# White Wines Quality

HarvardX: PH125.9x Capstone, CYO Project

*Andres Gomez*

*June, 2020*

## Contents

## Introduction

In this project, we are going to use the white wines data set (only the white one) which is part of the study done by [Cortez et al., 2009], and you can find at the UCI list of curated datasets.

The dataset has the following *input* variables (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

And one *output* variable (based on sensory data):

12. quality (score between 0 and 10)

Our **goal** is to build some models that *predict the white wines quality and help deciding whether to "buy" or "avoid" a wine. We will reduce the quality of wines to these two categories.* We will explain how we define these categories in the Modeling approach section.

First, in the **Method/Analysis** section we import the data and clean it, preparing it for analysis. We perform some exploratory data analysis, variables distributions, box-plots, correlation matrix, etc. Finally, we expose the *modeling approach* which will consist of two techniques: KNN and Random Forests, and the corresponding performance measures.

Secondly, in the **Results** section, we share the results of the models, interpreting the results and evaluating the models.

Third, in the **Conclusions** section, we summarize the work done, and share thoughts on possible improvements and next steps.

## Method/Analysis

In this section, we will do:

1. Data import and cleaning.
2. Exploratory data analysis.
3. The *modeling approach*, using KNN and Random Forest.

### Data import and cleaning.

We start by downloading the white wines data used by [Cortez et al., 2009], which you can find at the UCI list of curated datasets https://archive.ics.uci.edu/ml/datasets/Wine+Quality. Keep in mind that we are just using the white wines dataset, and not the red wines. We narrow our goal just for simplicity.

```r
# Download data to be used and create a DF (actually a spec_tbl_df type of file)
if(!require(readr)) install.packages("readr",  repos = "http://cran.us.r-project.org")


if(!file.exists("winequality-white.csv"))
download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white

wines <- read_delim("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")
# Rename columns to make it friendlier for R (at least for me)
colnames(wines) <- cnames
# Quality is numeric,
# let's create a variable "rating" that will be quality as factor with convenient format
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## package 'tidyverse' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\andre\AppData\Local\Temp\RtmpC4tLME\downloaded_packages
```

```r
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")

wines <- mutate(wines,
               rating = as.factor(quality))
levels(wines$rating) <- paste0("R_", levels(wines$rating))
```

### Exporatory data analysis

First of all, let's get some high level stats, on the complete dataset, to understand how the looks like. I avoid doing the exploratory data analysis on partitioned datasets because we might lose some outliers, for example.

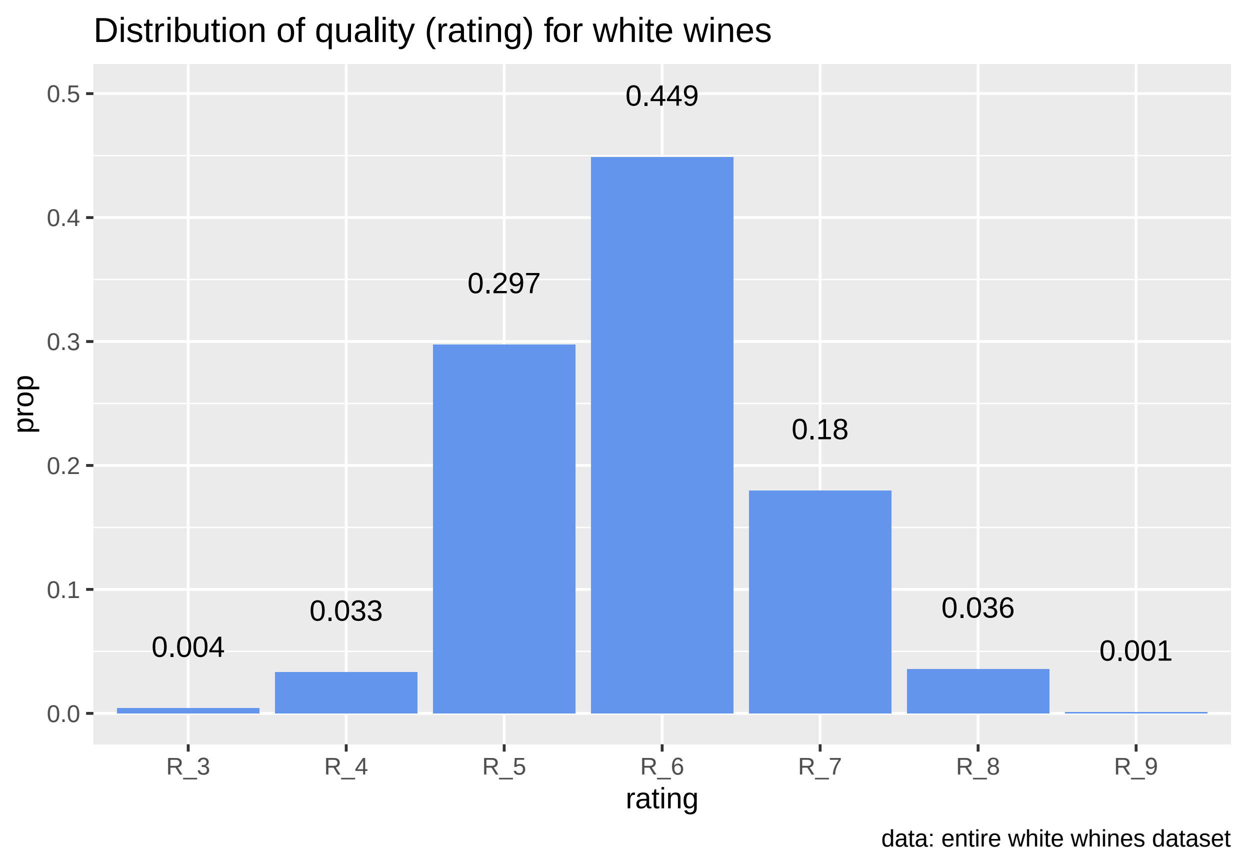
### Summary Statistics

```r
summary(wines)
```

```
##  fixed_acidity    volatile_acidity  citric_acid     residual_sugar
##  Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
##  1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700
##  Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200
##  Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6.391
##  3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900
##  Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800
```

```
##
##    chlorides       free_sulfur_dioxide total_sulfur_dioxide
##  Min.   :0.00900   Min.   :  2.00      Min.   :  9.0
##  1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:108.0
##  Median :0.04300   Median : 34.00      Median :134.0
##  Mean   :0.04577   Mean   : 35.31      Mean   :138.4
##  3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:167.0
##  Max.   :0.34600   Max.   :289.00      Max.   :440.0
##
##     density             pH          sulphates         alcohol
##  Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00
##  1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50
##  Median :0.9937   Median :3.180   Median :0.4700   Median :10.40
##  Mean   :0.9940   Mean   :3.188   Mean   :0.4898   Mean   :10.51
##  3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40
##  Max.   :1.0390   Max.   :3.820   Max.   :1.0800   Max.   :14.20
##
##     quality      rating
##  Min.   :3.000   R_3:  20
##  1st Qu.:5.000   R_4: 163
##  Median :6.000   R_5:1457
##  Mean   :5.878   R_6:2198
##  3rd Qu.:6.000   R_7: 880
##  Max.   :9.000   R_8: 175
##                  R_9:   5
```

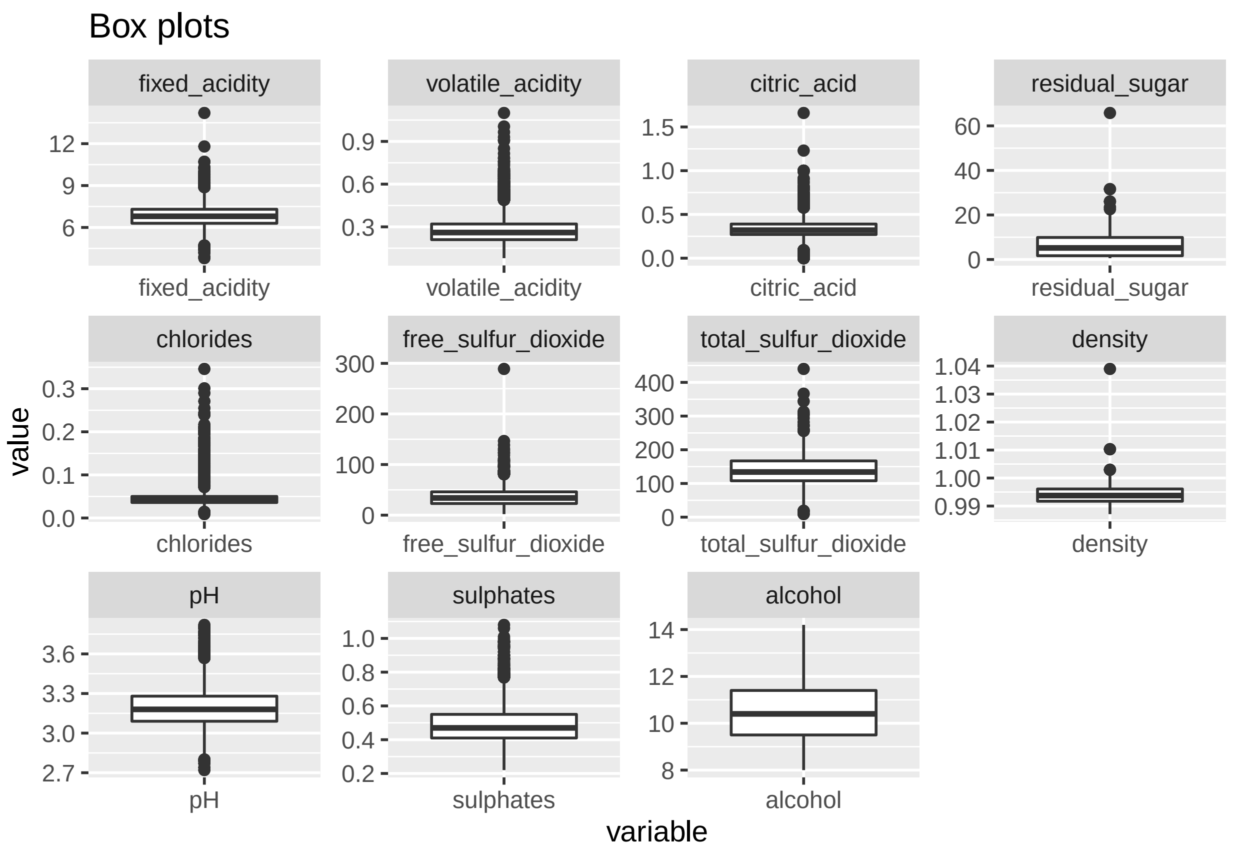
**Quality of wines distribution**



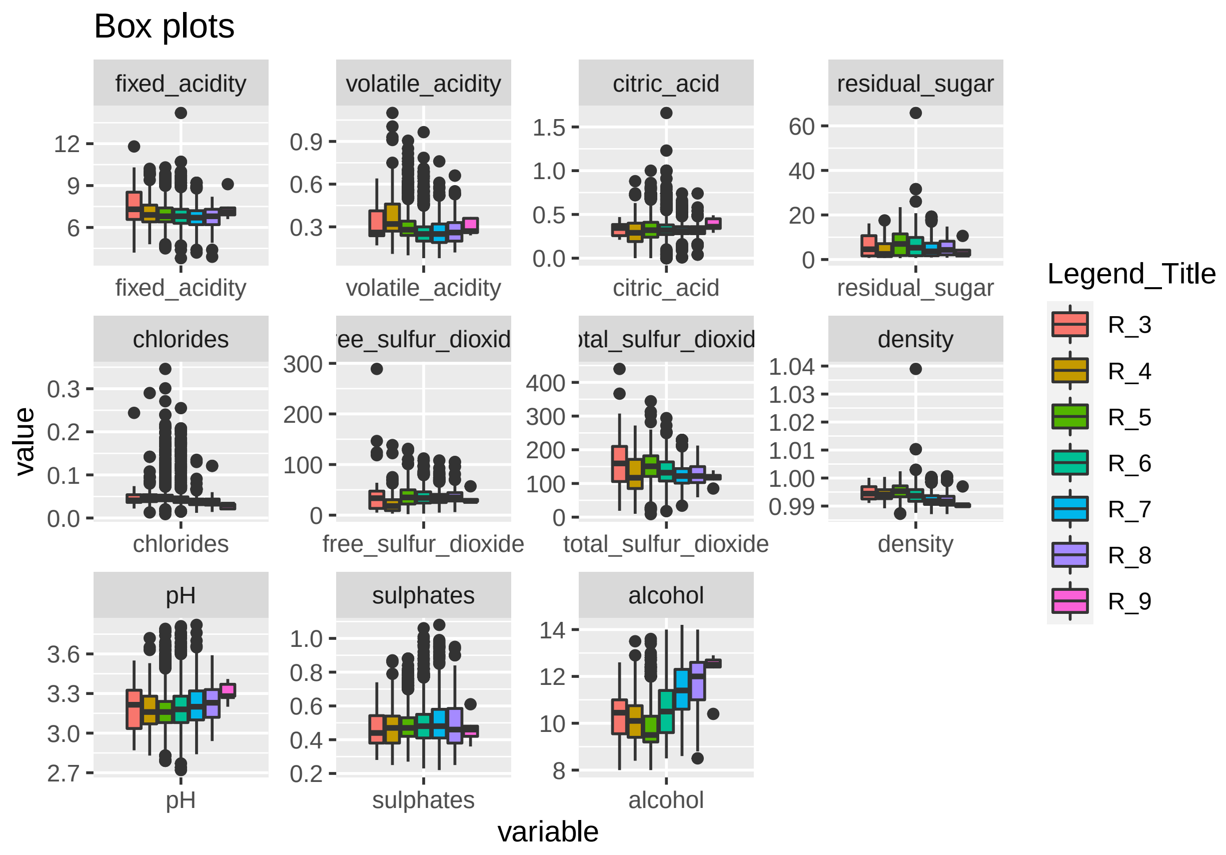
We see that most of wines have ratings between 5 and 7, being extreme ratings very rare.

### Box plots

Let's get box plots (quantiles, min and max, etc) for all variables.



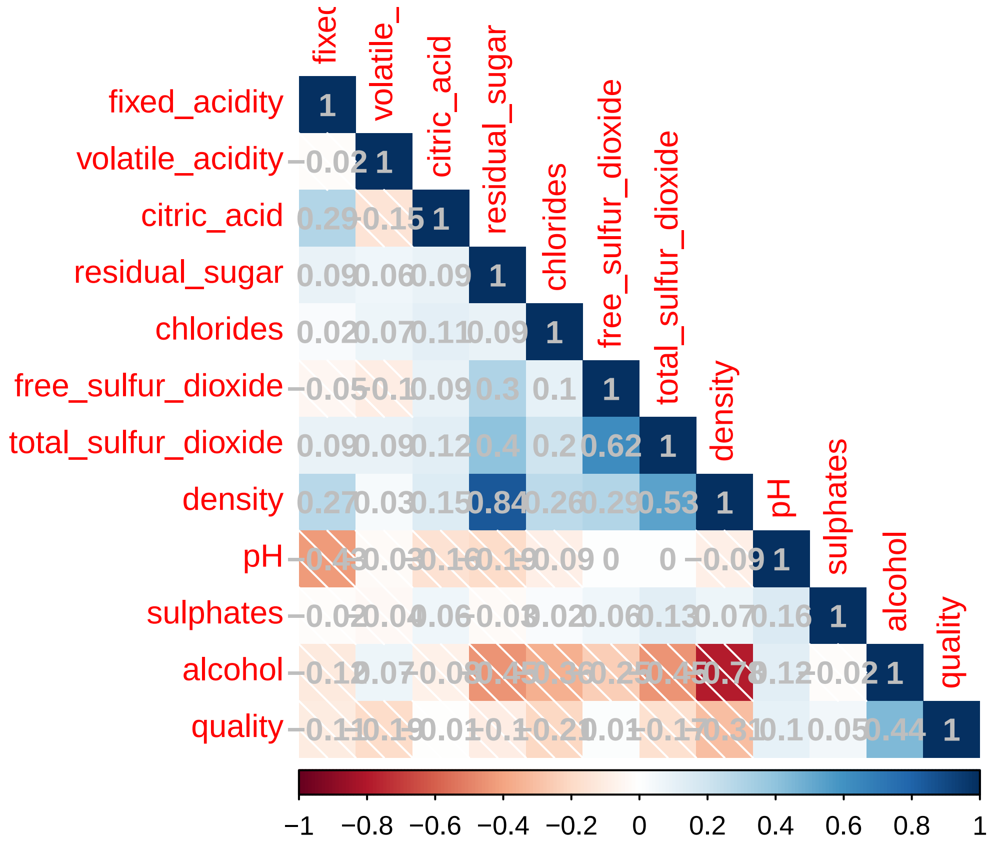
And now let's replicate it by wine rating:



We can see that the higher the rating, higher alcohol. Also there is some similar pattern for pH.

**Correlation Matrix**

Other than density & residual sugar, and alcohol & dennsity, it seems that the variables are not correlated, as it is shown in the correlation matrix:



**Modeling approach**

Let's start by preparing the data for modeling. First, we need to define what the category "buy/avoid" is. Using the descriptive statistics obtained earlier, we will define "avoid" as wines with ratings 3, 4 and 5, and "buy" was wines with ratings 6, 7, 8 and 9. "avoid" will represent 33.4% of the white wines sample, while "buy" 66.6%. Let's create this variable:

```r
wines_m <- wines %>%
  mutate(recom = factor(case_when(
    rating %in% c("R_3", "R_4","R_5") ~ "avoid",
    rating %in% c("R_6", "R_7","R_8","R_9") ~ "buy",
    TRUE ~ "other"
    )))
```

Now, let's create training and test sets to build our models. We will keep 10% of the sample as test, given that our inicial dataset consists of 4898 observations, this should be sufficient.

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
# Test set will be 10% of the entire dataset
set.seed(229, sample.kind = "Rounding")
#set.seed(229) if you are using an R version earlier than 3.6
indxTrain <- createDataPartition(y = wines_m$recom,
                                 times = 1,
                                 p = 0.9,
                                 list = FALSE)
```

```r
# Train and test sets for wine type
training <- wines_m[indxTrain,]
testing  <- wines_m[-indxTrain,]
```

Let's check the basic stats for both *train_set* and *test_test* to see if they are consistent with the entire dataset (which should be).

```r
summary(training)
```

```
##  fixed_acidity    volatile_acidity  citric_acid     residual_sugar
##  Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
##  1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700
##  Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.100
##  Mean   : 6.851   Mean   :0.2773   Mean   :0.3345   Mean   : 6.393
##  3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900
##  Max.   :14.200   Max.   :1.0050   Max.   :1.6600   Max.   :65.800
##
##    chlorides       free_sulfur_dioxide total_sulfur_dioxide
##  Min.   :0.00900   Min.   :  2.00      Min.   :  9.0
##  1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:108.0
##  Median :0.04300   Median : 34.00      Median :134.0
##  Mean   :0.04583   Mean   : 35.33      Mean   :138.3
##  3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:167.0
##  Max.   :0.29000   Max.   :289.00      Max.   :440.0
##
##     density             pH          sulphates         alcohol
##  Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00
##  1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50
##  Median :0.9938   Median :3.180   Median :0.4700   Median :10.40
##  Mean   :0.9940   Mean   :3.188   Mean   :0.4889   Mean   :10.51
##  3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40
##  Max.   :1.0390   Max.   :3.820   Max.   :1.0800   Max.   :14.20
##
##     quality       rating       recom
##  Min.   :3.000   R_3:  19   avoid:1476
##  1st Qu.:5.000   R_4: 148   buy  :2933
##  Median :6.000   R_5:1309
##  Mean   :5.874   R_6:1984
##  3rd Qu.:6.000   R_7: 795
##  Max.   :9.000   R_8: 149
##                  R_9:   5
```

```r
summary(testing)
```

```
##  fixed_acidity   volatile_acidity  citric_acid     residual_sugar
##  Min.   :4.600   Min.   :0.100    Min.   :0.0000   Min.   : 0.700
##  1st Qu.:6.300   1st Qu.:0.220    1st Qu.:0.2700   1st Qu.: 1.900
##  Median :6.800   Median :0.270    Median :0.3100   Median : 5.600
##  Mean   :6.888   Mean   :0.287    Mean   :0.3313   Mean   : 6.381
##  3rd Qu.:7.400   3rd Qu.:0.330    3rd Qu.:0.3800   3rd Qu.:10.000
##  Max.   :9.900   Max.   :1.100    Max.   :0.7900   Max.   :19.800
##
```

```
##    chlorides       free_sulfur_dioxide total_sulfur_dioxide
##  Min.   :0.01700   Min.   :  5.00      Min.   : 40.0
##  1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:109.0
##  Median :0.04300   Median : 33.00      Median :135.0
##  Mean   :0.04528   Mean   : 35.09      Mean   :138.7
##  3rd Qu.:0.05000   3rd Qu.: 45.00      3rd Qu.:166.0
##  Max.   :0.34600   Max.   :131.00      Max.   :344.0
##
##     density             pH          sulphates         alcohol
##  Min.   :0.9889   Min.   :2.770   Min.   :0.2500   Min.   : 8.70
##  1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50
##  Median :0.9936   Median :3.180   Median :0.4900   Median :10.40
##  Mean   :0.9940   Mean   :3.187   Mean   :0.4985   Mean   :10.56
##  3rd Qu.:0.9962   3rd Qu.:3.280   3rd Qu.:0.5600   3rd Qu.:11.50
##  Max.   :1.0012   Max.   :3.790   Max.   :1.0600   Max.   :13.70
##
##     quality     rating      recom
##  Min.   :3.00   R_3:  1   avoid:164
##  1st Qu.:5.00   R_4: 15   buy  :325
##  Median :6.00   R_5:148
##  Mean   :5.91   R_6:214
##  3rd Qu.:6.00   R_7: 85
##  Max.   :8.00   R_8: 26
##                 R_9:  0
```

Results are as expected (both datasets share the same estructure).

Now we are ready to develop a model using the *training set* and evaluate it using the *testing set*. Please note that in the moment that the model requires tuning, we will further partition the *training set* so we keep the *testing set* "pure" and use it just for the final evaluation.

The models that we will try are the following:

(1) KNN (k-nearest neighbors)

The k-nearest neighbors algorithm estimates the conditional probability:

$$p(x_1, .., x_p) = Pr(Y = k | X_1 = x_1, .., X_p = x_p)$$

The algorithm calculates the euclidean distance of all predictors, then for any point $(x_1, .., x_p)$ in the multi-dimensional space that we want to predict, the algorithm determines the distance to $k$ points. The $k$ nearest points is refereed as neighborhood.

For $k = 1$ the algorithm finds the distance to a single neighbor. For $k$ equals to the number of samples, the algorithm uses all points. Hence, $k$ is a tuning parameter that can be calculated running the algorithm for several values of $k$ and picking the result with highest accuracy.

(2) Random Forest

Random forests improve prediction performance over classification trees by averaging multiple decision trees. The algorithm creates several random subsets of the original data, in this case the training set, and calculates the classification trees, then the final result is the average of all trees. A tree is basically a flow chart of *yes* or *no* questions.

The name random forest derives from the random process of splitting the data and creating many trees, or a forest.

(3) Performance Measures

There are several measures to consider when evaluating the performance of a classification model as we have.

It's very important to define and understand what 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 (FP) 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.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

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

$$Sensitivity = \frac{TP}{TP + FN}$$

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

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

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

$$Specificity = \frac{TN}{TN + FP}$$

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

$$Prevalence = \frac{TP + FN}{TP + FP + TN + FN}$$

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

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

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

$$Precision = \frac{TP}{TP + FP}$$

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

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

$$Recall = \frac{TP}{TP + FN}$$

## Results

Let's start by:

### KNN

We'll use all features to predict the variable "recom" (our "recommendation" or "buy/avoid")

We also *center* and *scale* the variables since is a requirement for the technique:

```
trainX <- training[,names(training) != "recom"]
preProcValues <- preProcess(x = trainX,method = c("center", "scale"))
preProcValues
```

```
## Created from 4409 samples and 13 variables
##
## Pre-processing:
##   - centered (12)
##   - ignored (1)
##   - scaled (12)
```

We will use the caret package to run KNN

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(ISLR)) install.packages("ISLR", repos = "http://cran.us.r-project.org")
trainX <- training[,names(training) != "recom"]
trainX_Rel <-trainX[,-c(12,13)]
preProcValues <- preProcess(x = trainX_Rel,method = c("center", "scale"))
preProcValues
```

```
## Created from 4409 samples and 11 variables
##
## Pre-processing:
##   - centered (11)
##   - ignored (0)
##   - scaled (11)
```

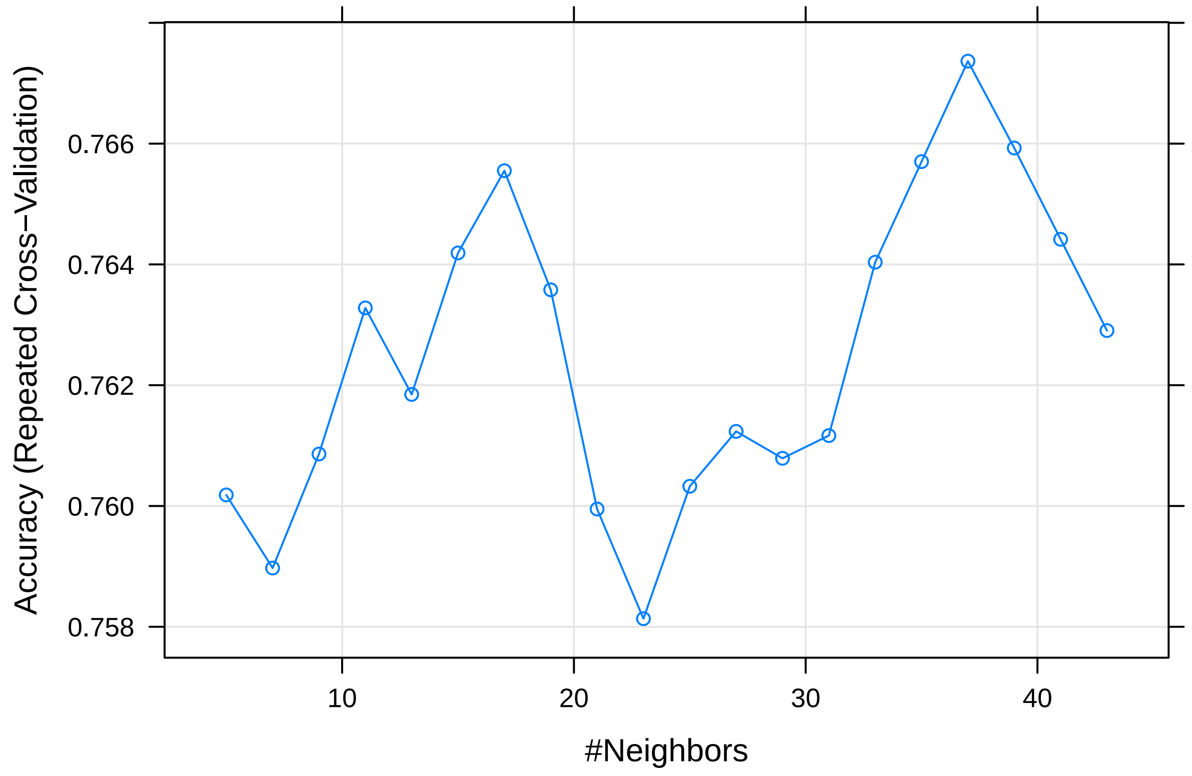
```
#training and training control
set.seed(229, sample.kind = "Rounding")
ctrl <- trainControl(method="repeatedcv",repeats = 3)
knnFit <- train(recom ~ ., data = training[,-c(12,13)], method = "knn", trControl = ctrl, preProcess = c
```

We can check the output of our model:

```
#Output of kNN fit
knnFit
```

```
## k-Nearest Neighbors
##
## 4409 samples
##   11 predictor
##    2 classes: 'avoid', 'buy'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3967, 3968, 3969, 3969, 3968, 3969, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7601829  0.4465257
##    7  0.7589727  0.4381366
##    9  0.7608604  0.4411401
##   11  0.7632793  0.4468791
##   13  0.7618470  0.4425077
##   15  0.7641901  0.4473107
##   17  0.7655512  0.4481227
##   19  0.7635796  0.4426478
##   21  0.7599496  0.4333646
##   23  0.7581342  0.4268754
##   25  0.7603265  0.4312630
##   27  0.7612354  0.4321200
##   29  0.7607893  0.4305794
##   31  0.7611659  0.4302653
##   33  0.7640367  0.4369986
##   35  0.7657014  0.4411798
##   37  0.7673653  0.4457811
##   39  0.7659271  0.4414698
##   41  0.7644161  0.4371673
##   43  0.7629045  0.4327092
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 37.
```

```r
#Plotting yields Number of Neighbours Vs accuracy (based on repeated cross validation)
plot(knnFit)
```



We can see that the final k value used for our model was k=37.

Confusion matrix is as follows:

```r
knnPredict <- predict(knnFit,newdata = testing[,-c(12,13)] )
#Get the confusion matrix to see accuracy value and other parameter values
confusionMatrix(knnPredict, testing$recom )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction avoid buy
##      avoid    91  31
##      buy      73 294
##
##                Accuracy : 0.7873
##                  95% CI : (0.7483, 0.8228)
##     No Information Rate : 0.6646
##     P-Value [Acc > NIR] : 1.637e-09
##
##                   Kappa : 0.4906
##
##  Mcnemar's Test P-Value : 5.810e-05
```

```
##
##             Sensitivity : 0.5549
##             Specificity : 0.9046
##          Pos Pred Value : 0.7459
##          Neg Pred Value : 0.8011
##              Prevalence : 0.3354
##          Detection Rate : 0.1861
##    Detection Prevalence : 0.2495
##       Balanced Accuracy : 0.7297
##
##        'Positive' Class : avoid
##
```

We see our accuracy is about 78%, but our prevalence is about 33%. Ideally we would like a greater prevalence. Also notice that in 73 cases we are telling "buy" while actually we should have suggested to "avoid", which is still high. We consider this like the worst error, since it's not that bad when you suggest to "avoid" and actually should be a "buy" (you would not have a bad experience).

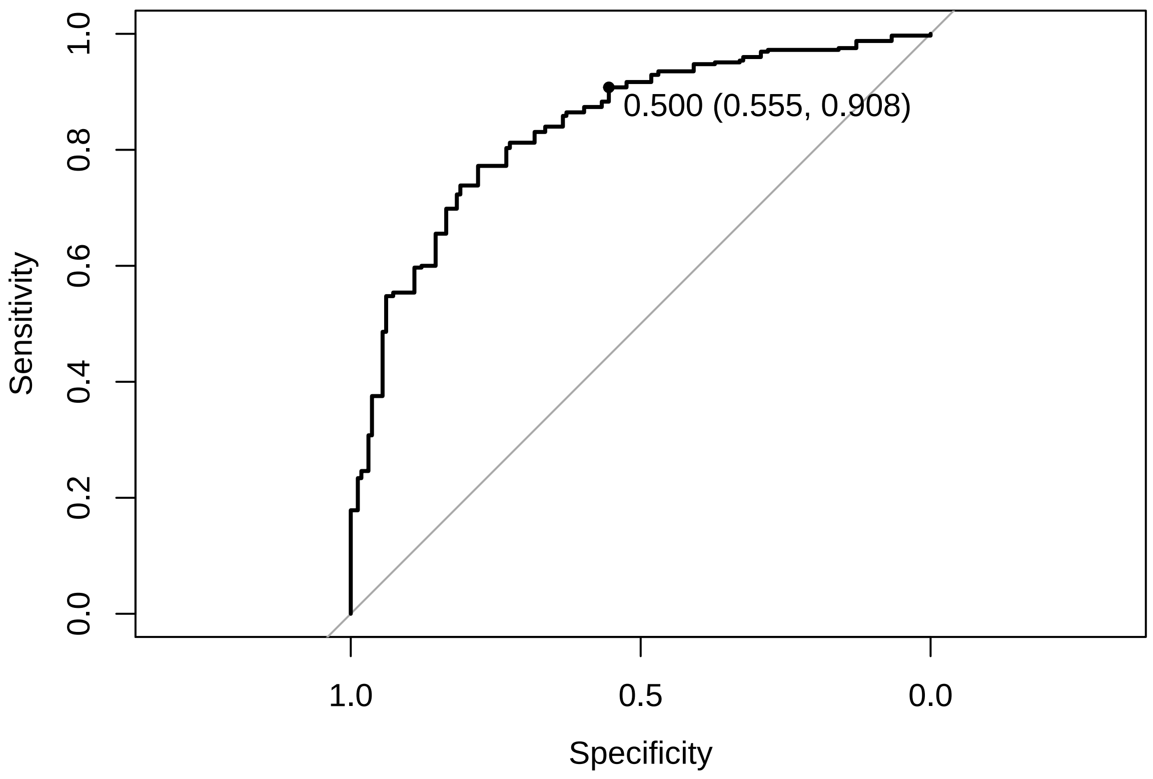
And the ROC with respective AUC is:

```r
if(!require(pROC)) install.packages("pROC", repos = "http://cran.us.r-project.org")

knnPredict <- predict(knnFit,newdata = testing , type="prob")
knnROC <- roc(testing$recom,knnPredict[,"avoid"])
knnROC
```

```
##
## Call:
## roc.default(response = testing$recom, predictor = knnPredict[,     "avoid"])
##
## Data: knnPredict[, "avoid"] in 164 controls (testing$recom avoid) > 325 cases (testing$recom buy).
## Area under the curve: 0.8454
```

```r
plot(knnROC, type="S", print.thres= 0.5)
```



So AUC is 0.8454… not terrible but could be improved, which we will try to do using Random Forest.

**Random Forest**

We have already installed the packages to use (caret and pROC)… let's start by training:

```
set.seed(229, sample.kind = "Rounding")
ctrl <- trainControl(method="repeatedcv",repeats = 3)
# Random forrest
rfFit <- train(recom~., data=training[,-c(12,13)],method="rf",trControl= ctrl,preProcess=c("center","sc
```

```
## note: only 10 unique complexity parameters in default grid. Truncating the grid to 10 .
```

The trained model looks like:

```
rfFit
```

```
## Random Forest
##
## 4409 samples
##   11 predictor
##    2 classes: 'avoid', 'buy'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3967, 3968, 3969, 3969, 3968, 3969, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8438727  0.6376775
##    3    0.8412280  0.6326579
##    4    0.8413766  0.6335405
##    5    0.8397139  0.6303019
##    6    0.8390334  0.6287024
##    7    0.8392602  0.6301242
##    8    0.8378229  0.6265734
##    9    0.8394119  0.6305751
##   10    0.8383515  0.6283889
##   11    0.8372931  0.6264728
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Confusion matrix:

```
rfPredict <- predict(rfFit,newdata = testing[,-c(12,13)] )
confusionMatrix(rfPredict, testing$recom )
```

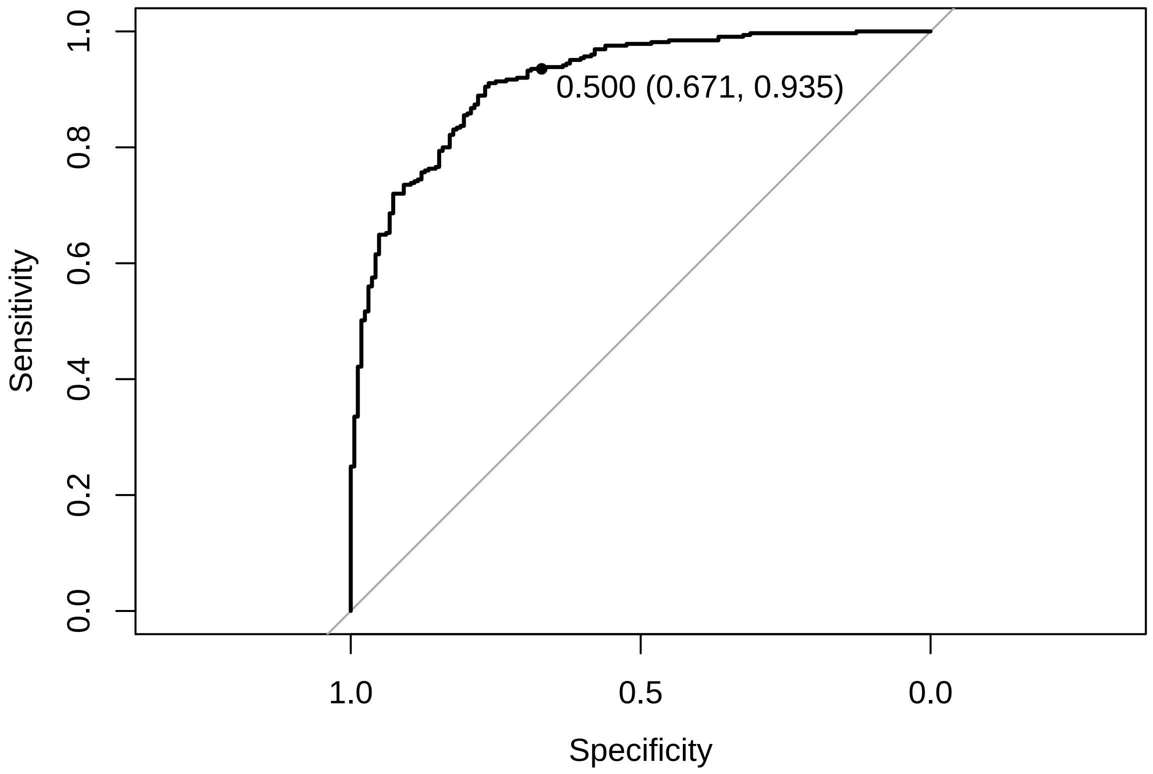
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction avoid buy
##      avoid   110  21
##      buy      54 304
##
##                Accuracy : 0.8466
##                  95% CI : (0.8116, 0.8774)
##     No Information Rate : 0.6646
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6379
##
##  Mcnemar's Test P-Value : 0.0002199
##
##             Sensitivity : 0.6707
##             Specificity : 0.9354
##          Pos Pred Value : 0.8397
##          Neg Pred Value : 0.8492
##              Prevalence : 0.3354
##          Detection Rate : 0.2249
##    Detection Prevalence : 0.2679
##       Balanced Accuracy : 0.8031
##
##        'Positive' Class : avoid
##
```

We improved accuracy, sensitivity and specificity… we have less number of false positives. Let's see how ROC and AUC look like:

```r
rfPredict <- predict(rfFit,newdata = testing[,-c(12,13)] , type="prob")
rfROC <- roc(testing$recom,rfPredict[,"avoid"])
rfROC
```

```
##
## Call:
## roc.default(response = testing$recom, predictor = rfPredict[,     "avoid"])
##
## Data: rfPredict[, "avoid"] in 164 controls (testing$recom avoid) > 325 cases (testing$recom buy).
## Area under the curve: 0.9156
```

```r
plot(rfROC, type="S", print.thres= 0.5)
```



Now the AUC is 0.9156, which is much better.

## Conclusions

In this report we have shown how to use the caret package to perform classification techniques in R, more especifically KNN and Random Forest, in order to predict white wines quality.

First we have explored the data, providing statistic summaries like quantiles, distributions, correlation, etc. Then we have shown how to perform KNN and Random Forest using the caret package.

We showed that we improve the KNN results by running Random Forest and obtaining an AUC = 0.9156. So we could say we have a model to "recommend" whether to "buy" or "avoid" a white wine based on variables generated by physicochemical tests.

As limitations we could highlight:

1. Personally I run into hardward limitations when running Random Forest, which limited my capacity to run several iterations with different parameters for example.
2. This model only uses features created by physicochemical tests, so we don't count with other information that could be valuable like market price, or customer experience.

Some next steps would be:

1. Enhance the models running different iterations with different parameters.
2. Use other models like LDA, QDA, Logistic Regression, etc… and even some clustering techniques (maybe with a modified goal, but still classifying).

## References

- 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.