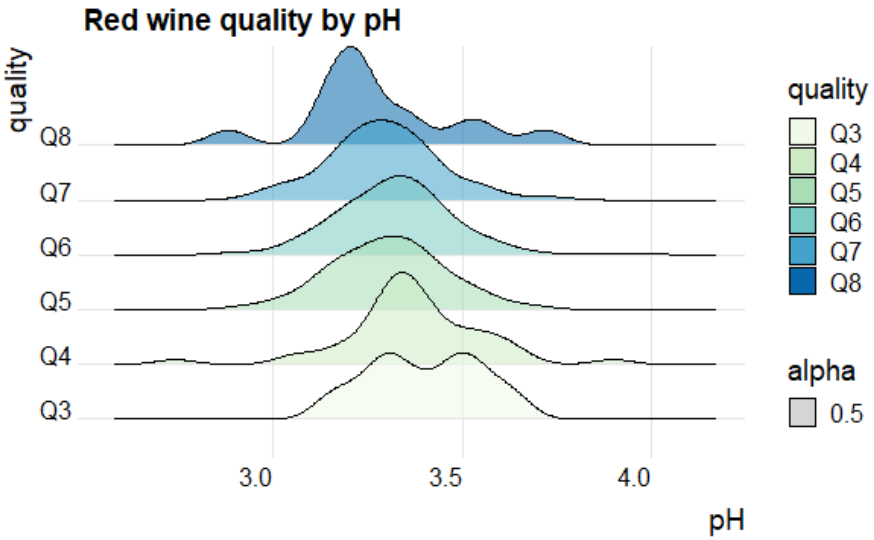


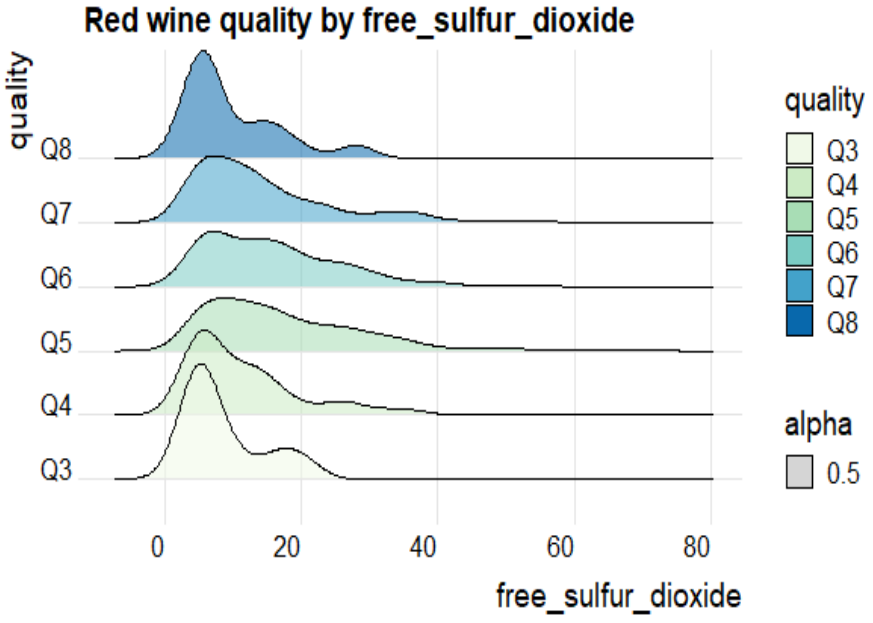
2.3.6 Density ridge plot

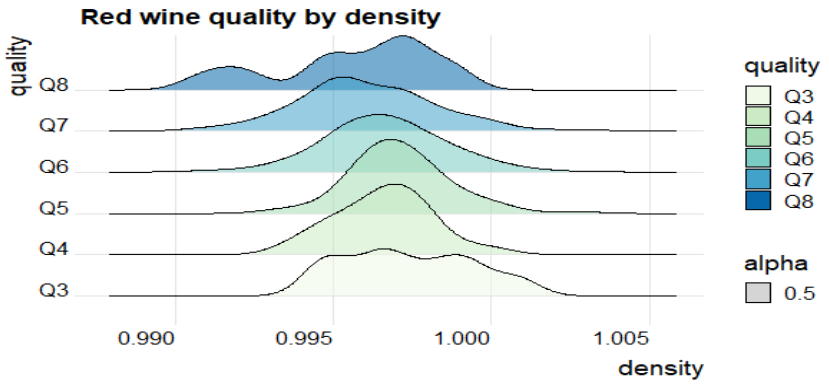
The density plots overlaps the distribution of both wine types. For the visualization of the distribution of features by quality, we use density ridge plots. In every feature the distribution overlap among quality levels, meaning that no single feature alone is able to predict quality.

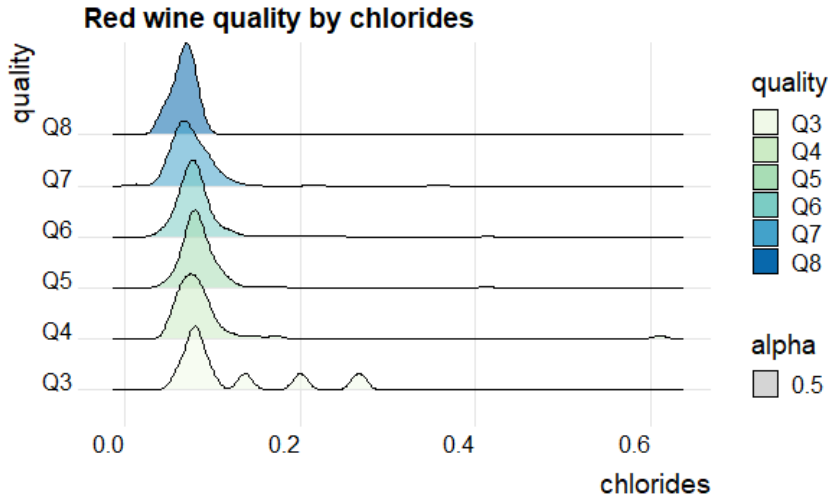
```
# Density ridge plots
lapply(xcol, FUN=function(var) {

  train_set_red %>%
    ggplot(data = ., aes_string(x = var,
                                y = "quality",
                                fill = "quality",
                                alpha = 0.5)) +
    geom_density_ridges() +
    theme_ridges() +
    theme(axis.text.x = element_text(hjust = 1)) +
    scale_fill_brewer(palette = 4) +
    ggtitle(paste0("Red wine quality by ", var))
})
```

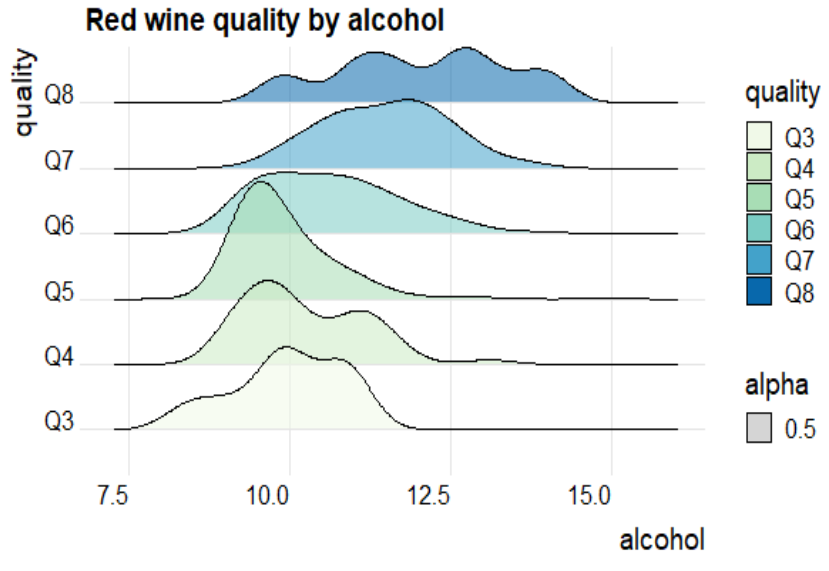

Red wine quality by sulphates
quality
Q8
Q7
Q6
Q5
Q4
Q3
0.5
1.0
1.5
2.0
sulphates
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

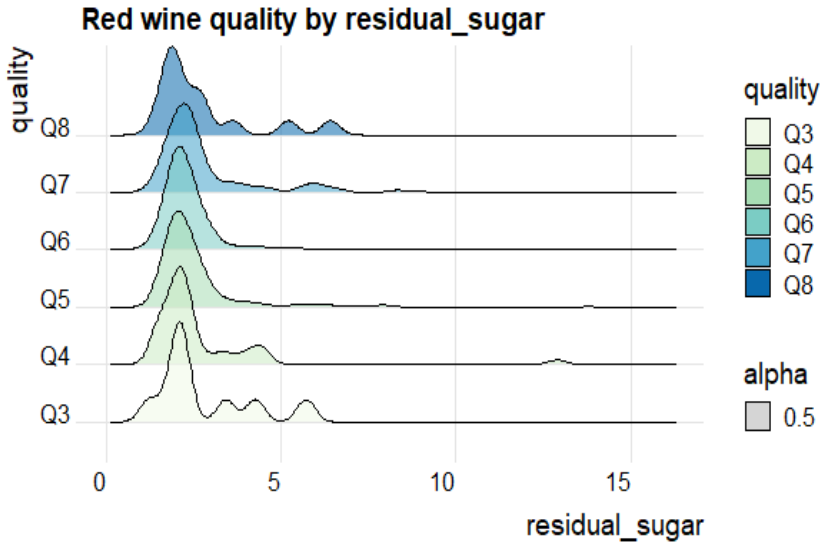

Red wine quality by pH
quality
Q8
Q7
Q6
Q5
Q4
Q3
3.0
3.5
4.0
pH
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

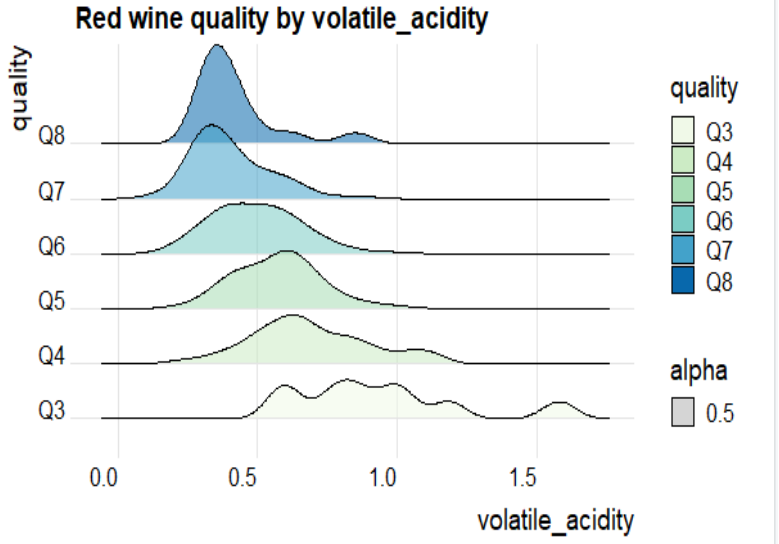

Red wine quality by free_sulfur_dioxide
quality
Q8
Q7
Q6
Q5
Q4
Q3
0
20
40
60
80
free_sulfur_dioxide
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

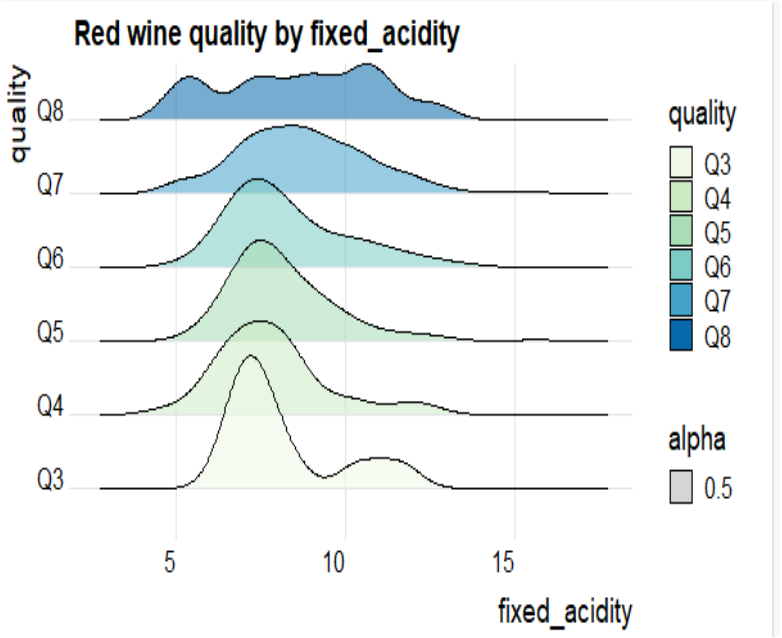

Red wine quality by density
quality
Q8
Q7
Q6
Q5
Q4
Q3
0.990
0.995
1.000
1.005
density
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

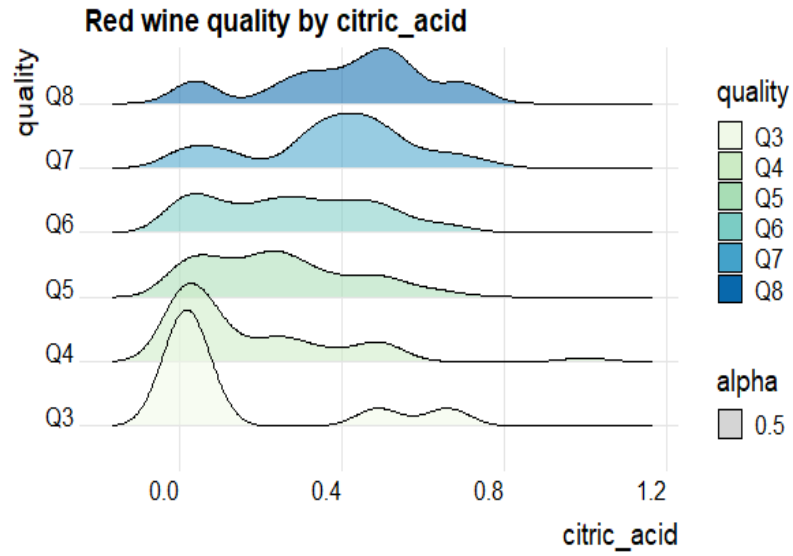

Red wine quality by chlorides
quality
Q8
Q7
Q6
Q5
Q4
Q3
0.0
0.2
0.4
0.6
chlorides
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

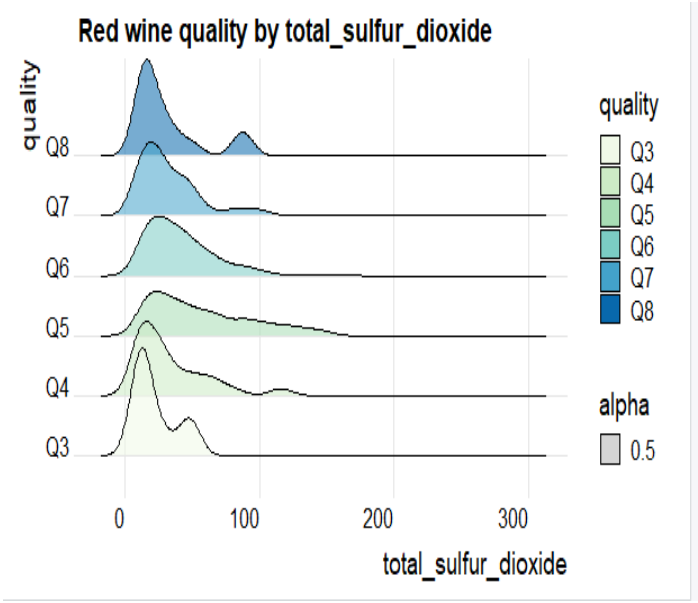

Red wine quality by alcohol
quality
Q8
Q7
Q6
Q5
Q4
Q3
7.5
10.0
12.5
15.0
alcohol
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5


Red wine quality by residual_sugar
quality
Q8
Q7
Q6
Q5
Q4
Q3
0
5
10
15
residual_sugar
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5


Red wine quality by volatile_acidity
quality
Q8
Q7
Q6
Q5
Q4
Q3
0.0
0.5
1.0
1.5
volatile_acidity
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5


Red wine quality by fixed_acidity
quality
Q8
Q7
Q6
Q5
Q4
Q3
5
10
15
fixed_acidity
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

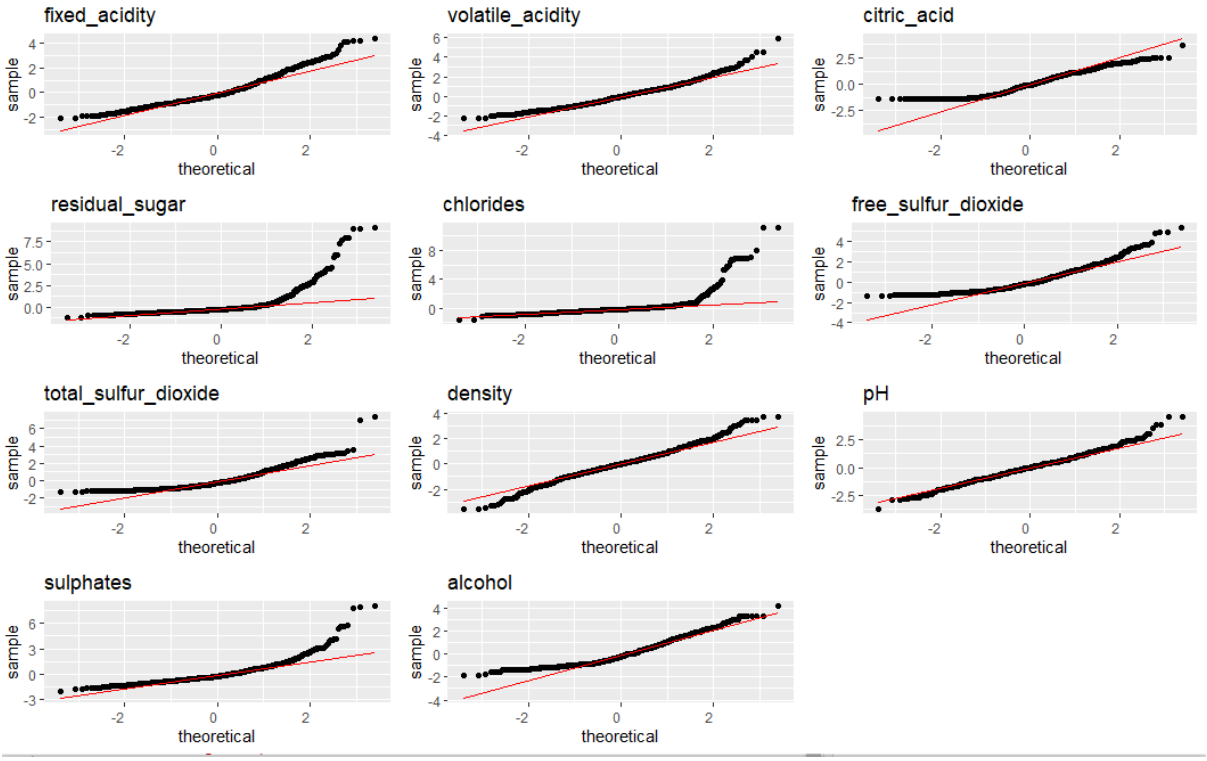




The distribution overlaps for all quality levels and all predictors. Grouping quality levels makes small improvement.

2.3.7 QQ plot

The QQ plots compare the feature distribution with the normal distribution. All variables are close to the normal distribution within two standard deviations of the mean. Some machine learning algorithms, such as LDA and QDA, assume the predictors are normally distributed.



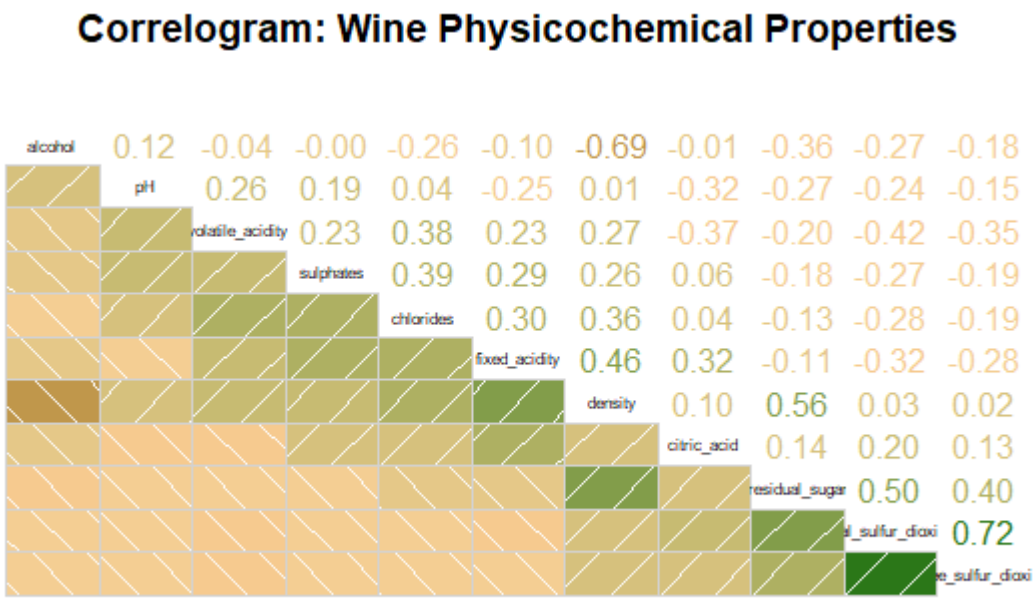
2.3.8 Correlation

The exploration done so far shows that volatile acid, chlorides and total sulfur dioxide are good candidates as wine type predictors. The prediction may be lower than expected if any two of these variables are highly correlated.

The correlogram shows the correlation of all predictors in pairs. The diagonal contains the variable names. The correlation of two variables are in the intersection of the variables rows and columns. The color intensity of the boxes and numbers indicate the correlation degree: strong colors indicate high correlation, whereas light colors indicate low correlation.

In this analysis, we assume that low correlation is between -0.5 and 0.5, and high correlation otherwise. Most colors in the correlogram are light and the correlations are low.

```
# Load the "corrgram" package to draw a correlogram
load_lib("corrgram")

# Draw a correlogram
corrgram(train_set[,xcol], order=TRUE,
        lower.panel = panel.shade,
        upper.panel = panel.cor,
        text.panel  = panel.txt,
        main = "Correlogram: Wine Physicochemical Properties",
        col.regions=colorRampPalette(c("darkgoldenrod4", "burlywood1",
                                       "darkkhaki", "darkgreen")))
```



2.4 Modeling

2.4.1 Single Predictor

Since the distributions of total sulfur dioxide, chlorides and volatile acidity stratified by wine type have small overlapping areas, we can find a cutoff value that maximizes the conditional probability.

$$Pr(Y=k|X=x)$$

Then, we combine the results in a single prediction, using majority of votes.

2.4.2 Cross Validation

The original dataset is partitioned in the training set used to train the model, and the test set used to predict the values with the trained model. Cross validation partitions the training set in the same way and performs the training and prediction several times. Then, the result with the best RMSE, accuracy, AUC or the chosen metric is selected. This process can be used in conjunction to tuning parameters, for example picking the best kk number of neighbours in knn.

2.4.3 Ensemble

Combining the results of several predictions may improve prediction accuracy. This method is called ensemble and consists in selecting the most common class for a given set of observations.

For example, suppose the predictions of wine type for three models as illustrated in the table below. The ensemble is just the majority of votes of the combined models.

Generally, the final result provides better accuracy than the best model.

.

Model 1	Model 2	Model 3	Ensemble
red	red	red	red
red	white	red	red
white	red	white	white

3 Results

This section presents the modelling results and discusses the model performance. For the most part of this section, the models are applied to predicting wine type, allowing the comparison of different models. The prediction of red wine quality is performed in the Predicting quality section.

Formula used in this section.

```
# Formula used in predictions
fml <- as.formula(paste("type", "~",
                        paste(xcol, collapse=' + ')))
```

3.1 Single Predictor

Volatile acidity, chlorides and total sulfur dioxide have the lowest overlapping area in the density plots of wine type. In this model, we calculate accuracy for several cutoff values in the distribution range, and pick the parameters with highest accuracy.

This process is repeated for each of these three predictors, then the result is combined in a single prediction. The final combination has superior performance over any individual predictor.

```
# Create a list with variable names and cutoff decision rule.
# If the predicted value is lower than the cutoff value, the first color
# is chosen, otherwise the second. To understand this, look at the
# density plots in data visualization.
type_var <- list( c("white", "red"), c("white", "red"), c("red", "white"))
names(type_var) <- c("volatile_acidity", "chlorides", "total_sulfur_dioxide")

# Create an empty results table. The first row
# contains NAs and will be removed after the predictions.
type_results <<- data.frame(Feature = NA,
                            Accuracy = NA,
                            Sensitivity = NA,
                            Specificity = NA,
                            stringsAsFactors = FALSE)

# Prediction function
```

```r
preds <- sapply(1:length(type_var), function(x){

    # Get the variable name
    var <- names(type_var[x])

    # Cutoff value is the distribution range divided by 500
    cutoff <- seq(min(train_set[,var]),
                  max(train_set[,var]),
                  length.out = 500)

    # Calculate accuracy
    acc <- map_dbl(cutoff, function(y){
      type <- ifelse(train_set[,var] < y, type_var[[x]][1],
                     type_var[[x]][2]) %>%
        factor(levels = levels(train_set$type))

      # Accuracy
      mean(type == train_set$type)
      })

    # Build the accuracy vs cutoff curve
    acc_plot <- data.frame(cutoff = cutoff, Accuracy = acc) %>%
      ggplot(aes(x = cutoff, y = Accuracy)) +
        geom_point() +
        ggtitle(paste0("Accuracy curve for ", var))

    # Print the plot
    print(acc_plot)

    # Predict new values in the test set
    # The model uses the cutoff value with the best accuracy.
    max_cutoff <- cutoff[which.max(acc)]
    y_hat <- ifelse(test_set[,var] < max_cutoff,
                    type_var[[x]][1], type_var[[x]][2]) %>%
      factor(levels = levels(test_set$type))

    # Calculate accuracy, specificity and sensitivity
    acc <- max(acc)
    sens <- sensitivity(y_hat, test_set$type)
    spec <- specificity(y_hat, test_set$type)

    # Update results table
    type_results <<- rbind(type_results,
                           data.frame(Feature = names(type_var[x]),
                                      Accuracy = acc,
                                      Sensitivity = sens,
                                      Specificity = spec,
                                      stringsAsFactors = FALSE))

    # The prediction will be used in the ensemble
    return(y_hat)
})

# Remove first row with NA
type_results <- type_results[2:nrow(type_results),]

# Combine the results using majority of votes
y_hat_ens <-as_factor(data.frame(preds) %>%
                        mutate(x = as.numeric(preds[,1] == "red") +
                               as.numeric(preds[,2] == "red") +
```

```
                        as.numeric(preds[,3]  == "red"),
                      y_hat = ifelse(x >=2, "red", "white")) %>%
                pull(y_hat))

# Update results table
type_results <<- rbind(type_results,
                data.frame(Feature = "Ensemble",
                        Accuracy = mean(y_hat_ens == test_set$type),
                        Sensitivity = sensitivity(y_hat_ens, test_set$type),
                        Specificity = specificity(y_hat_ens, test_set$type),
                        stringsAsFactors = FALSE))
# Show the results table
as_hux(type_results,
       add_colnames = TRUE) %>%
  # Format header
  set_bold(row = 1, col = everywhere, value = TRUE)        %>%
  set_top_border(row = 1, col = everywhere, value = 1)     %>%
  set_bottom_border(row = c(1,5), col = everywhere, value = 1)  %>%
  # Format cells
  set_align(row = 1:4, col = 2, value = 'right')                %>%
  # Format numbers
  set_number_format(row = everywhere, col = 2:4, value = 3)     %>%
  # Format table
  set_caption("Superior Performance for Combined Predictions")  %>%
  set_position(value = "center")
```

3.2 Linear Regression

Linear regression approximates the multidimensional space of predictors XX and outcomes YY into a function. The outcome, which is wine type, is categorical, so we need to convert to number before building the function. Here, we assign 1 to red wine and 0 to white wine.

The predicted values are also numeric, so we need to convert back to categories.

Again, we use only the three main features to predict wine type: total sulfur dioxide, chlorides and volatile acidity.

```
# Train the linear regression model
fit_lm <- train_set %>%
  # Convert the outcome to numeric
  mutate(type = ifelse(type == "red", 1, 0)) %>%
  # Fit the model
  lm(type ~ total_sulfur_dioxide + chlorides + volatile_acidity, data = .)

# Predict
p_hat_lm <- predict(fit_lm, newdata = test_set)

# Convert the predicted value to factor
y_hat_lm <- factor(ifelse(p_hat_lm > 0.5, "red", "white"))

# Evaluate the results
caret::confusionMatrix(y_hat_lm, test_set$type)
## Confusion Matrix and Statistics
##
##          Reference
## Prediction red white
```

```
##      red   151     7
##      white   9   483
##
##               Accuracy : 0.975
##                 95% CI : (0.96, 0.986)
##    No Information Rate : 0.754
##    P-Value [Acc > NIR] : <2e-16
##
##                  Kappa : 0.933
##
##  Mcnemar's Test P-Value : 0.803
##
##             Sensitivity : 0.944
##             Specificity : 0.986
##          Pos Pred Value : 0.956
##          Neg Pred Value : 0.982
##              Prevalence : 0.246
##          Detection Rate : 0.232
##    Detection Prevalence : 0.243
##       Balanced Accuracy : 0.965
##
##        'Positive' Class : red
```

3.3 KNN

Now we use all features to predict wine type.

```
# Train
fit_knn <- knn3(formula = fml, data = train_set, k = 5)

# Predict
y_knn <- predict(object = fit_knn,
                 newdata = test_set,
                 type ="class")

# Compare the results: confusion matrix
caret::confusionMatrix(data = y_knn,
                       reference = test_set$type,
                       positive = "red")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction red white
##      red   145    15
##      white  15   475
##
##               Accuracy : 0.954
##                 95% CI : (0.935, 0.969)
##    No Information Rate : 0.754
##    P-Value [Acc > NIR] : <2e-16
##
##                  Kappa : 0.876
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.906
##             Specificity : 0.969
```

```
##          Pos Pred Value : 0.906
##          Neg Pred Value : 0.969
##              Prevalence : 0.246
##          Detection Rate : 0.223
##    Detection Prevalence : 0.246
##       Balanced Accuracy : 0.938
##
##        'Positive' Class : red
##
```

```
# F1 score
F_meas(data = y_knn, reference = test_set$type)
```

```
## [1] 0.906
```

3.4 Linear Discriminant Analysis – LDA

LDA uses all features to predict wine type.

```
load_lib("MASS")
# Train the model
fit_lda <- lda(formula = fml, data = train_set)

# Predict
y_lda <- predict(object = fit_lda, newdata = test_set)

# Compare the results: confusion matrix
caret::confusionMatrix(data = y_lda[[1]],
                       reference = test_set$type,
                       positive = "red")
```

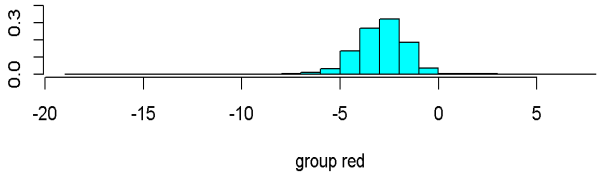
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction red white
##      red   158     0
##      white   2   490
##
##               Accuracy : 0.997
##                 95% CI : (0.989, 1)
##    No Information Rate : 0.754
##    P-Value [Acc > NIR] : <2e-16
##
##                  Kappa : 0.992
##
##  Mcnemar's Test P-Value : 0.48
##
##             Sensitivity : 0.988
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 0.996
##              Prevalence : 0.246
##          Detection Rate : 0.243
##    Detection Prevalence : 0.243
##       Balanced Accuracy : 0.994
##
##        'Positive' Class : red
##
```

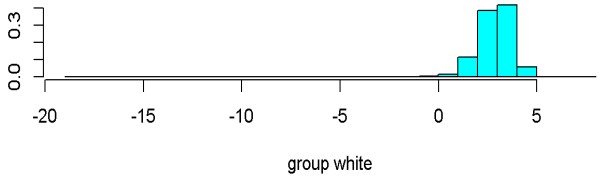
```
# F1 score
```

```
F_meas(data = y_lda[[1]], reference = test_set$type)
## [1] 0.994
```

LDA is able to create clear distinct groups for red and white wines.

```
# Plot the result
plot(fit_lda)
```





3.5 Quadratic Discriminant Analysis - QDA

QDA uses all features to predict wine type.

```
load_lib(c("MASS", "scales"))
# Train the model
fit_qda <- qda(formula = fml, data = train_set)

# Predict
y_qda <- predict(object = fit_qda, newdata = test_set)

# Compare the results: confusion matrix
caret::confusionMatrix(data = y_qda[[1]],
                       reference = test_set$type,
                       positive = "red")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction red white
##      red   158     4
##      white   2   486
##
##               Accuracy : 0.991
##                 95% CI : (0.98, 0.997)
##    No Information Rate : 0.754
##    P-Value [Acc > NIR] : <2e-16
##
##                  Kappa : 0.975
```

```
##
##  Mcnemar's Test P-Value : 0.683
##
##             Sensitivity : 0.988
##             Specificity : 0.992
##          Pos Pred Value : 0.975
##          Neg Pred Value : 0.996
##              Prevalence : 0.246
##          Detection Rate : 0.243
##    Detection Prevalence : 0.249
##       Balanced Accuracy : 0.990
##
##        'Positive' Class : red
##
```

```
# F1 score
F_meas(data = y_qda[[1]], reference = test_set$type)
```

```
## [1] 0.981
```

3.6 Prediction results

Random forest and LDA have best performance. Although the single predictor model and linear regression models used just three features as predictors, they have better performance than knn that used all features.

```
##              Model Accuracy Sensitivity Specificity
## 1  Single predictor    96.9%       89.4%       99.4%
## 2 Linear Regression    97.5%       94.4%       98.6%
## 3              Knn    95.4%       90.6%       96.9%
## 4  Regression trees    98.8%       96.2%       99.6%
## 5    Random forest    99.7%       98.8%      100.0%
## 6              LDA    99.7%       98.8%      100.0%
## 7              QDA    99.1%       98.8%       99.2%
```

3.7 Cross Validation and Ensemble

All models used up to now run on a single partition of the training set. The result is near perfect for identifying wine type. Now we apply cross validation on 11 classification algorithms and then combine the results.

```
# Now we are going to do several things:
# 1. train 10 classification models,
# 2. make the predictions for each model
# 3. calculate some statistics and store in the 'results' table
# 4. plot the ROC and precision-recall curves
# Then, we're going to plot the values in the 'results' table
# and make the ensemble of all models together.

# Load the packages used in this section
# Package "pROC" creates ROC and precision-recall plots
load_lib(c("pROC", "plotROC"))

# Several machine learning libraries
load_lib(c("e1071", "dplyr", "fastAdaboost", "gam",
           "gbm", "import", "kernlab", "kknn", "klaR",
           "MASS", "mboost", "mgcv", "monmlp", "naivebayes", "nnet", "plyr",
```

```
          "ranger", "randomForest", "Rborist", "RSNNS", "wsrf"))


# Define models
models <- c("glm", "lda", "naive_bayes", "svmLinear", "rpart",
          "knn", "gamLoess", "multinom", "qda", "rf", "adaboost")

# We run cross validation in 10 folds, training with 90% of the data.
# We save the prediction to calculate the ROC and precision-recall curves
# and we use twoClassSummary to compute the sensitivity, specificity and
# area under the ROC curve
control <- trainControl(method = "cv", number = 10, p = .9,
                        summaryFunction = twoClassSummary,
                        classProbs = TRUE,
                        savePredictions = TRUE)

control <- trainControl(method = "cv", number = 10, p = .9,
                        classProbs = TRUE,
                        savePredictions = TRUE)

# Create 'results' table. The first row
# contains NAs and will be removed after
# the training
results <- tibble(Model = NA,
                  Accuracy = NA,
                  Sensitivity = NA,
                  Specificity = NA,
                  F1_Score = NA,
                  AUC = NA)
#------------------------------
# Start parallel processing
#------------------------------
# The 'train' function in the 'caret' package allows the use of
# parallel processing. Here we enable this before training the models.
# See this link for details:
# http://topepo.github.io/caret/parallel-processing.html
cores <- 4    # Number of CPU cores to use
# Load 'doParallel' package for parallel processing
load_lib("doParallel")
cl <- makePSOCKcluster(cores)
registerDoParallel(cl)
```

3.8 Train the models

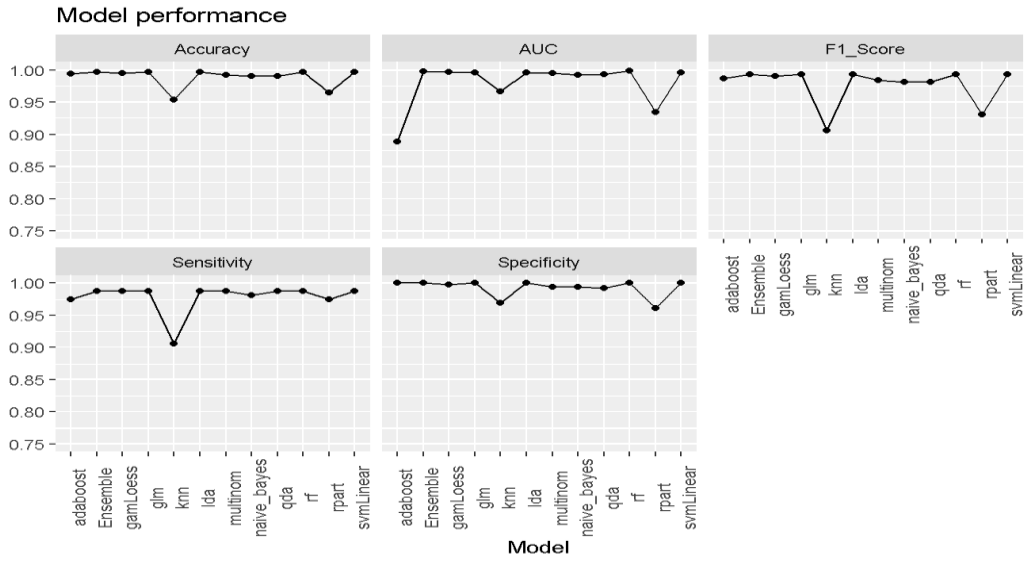
Here, we train the models, make predictions, calculate stats and store in ‘results’ table.

We use standard tuning parameters for all models except knn and random forest.

```
set.seed(1234, sample.kind = "Rounding")
# Formula used in predictions
fml <- as.formula(paste("type", "~",
                        paste(xcol, collapse=' + ')))

# Run predictions
preds <- sapply(models, function(model){

  if (model == "knn") {
    # knn use custom tuning parameters
    grid <- data.frame(k = seq(3, 50, 2))
```

```
    fit <- caret::train(form = fml,
                        method = model,
                        data = train_set,
                        trControl = control,
                        tuneGrid = grid)
  } else if (model == "rf") {
    # Random forest use custom tuning parameters
    grid <- data.frame(mtry = c(1, 2, 3, 4, 5, 10, 25, 50, 100))

    fit <- caret::train(form = fml,
                        method = "rf",
                        data = train_set,
                        trControl = control,
                        ntree = 150,
                        tuneGrid = grid,
                        nSamp = 5000)
  } else {
    # Other models use standard parameters (no tuning)
    fit <- caret::train(form = fml,
                        method = model,
                        data = train_set,
                        trControl = control)
  }

  # Predictions
  pred <- predict(object = fit, newdata = test_set)

  # Accuracy
  acc <- mean(pred == test_set$type)

  # Sensitivity
  sen <- sensitivity(data = pred,
                     reference = test_set$type,
                     positive = "red")
  # Specificity
  spe <- specificity(data = pred,
                     reference = test_set$type,
                     positive = "red")

  # F1 score
  f1 <- F_meas(data = factor(pred), reference = test_set$type)

  # AUC
  auc_val <- auc(fit$pred$obs, fit$pred$red)

  # Store stats in 'results' table
  results <<- rbind(results,
                    tibble(
                      Model = model,
                      Accuracy = acc,
                      Sensitivity = sen,
                      Specificity = spe,
                      AUC = auc_val,
                      F1_Score = f1))

  # The predictions will be used for ensemble
  return(pred)
})

# Remove the first row of 'results' that contains NAs
```

```
results <- results[2:(nrow(results)),]
  pivot_longer(cols = 2:6, names_to = "Metric", values_drop_na = TRUE) %>%
  ggplot(aes(x = Model, y = value, group = 1)) +
     geom_line() +
     geom_point() +
     # Y axis scale
     ylim(0.75, 1) +
     # Format labels
     ggtitle("Model performance") +
     ylab("") +
     theme(legend.position="none" ,
           axis.text.x = element_text(angle = 90)) +
     # Arrange in grid
    facet_wrap(~Metric)
```




```
results
## # A tibble: 12 x 6
##    Model       Accuracy Sensitivity Specificity F1_Score   AUC
##    <chr>          <dbl>       <dbl>       <dbl>    <dbl> <dbl>
##  1 glm            0.997       0.988       1        0.994 0.996
##  2 lda            0.997       0.988       1        0.994 0.996
##  3 naive_bayes    0.991       0.981       0.994    0.981 0.992
##  4 svmLinear      0.997       0.988       1        0.994 0.996
##  5 rpart          0.965       0.975       0.961    0.931 0.934
##  6 knn            0.954       0.906       0.969    0.906 0.967
##  7 gamLoess       0.995       0.988       0.998    0.991 0.997
##  8 multinom       0.992       0.988       0.994    0.984 0.995
##  9 qda            0.991       0.988       0.992    0.981 0.993
## 10 rf             0.997       0.988       1        0.994 0.999
## 11 adaboost       0.994       0.975       1        0.987 0.889
## 12 Ensemble       0.997       0.988       1        0.994 0.998
```

3.9 Predicting Quality

The prevalence of quality levels 3, 4 and 8 is very low in the dataset. There are just 5 samples of quality level 9 in the red and white datasets combined, and no sample in the train_set_red dataset.

We previously combined the levels into the low, medium and high quality categories in the train_set_red dataset. In this section, we use cross validation with ensemble to predict the combined levels.

The models glm, gamLoess, qda and adaboost don't work in this case.

```
set.seed(1234, sample.kind = "Rounding")
# Formula used in predictions
fml_qual <- as.formula(paste("quality2", "~",
                             paste(xcol, collapse=' + ')))

# Define models
#"glm",gamLoess, qda, adaboost
models <- c( "lda", "naive_bayes", "svmLinear", "rpart",
          "knn", "multinom", "rf")

# Create 'results' table. The first row
# contains NAs and will be removed after
# the training
quality_results <- tibble(Model = NA,
                          Quality = NA,
                          Accuracy = NA,
                          Sensitivity = NA,
                          Specificity = NA,
                          F1_Score = NA)

preds_qual <- sapply(models, function(model){

  print(model)
  if (model == "knn") {
    # knn use custom tuning parameters
    grid <- data.frame(k = seq(3, 50, 2))
    fit <- caret::train(form = fml_qual,
                        method = model,
                        data = train_set_red,
                        trControl = control,
                        tuneGrid = grid)
  } else if (model == "rf") {
    # Random forest use custom tuning parameters
    grid <- data.frame(mtry = c(1, 2, 3, 4, 5, 10, 25, 50, 100))

    fit <- caret::train(form = fml_qual,
                        method = "rf",
                        data = train_set_red,
                        trControl = control,
                        ntree = 150,
                        tuneGrid = grid,
                        nSamp = 5000)
  } else {
    # Other models use standard parameters (no tuning)
    fit <- caret::train(form = fml_qual,
                        method = model,
                        data = train_set_red,
                        trControl = control)
  }

  # Predictions
  pred <- predict(object = fit, newdata = test_set_red)
```

```
  # Accuracy
  acc <- mean(pred == test_set_red$quality2)

  # Sensitivity
  sen <- caret::confusionMatrix(pred,
                                test_set_red$quality2)$byClass["Sensitivity"]
  # Specificity
  spe <- caret::confusionMatrix(pred,
                                test_set_red$quality2)$byClass["Specificity"]

  # F1 score
  f1 <- caret::confusionMatrix(pred,
                               test_set_red$quality2)$byClass["F1"]

  # Store stats in 'results' table
  quality_results <<- rbind(quality_results,
                            tibble(Model = model,
                                   Quality = levels(test_set_red$quality2),
                                   Accuracy = acc,
                                   Sensitivity = sen,
                                   Specificity = spe,
                                   F1_Score = f1))

  # The predictions will be used for ensemble
  return(pred)
})

# Remove the first row of 'results' that contains NAs
quality_results <- quality_results[2:(nrow(quality_results)),]
```

Next, we combine the best results of all models in a single ensemble. The next table summarizes the results.

```
# Combine all models
# Use votes method to ensemble the predictions
votes <- data.frame(low    = rowSums(preds_qual =="low"),
                    medium = rowSums(preds_qual =="medium"),
                    high   = rowSums(preds_qual =="high"))

y_hat <- factor(sapply(1:nrow(votes), function(x)
  colnames(votes[which.max(votes[x,])])))

y_hat <- relevel(y_hat, "medium")

  # Accuracy
  acc <- caret::confusionMatrix(y_hat,
                                test_set_red$quality2)$overall["Accuracy"]

  # Sensitivity
  sen <- caret::confusionMatrix(y_hat,
                                test_set_red$quality2)$byClass["Sensitivity"]
  # Specificity
  spe <- caret::confusionMatrix(y_hat,
                                test_set_red$quality2)$byClass["Specificity"]

  # F1 score
  f1 <- caret::confusionMatrix(y_hat,
                               test_set_red$quality2)$byClass["F1"]
```

```
quality_results <<- rbind(quality_results,
                          tibble(Model = "Ensemble",
                                 Quality = levels(test_set_red$quality2),
                                 Accuracy = acc,
                                 Sensitivity = sen,
                                 Specificity = spe,
                                 F1_Score = f1))
#------------------------------
# Plot the 'results' table
#------------------------------
# Make a plot for each metric with the 3 quality levels
# The plots are grouped using the chunk options.

# Metric names
metrics <- names(quality_results)[3:6]

res <- sapply(metrics, function(var){

  # Plot stored in 'p'
  p <- quality_results %>%
    # Convert columns to lines
    pivot_longer(cols = 3:6, names_to = "Metric",
                 values_drop_na = TRUE) %>%

    pivot_wider(names_from = Quality) %>%
    filter(Metric == var) %>%

    ggplot(aes(x = Model, group = 1)) +
        # Draw lines
        geom_line(aes(y = low,     col = "low")) +
        geom_line(aes(y = medium,  col = "medium")) +
        geom_line(aes(y = high,    col = "high")) +
        # Draw points
        geom_point(aes(y = low,    col = "low")) +
        geom_point(aes(y = medium, col = "medium")) +
        geom_point(aes(y = high,   col = "high")) +
        # Format labels
        ggtitle(var) +
        ylab("Model") +
        theme(axis.text.x = element_text(angle = 90))

    # Show the plot
    print(p)
})
```

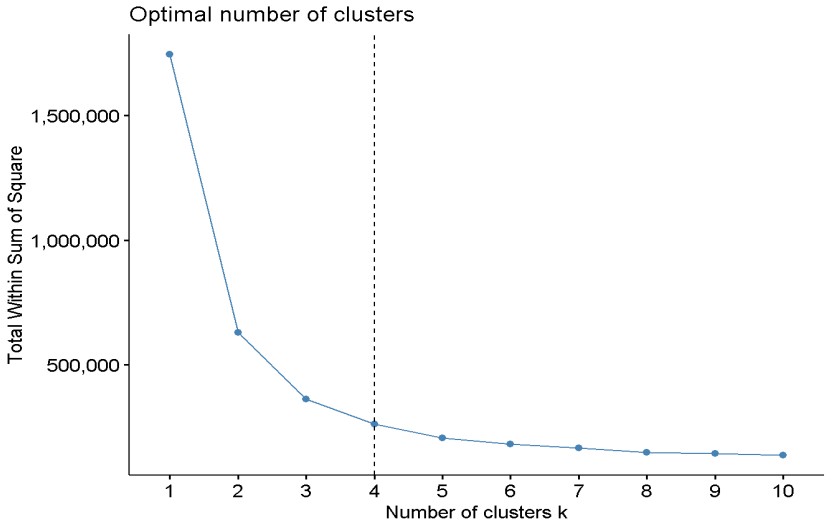
3.10 Clustering

In this section, we use unsupervised learning to identify possible clusters.

Partitioning methods, such as k-means clustering require the users to specify the number of clusters to be generated. The function fviz_nbclust from the factoextra package determines and visualize the optimal number of clusters using different methods: within cluster sums of squares, average silhouette and gap statistics.

The properties of red and white wines differ, so in this analysis we create clusters just for red wines.

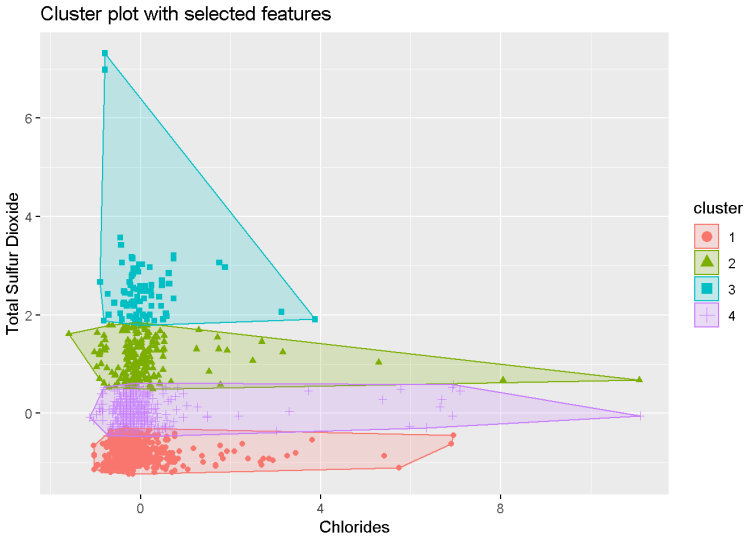
```
# k-means
# Reference:
# https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/

# Load 'factoextra' to visualize clusters
load_lib("factoextra")

# Determine and visualize the optimal number of clusters
# using total within sum of square (method = "wss")
train_set %>% filter(type == "red") %>% .[,xcol] %>%
  fviz_nbclust(x = ., FUNcluster = kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype =2) +
  scale_y_continuous(labels = comma)
```



The plot shows that the total sum of squares decreases slowly for more than 4 clusters. Selecting a larger value provides little improvement and increases the overlaps between the clusters.

The fviz_cluster function plots the clusters for the combination of two features. Here, we plot the clusters for total sulfur dioxide and chlorides.

```
# Plot the cluster
train_set %>% filter(type == "red") %>% .[,xcol] %>%
  fviz_cluster(object = k,
               choose.vars = c("chlorides", "total_sulfur_dioxide"),
               geom   = "point",
               repel  = TRUE,
               main   = "Cluster plot with selected features",
               xlab   = "Chlorides",
               ylab   = "Total Sulfur Dioxide")
```



4 Conclusion

This report uses two datasets of vinho verde wine to predict the wine type, red or white, and the quality based on physicochemical properties. Quality is a subjective measure, given by the average grade of three experts.

Before starting the predictions, the report makes a brief summary of model evaluation, explaining the most common metrics used in categorical problems in machine learning.

In data preparation, the datasets are downloaded and imported in R. In this phase, the training and testing sets are created and they will be used during the model building.

In data exploration and visualization, we look for features that may provide good prediction results. The best predictors have low distribution overlapping area and low correlation among them.

Modelling starts explaining very simple models and gradually moves to more complex ones. There's a brief explanation of the models used in this report. Other important machine learning concepts, such as ensemble and cross validation, are also discussed. Clustering is unsupervised method, which is included for illustrative purposes.

The results section presents the modelling results and discusses the model performance. The algorithms are used to predict wine type; there's a section for predicting quality and the last part demonstrates clustering.

The physicochemical properties of wine differ between red and white wines, but the difference is not so evident when evaluating red wine quality. Maybe there are other properties not considered in this dataset that are better indicators for quality.