PH125.9x Data Science: Capstone - Application and comparison of different Machine Learning methods to discover the quality of red wine

Fernando Gripe

February 22, 2023

1. Introduction and Machine Learning

According to Javatpoint Machine learning is a data analytics technique that teaches computers learn from experience. Machine learning algorithms use computational methods to directly "learn" from data. As the number of samples available for learning increases, the algorithm adapts to improve performance.

Machine learning can uses two techniques: supervised learning, which trains a model on known input and output data to predict future outputs, and unsupervised learning, which uses hidden patterns or internal structures in the input data. In the case of this work, we will apply supervised machine learning techniques to estimate the quality of red wines.

Supervised machine learning creates a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to the new data.

Supervised learning uses classification and regression techniques to develop machine learning models. Classification models classify the input data. Classification techniques predict discrete responses. For example, the email is genuine, or spam, if the tumor is cancerous or benign.

Common algorithms for performing classification include support vector machines (SVMs), boosted and bagged decision trees, k-nearest neighbors, Naive Bayes, discriminant analysis, logistic regression, and neural networks.

Regression techniques predict continuous responses - for example, changes in temperature or fluctuations in electricity demand. Typical applications include power load forecasting and algorithmic trading.

2. Overview and purpose of the project

This project is a requirement for the HarvardX Professional Certificate Data Science Program.

Candidates were free to choose a publicly available data set to apply ML techniques. In this work, the dataset choosed was the Wine Quality dataset from UCI. The dataset was assembled with the intention of being used in this scientific article by Cortez et al., 2009.

The objective in this project is to train a machine learning algorithm that predicts the wine quality using the inputs of a train subset and the validation subset.

Some works have already published on the subject and using this data set used groupings in the notes of the wines in order to simplify the prediction process. That is, they arbitrated between 'good'/'bad' or 'bad'/'medium'/'good' wines.

The purpose here is precisely to go the hard way, using wine notes without simplifications and trying various ML techniques in order to practice their use and also compare their results and possible adjustments.

That is, the objective is not to obtain the best possible result, but rather to create a robust framework of knowledge to be used in other datasets in the future.

3. Wine Quality

According to A.G. Reynolds, 2010, wine quality is the result of a complex set of interactions, which include geological and soil variables, climate, and many variables, climate, and many viticultural decisions.

Nevertheless, Doris Rauhut, Florian Kiene, 2019 says that red wine quality and style are highly influenced by the qualitative and quantitative composition of aromatic compounds having various chemical structures and properties and their interaction within different red wine matrices. The understanding of interactions between the wine matrix and volatile compounds and the impact on the overall flavor as well as on typical or specific aromas is getting more and more important for the creation of certain wine styles.

3.1. Wine dataset

Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.). The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones).

The input features are as follows:

- fixed acidity - most acids involved with wine or fixed or nonvolatile (do not evaporate readily);
- volatile acidity - the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste;
- citric acid - found in small quantities, citric acid can add ‘freshness’ and flavor to wines;
- residual sugar - the amount of sugar remaining after fermentation stops, it’s rare to find wines with less than 1 gram/liter and wines with greater than 45 grams/liter are considered sweet;
- chlorides - the amount of salt in the wine;
- free sulfur dioxide - the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion; it prevents microbial growth and the oxidation of wine;
- total sulfur dioxide - amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2 concentrations over 50 ppm, SO2 becomes evident in the nose and taste of wine;
- density - the density of water is close to that of water depending on the percent alcohol and sugar content;
- pH - describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the pH scale;
- sulphates - a wine additive which can contribute to sulfur dioxide gas (S02) levels, wich acts as an antimicrobial and antioxidant;
- alcohol - the percent alcohol content of the wine;

The output feature is:

- quality - output variable (based on sensory data, score between 3 and 8 in this dataset);

4. Basic Setup and Data Preparation

4.1. Basic Setup

```
# Install and load libraries
# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(reshape)) install.packages("reshape", repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")
if(!require(visNetwork)) install.packages("visNetwork", repos = "http://cran.us.r-project.org")
if(!require(GGally)) install.packages("GGally", repos = "http://cran.us.r-project.org")
if(!require(MASS)) install.packages("MASS", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(nnet)) install.packages("nnet", repos = "http://cran.us.r-project.org")
if(!require(class)) install.packages("class", repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")
if(!require(xgboost)) install.packages("xgboost", repos = "http://cran.us.r-project.org")
if(!require(png)) install.packages("png", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(ggplot2)
library(reshape)
library(rpart)
library(visNetwork)
library(GGally)
library(class)
library(cvms)
library(png)

library(caret)        #confusionMatrix
library(MASS)         #Linear Discriminant Analysis
library(randomForest) #random forest
library(nnet)         #multinom - logistic regression
library(xgboost)      #xgboost
library(e1071)        #Naive Bayes Classifier

options(timeout = 120)
set.seed(42)
```

4.2 Data Preparation

```
# Dataset
# Load original dataset and create Train and Test dataset

#data_loading and processing
df_red = read.csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-r

df_red <- df_red %>% mutate(quality.factor = as.factor(df_red$quality))

#spliting into train (80%) and test (20%) set
indx = sample(1:nrow(df_red), size = 0.8 * nrow(df_red))
```

```
TrainSet = df_red[indx,]
TestSet = df_red[-indx,]

rm(indx)

#checkin the split, if test is 20% of total rows
nrow(TestSet) / ( nrow(TrainSet) + nrow(TestSet) )
```

```
## [1] 0.2001251
```

5. Exploratory Analysis

5.1 Basic Structure

There are only 1279 records in the training sample and just below we will understand the quantities in each quality classification. This is an important and constraining factor in the robustness of the strength we will have to apply ML techniques since the sample is relatively small given the large number of notes/quality that each wine can have.

5.1.1 Display the Structure of the dataset

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 4.60   Min.   :0.120    Min.   :0.000   Min.   : 0.90
##  1st Qu.: 7.10   1st Qu.:0.395    1st Qu.:0.090   1st Qu.: 1.90
##  Median : 7.90   Median :0.520    Median :0.260   Median : 2.20
##  Mean   : 8.29   Mean   :0.530    Mean   :0.269   Mean   : 2.52
##  3rd Qu.: 9.15   3rd Qu.:0.640    3rd Qu.:0.420   3rd Qu.: 2.60
##  Max.   :15.90   Max.   :1.580    Max.   :0.790   Max.   :15.50
##    chlorides      free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.0120   Min.   : 1.0        Min.   :  6.0        Min.   :0.990
##  1st Qu.:0.0700   1st Qu.: 7.0        1st Qu.: 22.0        1st Qu.:0.996
##  Median :0.0790   Median :14.0        Median : 38.0        Median :0.997
##  Mean   :0.0874   Mean   :15.8        Mean   : 46.3        Mean   :0.997
##  3rd Qu.:0.0900   3rd Qu.:21.0        3rd Qu.: 62.0        3rd Qu.:0.998
##  Max.   :0.6110   Max.   :68.0        Max.   :289.0        Max.   :1.004
##        pH          sulphates        alcohol        quality     quality.factor
##  Min.   :2.86   Min.   :0.330   Min.   : 8.4   Min.   :3.00   3:  8
##  1st Qu.:3.21   1st Qu.:0.550   1st Qu.: 9.5   1st Qu.:5.00   4: 44
##  Median :3.31   Median :0.620   Median :10.2   Median :6.00   5:544
##  Mean   :3.31   Mean   :0.657   Mean   :10.4   Mean   :5.63   6:517
##  3rd Qu.:3.40   3rd Qu.:0.730   3rd Qu.:11.1   3rd Qu.:6.00   7:153
##  Max.   :4.01   Max.   :1.950   Max.   :14.9   Max.   :8.00   8: 13
```

5.1.2 Data head lines

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide
561	12.7	0.60	0.49	2.8	0.075	5
321	9.8	0.66	0.39	3.2	0.083	21
1177	6.5	0.88	0.03	5.6	0.079	23
1098	8.6	0.52	0.38	1.5	0.096	5
1252	7.5	0.58	0.14	2.2	0.077	27
1170	7.6	0.50	0.29	2.3	0.086	5

	total.sulfur.dioxide	density	pH	sulphates	alcohol	quality	quality.factor
561	19	0.99940	3.14	0.57	11.4	5	5
321	59	0.99890	3.37	0.71	11.5	7	7
1177	47	0.99572	3.58	0.50	11.2	4	4
1098	18	0.99666	3.20	0.52	9.4	5	5
1252	60	0.99630	3.28	0.59	9.8	5	5
1170	14	0.99502	3.32	0.62	11.5	6	6

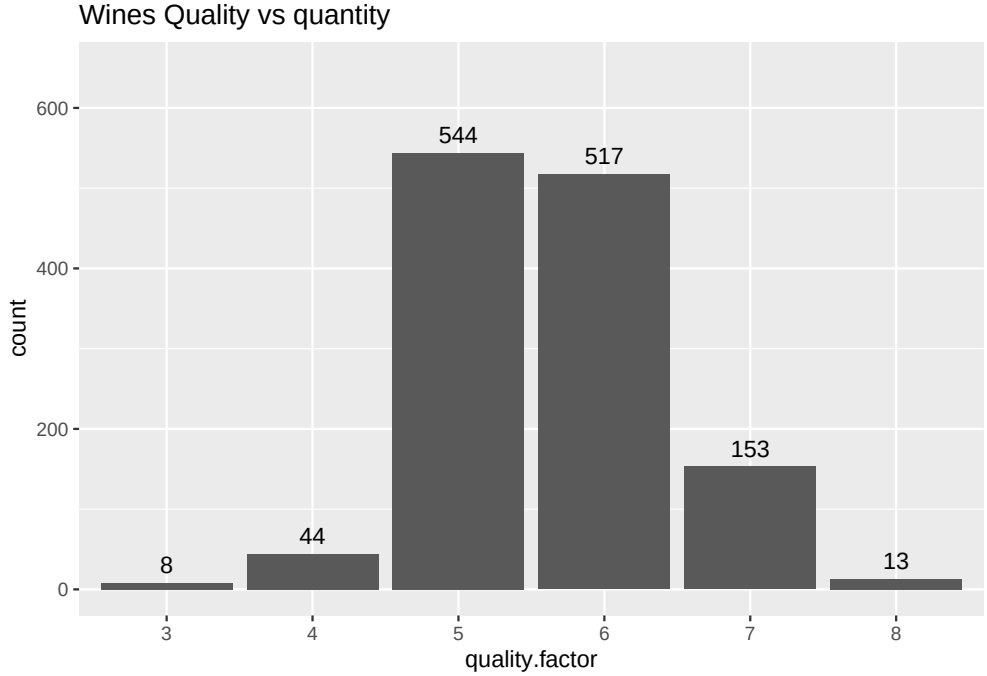
5.1.3 Data Sumary

```
## 'data.frame':    1279 obs. of  13 variables:
##  $ fixed.acidity       : num  12.7 9.8 6.5 8.6 7.5 7.6 10.1 6.4 6.1 6.7 ...
##  $ volatile.acidity    : num  0.6 0.66 0.88 0.52 0.58 0.5 0.935 0.4 0.58 0.46 ...
##  $ citric.acid         : num  0.49 0.39 0.03 0.38 0.14 0.29 0.22 0.23 0.23 0.24 ...
##  $ residual.sugar      : num  2.8 3.2 5.6 1.5 2.2 2.3 3.4 1.6 2.5 1.7 ...
##  $ chlorides           : num  0.075 0.083 0.079 0.096 0.077 0.086 0.105 0.066 0.044 0.077 ...
##  $ free.sulfur.dioxide : num  5 21 23 5 27 5 11 5 16 18 ...
##  $ total.sulfur.dioxide: num  19 59 47 18 60 14 86 12 70 34 ...
##  $ density             : num  0.999 0.999 0.996 0.997 0.996 ...
##  $ pH                  : num  3.14 3.37 3.58 3.2 3.28 3.32 3.43 3.34 3.46 3.39 ...
##  $ sulphates           : num  0.57 0.71 0.5 0.52 0.59 0.62 0.64 0.56 0.65 0.6 ...
##  $ alcohol             : num  11.4 11.5 11.2 9.4 9.8 11.5 11.3 9.2 12.5 10.6 ...
##  $ quality             : int  5 7 4 5 5 6 4 5 6 6 ...
##  $ quality.factor      : Factor w/ 6 levels "3","4","5","6",..: 3 5 2 3 3 4 2 3 4 4 ...
```

5.2 Data Exploration

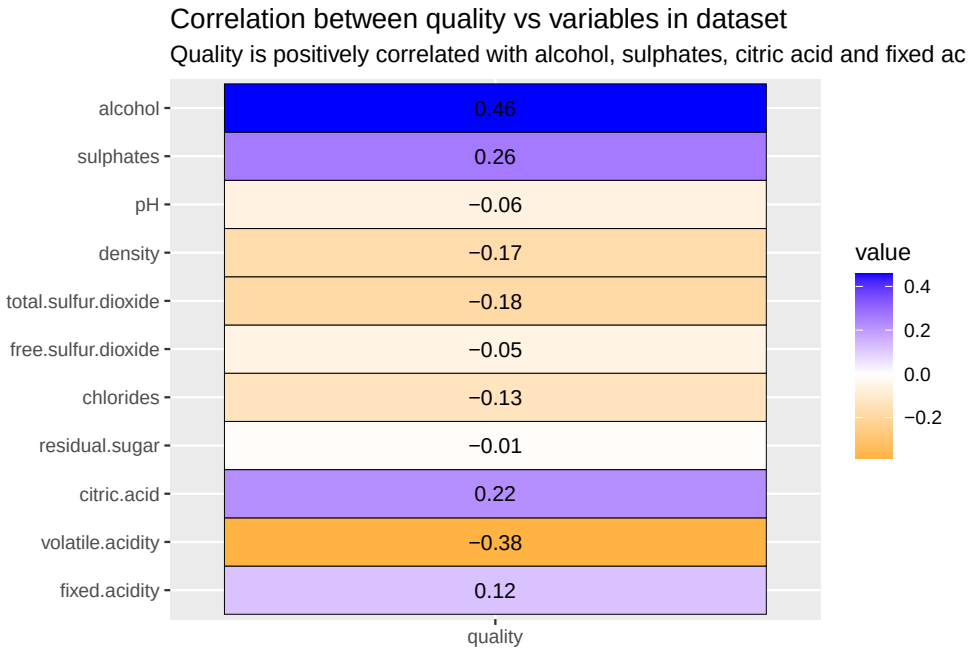
5.2.1 Plot: Wines Quality vs quantity

We can see in the graph below that most of the records in the training sample are concentrated in the quality classification 6 and 5. Quality 7 is the third most frequent and the others contain only a few records.



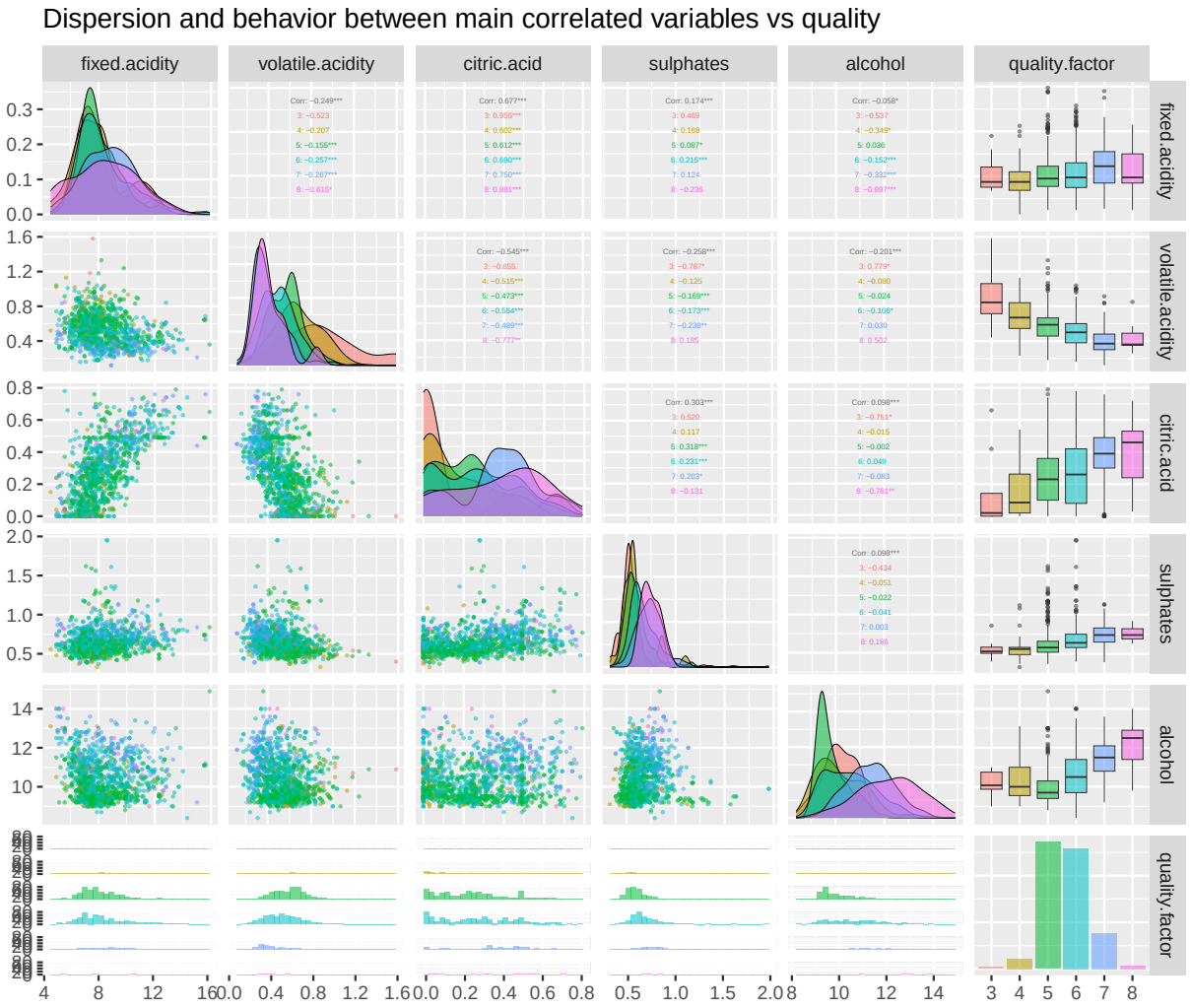
5.2.2 Plot: Correlation between quality vs variables in dataset

Crossing the correlation between the variables present in the dataset with 'quality', we have that 'quality' is positively correlated with alcohol, sulphates, citric acid and fixed acidity. However, is negatively correlated with volatile acidity'



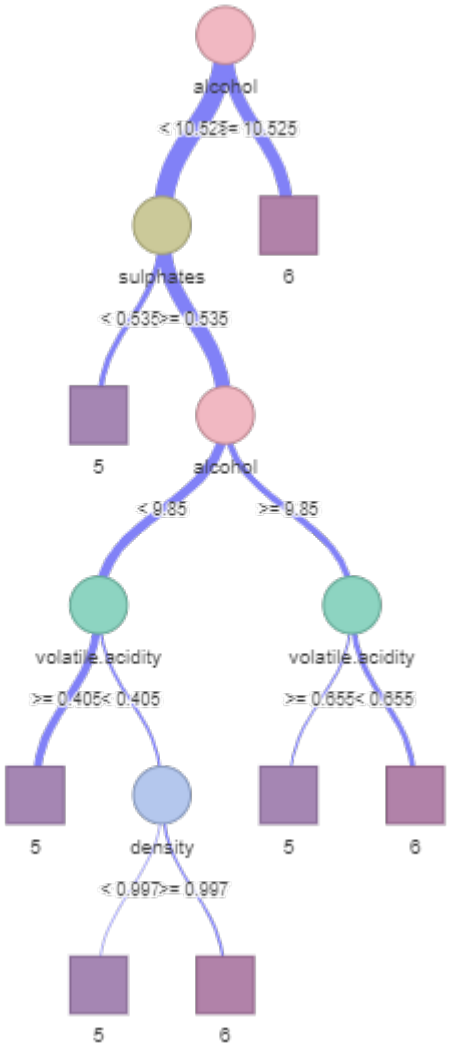
5.2.3 Plot: Dispersion and behavior between main correlated variables vs quality

Considering only the variables with the highest correlation vs 'quality' I used ggpairs to visually understand the differences in the characteristics of the selected variables vs 'quality'. We noticed that for some characteristics it is notable a distinct behavior of some degrees of 'quality', mainly the note 3 for many cases or for the note 5 for the concentration of alcohol. This form of visualization helps us to compare the sample dispersions, but the averages, medians and others can also be proved by the histograms on the right side.



5.2.4 Plot: Wine Quality Classification Tree

Another interesting tool to understand possible paths for classifying the sample is through a decision tree. It is interesting to see how the main characteristics found in the correlation study (alcohol, sulphates, citric acid and fixed acidity) make a difference in the branches of the decision tree.



6. Applying Machine Learning techniques

6.1. Nayve Bayes

Naive Bayes is a Supervised Machine Learning algorithm based on the Bayes Theorem that is used to solve classification problems by following a probabilistic approach. It is based on the idea that the predictor variables in a Machine Learning model are independent of each other. Meaning that the outcome of a model depends on a set of independent variables that have nothing to do with each other.

The principle behind Naive Bayes is the Bayes theorem also known as the Bayes Rule. The Bayes theorem is used to calculate the conditional probability, which is nothing but the probability of an event occurring based on information about the events in the past.

```
mod.naivebayes <- naiveBayes(quality.factor ~ ., TrainSet[,-c(12)])
confusionMatrix(predict(mod.naivebayes,  TestSet), TestSet$quality.factor)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  1  0  0  0
##          4  0  4  2  3  1  0
##          5  2  4 89 22  3  0
##          6  0  1 43 72 18  2
##          7  0  0  2 22 21  3
##          8  0  0  0  2  3  0
##
## Overall Statistics
##
##                Accuracy : 0.5812
##                  95% CI : (0.5251, 0.6359)
##     No Information Rate : 0.4281
##     P-Value [Acc > NIR] : 2.624e-08
##
##                   Kappa : 0.3617
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000  0.44444   0.6496   0.5950  0.45652  0.00000
## Specificity          0.996855  0.98071   0.8306   0.6784  0.90146  0.98413
## Pos Pred Value       0.000000  0.40000   0.7417   0.5294  0.43750  0.00000
## Neg Pred Value       0.993730  0.98387   0.7600   0.7337  0.90809  0.98413
## Prevalence           0.006250  0.02813   0.4281   0.3781  0.14375  0.01562
## Detection Rate       0.000000  0.01250   0.2781   0.2250  0.06563  0.00000
## Detection Prevalence 0.003125  0.03125   0.3750   0.4250  0.15000  0.01562
## Balanced Accuracy    0.498428  0.71258   0.7401   0.6367  0.67899  0.49206
```

Naive Bayes achieved 0.5812 of accuracy and the highest sensitivity occurred in grade 5, at 0.6496. In the Confusion Matrix we can see that most of the prediction errors occurred between grades 5, 6 and 7.

6.2. KNN

KNN is a Supervised Learning algorithm that uses labeled input data set to predict the output of the data points. It is mainly based on feature similarity. KNN checks how similar a data point is to its neighbor and

classifies the data point into the class it is most similar to.

To apply the KNN model first its important to calculate the the square root of the number of records to be used. as we arrived at a decimal number, I ran the two possible possibilities in order to compare the accuracy between these two possibilities.

```
#Finding the number of observations
sqrt(NROW(TrainSet))
```

```
## [1] 35.76311
```

```
#trying both

mod.knn.35 <- knn(train=TrainSet[,-c(12)], test=TestSet[,-c(12)], cl=TrainSet[,c(13)], k=35)
mod.knn.36 <- knn(train=TrainSet[,-c(12)], test=TestSet[,-c(12)], cl=TrainSet[,c(13)], k=36)

confusionMatrix(table(mod.knn.35 ,TestSet$quality.factor))
```

```
## Confusion Matrix and Statistics
##
##
## mod.knn.35   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   0   0   0   0   0   0
##          5   1   8 103  38  11   0
##          6   1   1  34  81  32   3
##          7   0   0   0   2   3   2
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5844
##                  95% CI : (0.5282, 0.6389)
##     No Information Rate : 0.4281
##     P-Value [Acc > NIR] : 1.388e-08
##
##                   Kappa : 0.3094
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000  0.00000   0.7518   0.6694 0.065217  0.00000
## Specificity           1.00000  1.00000   0.6831   0.6432 0.985401  1.00000
## Pos Pred Value            NaN      NaN   0.6398   0.5329 0.428571      NaN
## Neg Pred Value        0.99375  0.97188   0.7862   0.7619 0.862620  0.98438
## Prevalence            0.00625  0.02813   0.4281   0.3781 0.143750  0.01562
## Detection Rate        0.00000  0.00000   0.3219   0.2531 0.009375  0.00000
## Detection Prevalence  0.00000  0.00000   0.5031   0.4750 0.021875  0.00000
## Balanced Accuracy     0.50000  0.50000   0.7174   0.6563 0.525309  0.50000
```

```
confusionMatrix(table(mod.knn.36 ,TestSet$quality.factor))
```

```
## Confusion Matrix and Statistics
##
##
```

```
## mod.knn.36   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   0   0   0   0   0   0
##          5   1   8 103  38  10   0
##          6   1   1  34  81  33   4
##          7   0   0   0   2   3   1
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5844
##                  95% CI : (0.5282, 0.6389)
##     No Information Rate : 0.4281
##     P-Value [Acc > NIR] : 1.388e-08
##
##                   Kappa : 0.3088
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000  0.00000   0.7518   0.6694 0.065217  0.00000
## Specificity           1.00000  1.00000   0.6885   0.6332 0.989051  1.00000
## Pos Pred Value            NaN      NaN   0.6437   0.5260 0.500000      NaN
## Neg Pred Value        0.99375  0.97188   0.7875   0.7590 0.863057  0.98438
## Prevalence            0.00625  0.02813   0.4281   0.3781 0.143750  0.01562
## Detection Rate        0.00000  0.00000   0.3219   0.2531 0.009375  0.00000
## Detection Prevalence  0.00000  0.00000   0.5000   0.4813 0.018750  0.00000
## Balanced Accuracy     0.50000  0.50000   0.7202   0.6513 0.527134  0.50000
```

KNN achieved 0.5844 of accuracy and the highest sensitivity occurred in grade 5, at 0.7518. In the Confusion Matrix we can see that most of the prediction errors occurred between grades 5, 6 and 7.

6.3. Multinomial Logistic Regression

Multinomial logistic regression is used to model nominal outcome variables, in which the log odds of the outcomes are modeled as a linear combination of the predictor variables.

```
mod.logistic.regression = multinom(quality.factor~., TrainSet[,-c(12)])
```

```
## # weights:  78 (60 variable)
## initial  value 2291.660361
## iter  10 value 1522.270330
## iter  20 value 1394.257985
## iter  30 value 1207.984141
## iter  40 value 1181.197977
## iter  50 value 1177.562490
## iter  60 value 1175.818276
## iter  70 value 1173.757381
## iter  80 value 1172.102941
## iter  90 value 1171.882187
## iter 100 value 1171.722472
## final  value 1171.722472
## stopped after 100 iterations
```

```
confusionMatrix(predict(mod.logistic.regression, TestSet),  TestSet$quality.factor)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   1   0   0   0
##          4   0   0   0   0   0   0
##          5   2   7 104  34   2   0
##          6   0   2  32  78  28   3
##          7   0   0   0   9  16   2
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6188
##                  95% CI : (0.5631, 0.6722)
##     No Information Rate : 0.4281
##     P-Value [Acc > NIR] : 5.477e-12
##
##                   Kappa : 0.3846
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000  0.00000   0.7591   0.6446  0.34783  0.00000
## Specificity          0.996855  1.00000   0.7541   0.6734  0.95985  1.00000
## Pos Pred Value       0.000000      NaN   0.6980   0.5455  0.59259      NaN
## Neg Pred Value       0.993730  0.97188   0.8070   0.7571  0.89761  0.98438
## Prevalence           0.006250  0.02813   0.4281   0.3781  0.14375  0.01562
## Detection Rate       0.000000  0.00000   0.3250   0.2437  0.05000  0.00000
## Detection Prevalence 0.003125  0.00000   0.4656   0.4469  0.08438  0.00000
## Balanced Accuracy    0.498428  0.50000   0.7566   0.6590  0.65384  0.50000
```

MLR achieved 0.6188 of accuracy and the highest sensitivity occurred in grade 5, at 0.7591. In the Confusion Matrix we can see that most of the prediction errors occurred between grades 5, 6 and 7.

6.4. LDA

Linear Discriminant Analysis (LDA) is a dimensionality reduction technique. LDA used for dimensionality reduction to reduce the number of dimensions (i.e. variables) in a dataset while retaining as much information as possible.

LDA is used to determine group means and also for each individual, it tries to compute the probability that the individual belongs to a different group.

```
mod.linear.discriminant.analysis  <-lda(quality.factor ~ ., TrainSet[,-c(12)])
confusionMatrix(predict(mod.linear.discriminant.analysis, TestSet)$class, TestSet$quality.factor)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   2   1   0   0   0
##          4   0   0   0   0   0   0
##          5   2   5 105  35   2   0
##          6   0   2  30  71  25   3
##          7   0   0   1  15  19   2
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6094
##                  95% CI : (0.5535, 0.6632)
##     No Information Rate : 0.4281
##     P-Value [Acc > NIR] : 5.413e-11
##
##                   Kappa : 0.3792
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000  0.00000   0.7664   0.5868  0.41304  0.00000
## Specificity          0.990566  1.00000   0.7596   0.6985  0.93431  1.00000
## Pos Pred Value       0.000000      NaN   0.7047   0.5420  0.51351      NaN
## Neg Pred Value       0.993691  0.97188   0.8129   0.7354  0.90459  0.98438
## Prevalence           0.006250  0.02813   0.4281   0.3781  0.14375  0.01562
## Detection Rate       0.000000  0.00000   0.3281   0.2219  0.05937  0.00000
## Detection Prevalence 0.009375  0.00000   0.4656   0.4094  0.11563  0.00000
## Balanced Accuracy    0.495283  0.50000   0.7630   0.6426  0.67368  0.50000
```

LDA achieved 0.6094 of accuracy and the highest sensitivity occurred in grade 5, at 0.7664. In the Confusion Matrix we can see that most of the prediction errors occurred between grades 5, 6 and 7.

6.5. Random Forest

Random Forest is an ensemble of decision trees. It builds and combines multiple decision trees to get more accurate predictions. It's a non-linear classification algorithm. Each decision tree model is used when employed on its own.

Random Forest are called 'random' because they choose predictors randomly at a time of training. They are called forest because they take the output of multiple trees to make a decision. Random forest outperforms decision trees as a large number of uncorrelated trees(models) operating as a committee will always outperform the individual constituent models.

```
#random forest default
set.seed(42)
mod.random.forest.default <- randomForest(quality.factor~., TrainSet[,-c(12)])
```

```
confusionMatrix(predict(mod.random.forest.default, TestSet), TestSet$quality.factor)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   0   0   0   0   0   0
##          5   1   3 115  25   0   0
##          6   1   6  22  90  16   3
##          7   0   0   0   6  30   2
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.7344
##                  95% CI : (0.6824, 0.782)
##     No Information Rate : 0.4281
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5765
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000  0.00000   0.8394   0.7438  0.65217  0.00000
## Specificity           1.00000  1.00000   0.8415   0.7588  0.97080  1.00000
## Pos Pred Value            NaN      NaN   0.7986   0.6522  0.78947      NaN
## Neg Pred Value        0.99375  0.97188   0.8750   0.8297  0.94326  0.98438
## Prevalence            0.00625  0.02813   0.4281   0.3781  0.14375  0.01562
## Detection Rate        0.00000  0.00000   0.3594   0.2812  0.09375  0.00000
## Detection Prevalence  0.00000  0.00000   0.4500   0.4313  0.11875  0.00000
## Balanced Accuracy     0.50000  0.50000   0.8405   0.7513  0.81149  0.50000
```

RF achieved 0.7344 of accuracy and the highest sensitivity occurred in grade 5, at 0.8394. In the Confusion Matrix we can see that most of the prediction errors occurred between grades 5, 6 and 7.

6.5.1. Random Forest mtry analyse

According to Manish Saraswat there are three parameters in the random forest algorithm which we should look at for tuning:

ntree: The number of trees to grow. Larger the tree, it will be more computationally expensive to build models.

nodesize: Refers to how many observations we want in the terminal nodes. This parameter is directly related to tree depth. Higher the number, lower the tree depth. With lower tree depth, the tree might even fail to recognize useful signals from the data.

mtry: Refers to how many variables we should select at a node split. Also as mentioned above, the default value is p/3 for regression and sqrt(p) for classification. We should always try to avoid using smaller values of mtry to avoid overfitting.

Mtry is a good starting point for tuning the model and can give good results. I will plot below in a graph the possible values for mtry and their practical effects on the result. After that, I will use the best value to run

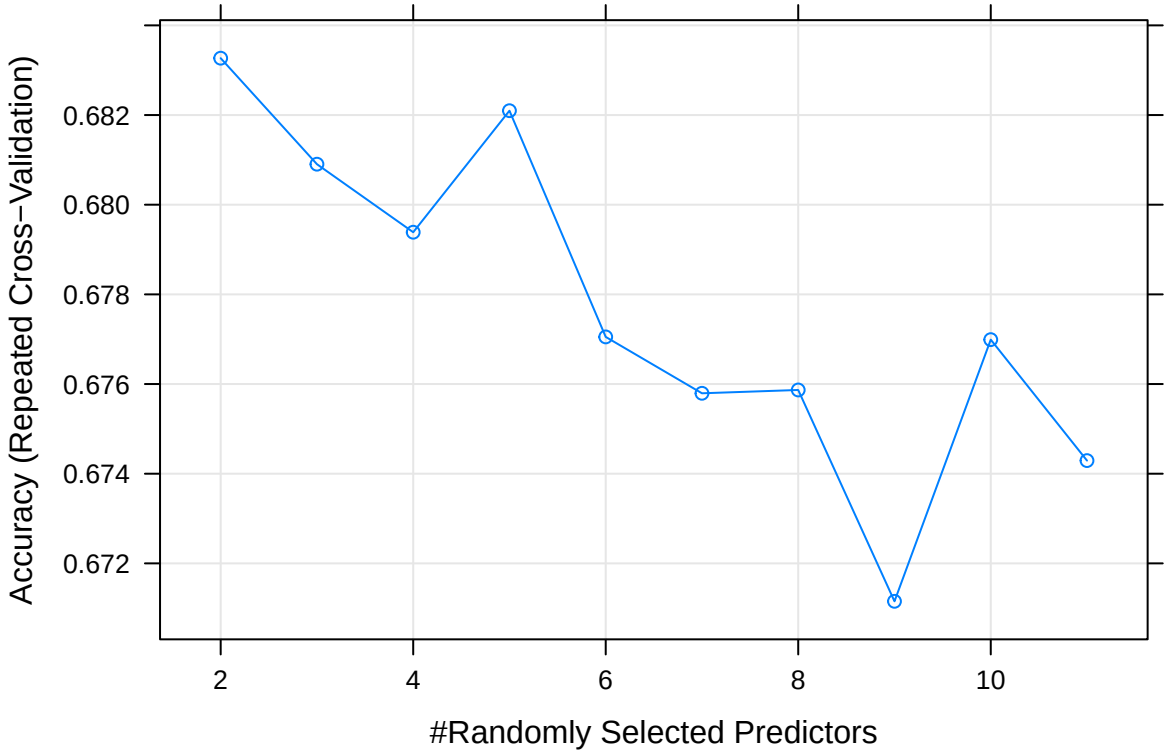
the model again if it is necessary.

Attention: This process may take several minutes to run.

```
#checking if the default mtry is good enough
#it can take several minutes

set.seed(42)
control <- trainControl(method='repeatedcv', number=10, repeats=2)
plot(train(quality.factor~., data=TrainSet[,-c(12)], method="rf", tuneLength=15, trControl=control))
```

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



The best highest accuracy already was achieved in the default parameters and mtry = 2.

6.6. XGBoost

XGBoost is a short form for Extreme Gradient Boosting and is similar to gradient boosting framework but more efficient. It has both linear model solver and tree learning algorithms. XGBoost only works with numeric vectors.

It belongs to a family of boosting algorithms that convert weak learners into strong learners. Boosting is a sequential process; i.e., trees are grown using the information from a previously grown tree one after the other. This process slowly learns from data and tries to improve its prediction in subsequent iterations.

According to the XGBoost website, before running the model we must set three types of parameters: general parameters, booster parameters and task parameters.

- General parameters relate to which booster we are using to do boosting, commonly tree or linear model

- Booster parameters depend on which booster you have chosen
- Learning task parameters decide on the learning scenario. For example, regression tasks may use different parameters with ranking tasks.

I tried several templates and tweaks and the one that worked best was this one below. Here is the explanation of the parameters:

max_depth: Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree

subsample: Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration.

eta: Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.

gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be.

multi:softprob: set XGBoost to do multiclass classification using the softprob objective. It will output a vector of ndata * nclass, which can be further reshaped to ndata * nclass matrix. The result contains predicted probability of each data point belonging to each class (it´s necessary to set the num_class too). I will show this output below.

mlogloss: Logistic loss or cross-entropy loss. This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of a logistic model that returns y_pred probabilities for its training data y_true.

```
set.seed(42)
xgb_train <- xgb.DMatrix(data = as.matrix(TrainSet[1:11]), label = as.integer(TrainSet$quality) - 3)
xgb_test <- xgb.DMatrix(data = as.matrix(TestSet[1:11]), label = as.integer(TestSet$quality) - 3)
xgb_params <- list(
  max_depth = 5,
  subsample = 0.9,
  eta               = 0.08,
  gamma             = 0.7,
  objective = "multi:softprob",
  eval_metric = "mlogloss",
  num_class = length(levels(as.factor(df_red$quality)))
)

xgb_model <- xgb.train(
  params = xgb_params,
  data = xgb_train,
  nrounds = 300,
  verbose = 0
)

xgb_model
```

```
## ##### xgb.Booster
## raw: 2.7 Mb
## call:
##   xgb.train(params = xgb_params, data = xgb_train, nrounds = 300,
##     verbose = 0)
```

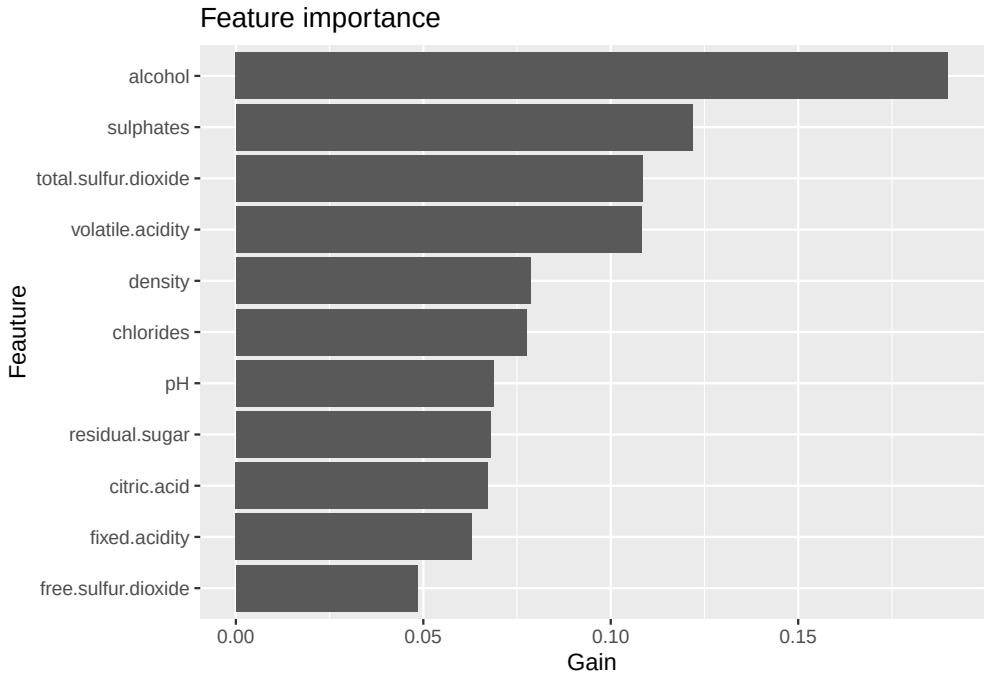
```
## params (as set within xgb.train):
##   max_depth = "5", subsample = "0.9", eta = "0.08", gamma = "0.7", objective = "multi:softprob", eval
## xgb.attributes:
##   niter
## # of features: 11
## niter: 300
## nfeatures : 11
```

6.6.1. XGBoost Importance List

It shows how much each feature contributed to the result.

```
importance_matrix <- xgb.importance(
  feature_names = colnames(xgb_train),
  model = xgb_model
)

ggplot(data=importance_matrix, aes(x=reorder(Feature, Gain), y= Gain )) +
  geom_bar(stat="identity") + coord_flip() +
  labs(title = "Feature importance", x = "Feauture", y = "Gain")
```



6.6.2. XGBoost Prediction

For each record of the test dataset, the percentage of chances that each factor has, according to the model, of being the most adequate answer is shown. In addition, the last two columns show both the predicted value (referring to the one with the highest probability) and the value found in the dataset.

```
set.seed(42)
xgb_preds <- predict(xgb_model, as.matrix(TestSet[1:11]), reshape = TRUE)
xgb_preds <- as.data.frame(xgb_preds)
colnames(xgb_preds) <- levels(as.factor(df_red$quality))
#xgb_preds
```

```r
xgb_preds$PredictedClass <- apply(xgb_preds, 1, function(y) colnames(xgb_preds)[which.max(y)])
xgb_preds$ActualClass <- levels(as.factor(df_red$quality))[as.integer(TestSet$quality) - 3 + 1]
#xgb_preds

head(xgb_preds) %>% mutate(across(where(is.numeric), ~ round(., 4)))
```

```
##        3      4      5      6      7      8 PredictedClass ActualClass
## 1 0.0015 0.0081 0.9387 0.0500 0.0010 0.0006              5           5
## 2 0.0029 0.0307 0.4952 0.4606 0.0091 0.0015              5           7
## 3 0.0012 0.0013 0.9383 0.0557 0.0021 0.0014              5           6
## 4 0.0007 0.0033 0.9246 0.0677 0.0028 0.0009              5           5
## 5 0.0046 0.0181 0.6609 0.3120 0.0031 0.0013              5           5
## 6 0.0015 0.0041 0.9449 0.0482 0.0010 0.0003              5           6
```

```r
cm <- confusionMatrix(factor(xgb_preds$ActualClass), factor(xgb_preds$PredictedClass, levels = 3:8))

cm
```

6.6.3. XGBoost Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   1   1   0   0
##          4   1   0   4   4   0   0
##          5   0   0 112  25   0   0
##          6   0   1  25  85  10   0
##          7   0   0   4  10  32   0
##          8   0   0   0   2   3   0
##
## Overall Statistics
##
##                Accuracy : 0.7156
##                  95% CI : (0.6628, 0.7644)
##     No Information Rate : 0.4562
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5517
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.000000   0.7671   0.6693   0.7111       NA
## Specificity          0.993730 0.971787   0.8563   0.8135   0.9491  0.98438
## Pos Pred Value       0.000000 0.000000   0.8175   0.7025   0.6957       NA
## Neg Pred Value       0.996855 0.996785   0.8142   0.7889   0.9526       NA
## Prevalence           0.003125 0.003125   0.4562   0.3969   0.1406  0.00000
## Detection Rate       0.000000 0.000000   0.3500   0.2656   0.1000  0.00000
## Detection Prevalence 0.006250 0.028125   0.4281   0.3781   0.1437  0.01562
## Balanced Accuracy    0.496865 0.485893   0.8117   0.7414   0.8301       NA
```

XGB achieved 0.7156 of accuracy and the highest sensitivity occurred in grade 5, at 0.7671. In the Confusion Matrix we can see that most of the prediction errors occurred between grades 5, 6 and 7.

7. Conclusion

The objective in this project was to train a machine learning algorithm that predicts the wine quality using the inputs of a train subset and the validation subset.

We could attest that the application of several ML models requires understanding the applicability of each model for each type of dataset and data. It is important to note that for each model it is necessary to model the data so that it best fits each need. For example, in the case of XGBoost, where it is necessary to feed the model with data separately from the labels, as well as defining the number of classes and factors being used.

There is no model that is best in all scenarios and needs. Simpler models can deliver better and faster results in some situations. More complex models and time-consuming tunings can deliver little or no improvement in the result after all, in addition to being necessary to consider the amount of additional processing that will be used.

Regarding the dataset used and the results obtained, it is important to note that there are characteristics that are confused, between different notes of the wines, which makes a better accuracy of the prediction difficult. In addition, the relatively small size of the dataset also influences the difficulty of obtaining better results. The best accuracy (0.7344) was achieved with the Random Forest.

As a suggestion for future works, I think it is worthwhile investing in the search for greater tuning in the models already presented or the use of new approaches. The simplification of the wine grades searched for between 'good'/'bad', among others, is a possibility to deliver greater accuracy according to the interest of the application.

8. References

A.G. Reynolds, 2010

Cortez et al., 2009

Doris Rauhut, Florian Kiene, 2019

Javatpoint

Manish Saraswat

Wine Quality dataset from UCI

XGBoost website