Report

Mehrad Haghshenas

2022-12-20

The document is organized as follows: Section 1 presents the research question; in Section 2, the introduction, project aim, wine data sets, ML models, and variable selection approach is described; Section 3 contains a full detail of the ML techniques and analysis of the results; in Section 4 conclusion are made; finally, in Section 5 the references are given.

Research Question

How precise can the quality of wine be predicted based on the following attributes: "fixed acidity", "volatile acidity", "citric acid", "residual sugar", "chlorides", "free sulfur dioxide", "total sulfur dioxide", "density", "pH", "sulphates", and "alcohol"?

Introduction

The project aims to examine multiple machine learning models where the response is the quality of wine and the predictors are the mentioned variables. The goal is to find the *best* model for prediction and inference of the quality of wine. By doing so, certification entities, wine producers, and even consumers can benefit from such a model. For this aim, two data sets have been used:

- The first data set is the "red.csv" data set which consists of "red wines".
- The second data set is the "white.csv" data set consisting of "white wines".

These data sets were publicly donated on the 7th of October 2009 and are the two most common variants, white and red (rosé is also produced), from the demarcated Portuguese region of vinho verde. Both have 12 columns; namely, *fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, quality.* The *red wine* data set has 1599 **observations**, whereas the *white wine* data set has 4898 **observations**. In the proposal it is written that the data sets contain 1600 rows and 4899 rows respectively. This is because the first row of the data sets are the names of each column. The data were collected from May of 2004 to February of 2007 using only protected designation of origin samples that were tested at the official certification entity (CVRVV). Moreover, this wine accounts for 15% of the total Portuguese production during that time, making this data set large compared to other data sets prepared in this domain. Both data sets can be found on the following website: UCI (University of California, Irvine), Center for Machine Learning & Intelligent Systems

In brief, the task is creating a model to predict the quality of a wine based on the given attributes. In other words, the data sets can be viewed as classification and regression problems. The intention is to answer the following questions:

- *Feature selection*: Which attributes contribute the most to the quality of wine?
  - Selection methods will be used to see whether all input variables are relevant to the quality of the wine. Seeing the predictors, particularly, *fixed acidity*, *volatile acidity*, and *citric acid*, they seem to be correlated. *Free sulfur dioxide* and *total sulfur dioxide* seem to have a correlation as well. Likewise, we are not sure if all input variables are relevant. Therefore, it could be interesting to test feature selection methods.
- *Prediction*: Which machine learning model will give better prediction for the quality of wine?
- *Inference*: Which model will given better intuition for predicting the quality of wine?

To answer these questions, several machine learning models will be applied on the two data sets separately and the results will be compared to each other. We will also see how the models differ for white wine and red wine; specifically, are there any attributes that is more *influential* for the quality of red wine which is not for white wine (or vice versa)?

The list below are the regression machine learning models applied on the data sets. The first one is a linear regression model. Moving on, in the second part, the model is a non-parametric non-linear model (trees). Although, they are less intuitive, they might have higher prediction accuracy. These assumptions will be tested however, with further analysis.

- **multiple linear regression & subset selection linear regression (best subset, forward step-wise, backward step-wise, sequential step-wise)**
- **regression decision trees & tree ensemble methods: bagging and random forests**

In addition, classification techniques will be applied on the data sets. Note that the quality of wine in both data sets is a quantitative variable with integer values ranging from 0 (poor quality) to 10 (excellent quality). However, these values can be treated in two ways: 1 - as continuous values in a regression problem or 2 - as eleven classes for a classification problem. Moreover, in order to apply classification techniques one can apply a threshold and categorize the wines based on their *quality* value. Treating quality as an *ordinal value*, the following methods will be applied.

- **logistic regression**
- **classification decision trees & tree ensemble methods: bagging and random forests.**
- **support vector machines (SVM)**

It is worth mentioning that wine certification is generally assessed by physicochemical and sensory tests. Determination of density, alcohol or pH values, are some of the Physicochemical laboratory tests used for wine certification. With regards to sensory tests, they rely mainly on human experts (in this case, the *median* of evaluations made by at least 3 wine experts was taken for the quality score). However, it should be emphasized that the weakest sense among the five prominent human senses is *taste*. For detailed information on the *background* and *data sets*, please read the *references* and *proposal.*

Machine Learning techniques

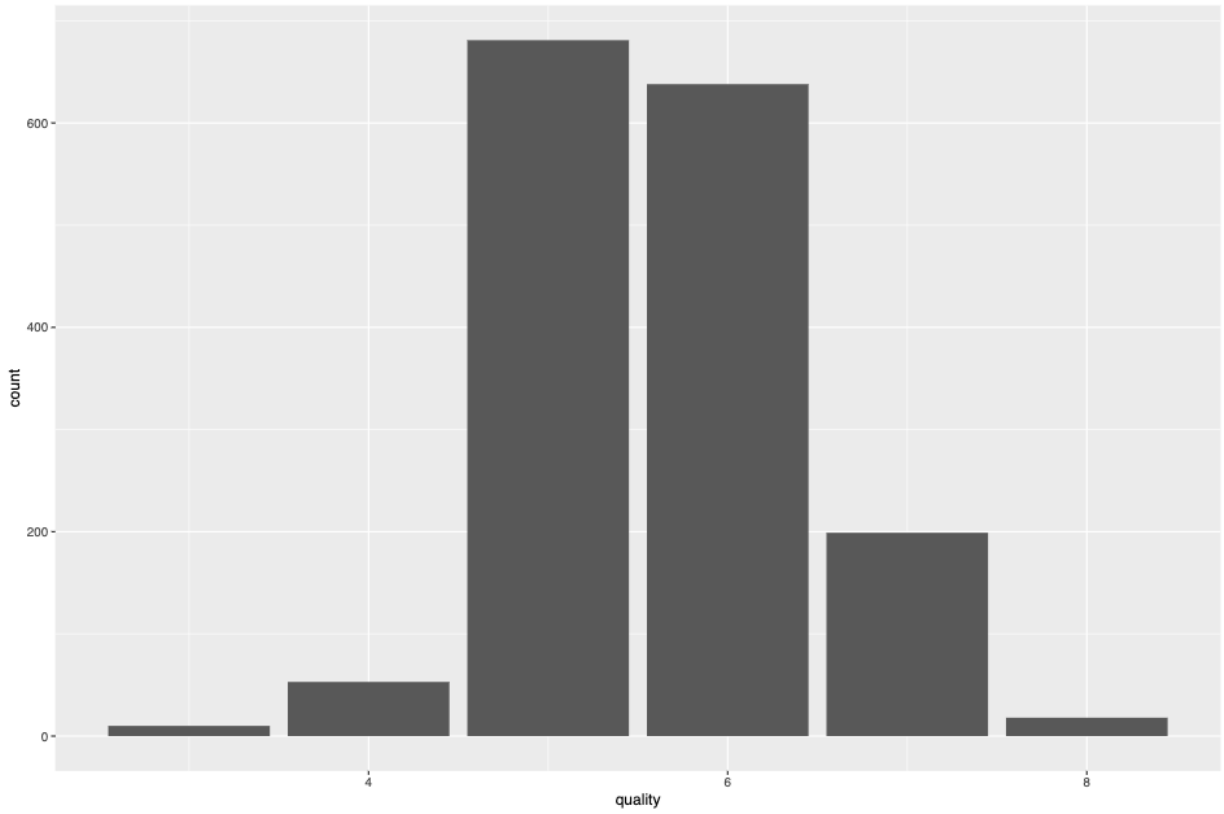
In the next step, the machine learning algorithms will be applied on the data sets. In this section, any data wrangling, tidying, and visualization will be done as needed. At the end of the section, the models will be compared with each other and conclusions will be made.

As the first step, we will read in the data sets.

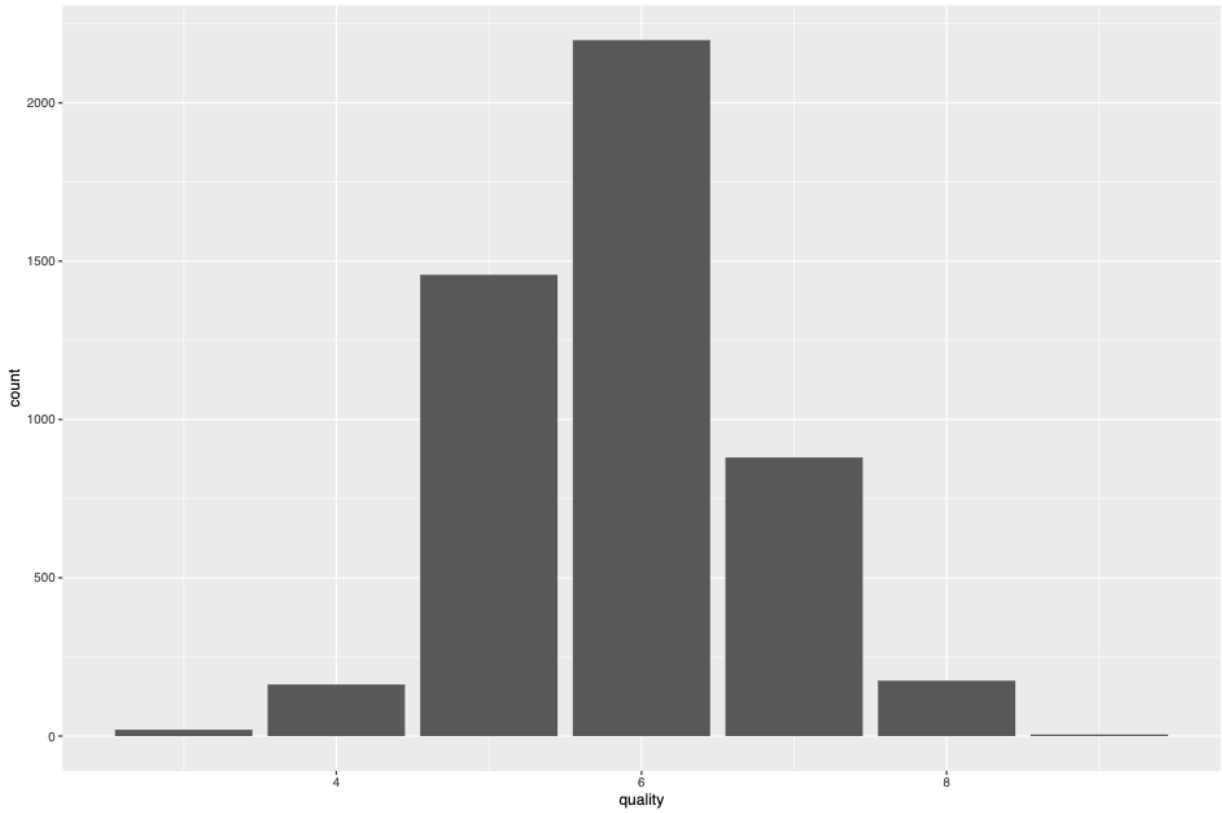
```
library (tidyverse)
red   <- read_csv2("red.csv")
white <- read_csv2("white.csv")
```

In the following, it can be seen that the *quality* variable has a normal distribution for the *red* and *white* data sets. In other words, there are more *normal* wine compared to excellent and very poor wine (as expected). For the *red* data set, the maximum quality is 8 and the minimum is 3. For the *white* data set, the maximum quality is 9 and the minimum is 3. Note that the *quality* variable can take any integer from a scale 0-10.

```
ggplot(data = red) +
  geom_bar(mapping = aes(x = quality))
```




```
ggplot(data = white) +
  geom_bar(mapping = aes(x = quality))
```



Moving on, we will see if the data sets contain any *not found* values.

```
sapply(red, function(x) sum(is.na(x)))
##        fixed acidity     volatile acidity          citric acid
##                    0                    0                    0
##       residual sugar            chlorides  free sulfur dioxide
##                    0                    0                    0
## total sulfur dioxide              density                   pH
##                    2                    0                    0
##            sulphates              alcohol              quality
##                    0                    0                    0
```

```
sapply(white, function(x) sum(is.na(x)))
##        fixed acidity     volatile acidity          citric acid
##                    0                    0                    0
##       residual sugar            chlorides  free sulfur dioxide
##                    0                    0                    0
## total sulfur dioxide              density                   pH
##                    0                    0                    0
##            sulphates              alcohol              quality
##                    0                    0                    0
```

As seen the white data set has no *NA* values where as in the red data set two *NA* values can be seen in the *total sulfur dioxide* column. The two data records containing a *null* value for the total sulfur dioxide column can be neglected given the large number of observations. In other words, when having 1600 observations, two observations can be excluded without any particular loss of information. So we exclude them from the red data set regardless of whether total sulfur dioxide is an influential predictor or not.

```
red <- na.omit(red)
```

Furthermore, looking at the scale of the predictors they are not all on the same scale. As an example, this can be seen in variables *chlorides* and *fixed acidity*:

```
min(red$chlorides)
## [1] "0.012"
max(red$chlorides)
## [1] "0.611"
min(red$`fixed acidity`)
## [1] 5
max(red$`fixed acidity`)
## [1] 159
```

Hence, before applying any machine learning methods, we will feature scale the predictors. By doing so, our models will converge much faster. More importantly, the variables which have larger values do not impact the prediction more than the variables with smaller values. For feature scaling, *standardization* will be used. Before standardizing, all the values of the data set will be converted to numerical values (although all the values are already numerical, but some have been saved as *char* data type. So, further type casting is needed to convert them to numeric data types). Ultimately, take note that it is not needed to scale the response variable (unless you are dealing with very large responses like GPU linear algebra Is It Necessary to Scale the Target Value in Addition to Scaling Features for Regression Analysis?, 2014). In our data set, the response is not large and thus will not be feature scaled. Having said that, it can be scaled if one wishes; nonetheless, in general, when applying feature scaling, the *interpretation* of the model becomes harder. Therefore, where there is no need, it is recommended to not feature scale.

```
red <- as.tibble(sapply(red, as.numeric))
white <- as.tibble(sapply(white, as.numeric))
red <- cbind(scale(red[,-ncol(red)]),red[,ncol(red)])
white <- cbind(scale(white[,-ncol(white)]),white[,ncol(white)])
```

If you look at the summary of the data set now, you can see that each variable has a mean and standard deviation of 0 and 1 respectively:

```
##  fixed acidity      volatile acidity   citric acid      residual sugar
##  Min.   :-2.02259   Min.   :-1.9668   Min.   :-2.7615   Min.   :-1.1418
##  1st Qu.:-0.04967   1st Qu.:-0.6770   1st Qu.:-0.5304   1st Qu.:-0.9250
##  Median : 0.16172   Median :-0.1810   Median :-0.1173   Median :-0.2349
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.37310   3rd Qu.: 0.4143   3rd Qu.: 0.4612   3rd Qu.: 0.6917
##  Max.   :22.99123   Max.   : 8.1528   Max.   :10.9553   Max.   :11.7129
##    chlorides       free sulfur dioxide total sulfur dioxide    density
##  Min.   :-1.6831   Min.   :-0.59860    Min.   :-0.90842     Min.   :-2.31280
##  1st Qu.:-0.4473   1st Qu.:-0.27634    1st Qu.:-0.26434     1st Qu.:-0.77063
##  Median :-0.1269   Median :-0.10754    Median :-0.09519     Median :-0.09608
##  Mean   : 0.0000   Mean   : 0.00000    Mean   : 0.00000     Mean   : 0.00000
##  3rd Qu.: 0.1935   3rd Qu.: 0.07661    3rd Qu.: 0.12438     3rd Qu.: 0.69298
##  Max.   :13.7417   Max.   :21.85193    Max.   :22.87686     Max.   :15.02976
##        pH            sulphates          alcohol            quality
##  Min.   :-3.0856   Min.   :-2.3645   Min.   :-0.05257   Min.   :3.000
##  1st Qu.: 0.1838   1st Qu.:-0.6996   1st Qu.:-0.05257   1st Qu.:5.000
##  Median : 0.3142   Median :-0.1739   Median :-0.05257   Median :6.000
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   :5.878
##  3rd Qu.: 0.4228   3rd Qu.: 0.5271   3rd Qu.:-0.05257   3rd Qu.:6.000
##  Max.   : 1.0310   Max.   : 5.1711   Max.   :28.00913   Max.   :9.000
##  fixed acidity      volatile acidity    citric acid       residual sugar
```

```
##  Min.   :-2.44891   Min.   :-2.27589   Min.   :-1.39360   Min.   :-0.58411
##  1st Qu.:-0.24493   1st Qu.:-0.76865   1st Qu.:-0.93132   1st Qu.:-0.22937
##  Median : 0.06992   Median :-0.04294   Median :-0.05812   Median :-0.11112
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.55970   3rd Qu.: 0.62694   3rd Qu.: 0.76371   3rd Qu.:-0.04018
##  Max.   : 2.93859   Max.   : 5.87437   Max.   : 3.74286   Max.   :14.85879
##    chlorides        free sulfur dioxide total sulfur dioxide
##  Min.   :-1.60231   Min.   :-0.8131     Min.   :-1.2289
##  1st Qu.:-0.37074   1st Qu.:-0.4986     1st Qu.:-0.7426
##  Median :-0.17963   Median :-0.1317     Median :-0.2562
##  Mean   : 0.00000   Mean   : 0.0000     Mean   : 0.0000
##  3rd Qu.: 0.05394   3rd Qu.: 0.2352     3rd Qu.: 0.4733
##  Max.   :11.11684   Max.   :20.3618     Max.   : 7.3735
##     density                pH            sulphates          alcohol
##  Min.   :-3.537031   Min.   :-3.0668   Min.   :-1.9381   Min.   :-0.04321
##  1st Qu.:-0.608105   1st Qu.: 0.1945   1st Qu.:-0.6398   1st Qu.:-0.04321
##  Median : 0.000985   Median : 0.3299   Median :-0.2267   Median :-0.04321
##  Mean   : 0.000000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.578296   3rd Qu.: 0.4341   3rd Qu.: 0.4225   3rd Qu.:-0.04321
##  Max.   : 3.676708   Max.   : 1.0801   Max.   : 7.9172   Max.   :28.32427
##     quality
##  Min.   :3.000
##  1st Qu.:5.000
##  Median :6.000
##  Mean   :5.637
##  3rd Qu.:6.000
##  Max.   :8.000
```

Upon standardizing the predictors, we will now run the pre-mentioned machine learning models in order:

multiple linear regression The first step will be to identify whether there is a relationship between the response and predictors? In other words, are the predictors at all related to the response or not. In this situation, the *null hypothesis* will be that all coefficients are zero. The *alternative hypothesis* will be that at least one of the coefficients (not including the intercept) in the multiple linear regression model is non-zero. To answer this question, F-statistic will be used.

multiple linear regression model for red wine data set:

```
library (ISLR2)
mlr_red <- lm (quality ~ ., data = red)
summary (mlr_red)
##
## Call:
## lm(formula = quality ~ ., data = red)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.81024 -0.45262 -0.04824  0.47109  2.32619
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           5.636819   0.017155 328.589  < 2e-16 ***
## `fixed acidity`       0.020112   0.019512   1.031   0.3028
## `volatile acidity`   -0.166464   0.022709  -7.330 3.64e-13 ***
## `citric acid`         0.113264   0.024767   4.573 5.18e-06 ***
```

```
## `residual sugar`        0.025597   0.017450   1.467   0.1426
## chlorides              -0.152849   0.019366  -7.893 5.47e-15 ***
## `free sulfur dioxide`   0.084741   0.021352   3.969 7.55e-05 ***
## `total sulfur dioxide` -0.178964   0.021582  -8.292 2.35e-16 ***
## density                -0.170220   0.020285  -8.391  < 2e-16 ***
## pH                      0.005365   0.017294   0.310   0.7564
## sulphates               0.210847   0.019623  10.745  < 2e-16 ***
## alcohol                 0.038254   0.017210   2.223   0.0264 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6855 on 1585 degrees of freedom
## Multiple R-squared:  0.2847, Adjusted R-squared:  0.2797
## F-statistic: 57.35 on 11 and 1585 DF,  p-value: < 2.2e-16
```

multiple linear regression model for white wine data set:

```
mlr_white <- lm (quality ~ ., data = white)
summary (mlr_white)
##
## Call:
## lm(formula = quality ~ ., data = white)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2979 -0.5124 -0.0288  0.4809  5.6981
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)             5.877909   0.011015 533.623  < 2e-16 ***
## `fixed acidity`        -0.014940   0.011152  -1.340 0.180401
## `volatile acidity`     -0.178504   0.011269 -15.841  < 2e-16 ***
## `citric acid`           0.021048   0.011381   1.849 0.064466 .
## `residual sugar`        0.521203   0.021389  24.368  < 2e-16 ***
## chlorides              -0.043513   0.011876  -3.664 0.000251 ***
## `free sulfur dioxide`  -0.004912   0.012014  -0.409 0.682667
## `total sulfur dioxide`  0.033612   0.012131   2.771 0.005614 **
## density                -0.708321   0.022299 -31.765  < 2e-16 ***
## pH                      0.013870   0.011064   1.254 0.210061
## sulphates               0.105959   0.011236   9.431  < 2e-16 ***
## alcohol                -0.016973   0.011021  -1.540 0.123617
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7709 on 4886 degrees of freedom
## Multiple R-squared:  0.244,  Adjusted R-squared:  0.2423
## F-statistic: 143.4 on 11 and 4886 DF,  p-value: < 2.2e-16
```

As seen, the F-statistic (57.35) for the red data set and the F-statistic for the white data set (143.4) are much larger than 1 indicating that indeed there *might* be at least one predictor related to the response, *quality*. However, to conform this statement, we use the p-value; which indicates the probability that the F-statistic is higher than 57.35 (for red wine) when all coefficients are zero. If this probability is lower than the significance level (normally defined as 0.05) then it shows that there is under 5 percent chance that the F-statistic will be higher than 57.35 when all coefficients are zero. Thus, this will indicate that the F-statistic is large enough

and that the null-hypothesis can be rejected in favor of the alternative hypothesis. In this case, the p-value for both data sets is approximately 0, hence, indicating that the F-statistics are sufficiently large and that the null hypothesis can be rejected. In other words, we can conclude that there is a correlation between the response and at least one predictor in both data sets (red wine and white wine). One might wonder why weren't the p-values for the t-tests between each predictor and the response used. It is worth mentioning that the p-values for each predictor are obtained using a t-test for the *simple linear regression* which is a model with that particular predictor and the response. The t-value is calculated by dividing the predicted coefficient (slope) for that predictor by the *standard error* of that coefficient ^ [Interestingly, the squared of the t-value is equal to the F-statistics of the multiple regression model when omitting that particular response.]. Whether the t-value is large enough or not will be indicated by whether its p-value is lower than 0.05 or not. If lower, then one can conclude that the predictor and the response are correlated. This might seem logical, but the logic might be flawed especially in *high dimensional data* where the number of predictors are a lot. The reason being is that when having many predictors then the probability of having at least one with p-value under 0.05 increases. In fact, if we have 100 predictors then we expect to see approximately five small p-values even in the absence of any true association between the predictors and the response. However, F-statistics does not suffer from this issue so for concluding for whether there is at least one predictor related to the response F-statistic has been used.

Now that we know that there is at least one predictor which is correlated to the response, we want to find out which predictor(s) hold this correlation. In other words, in this step we will do *feature selection*. Another question arises that why not pick all the predictors even the ones that are less correlated. Correspondingly, one might think that even including the predictors with weaker correlation will result in a more precise and comprehensive model for prediction. This is a naive assumption to make given that it is true that including more predictors will reduce the *training mean squared of errors (MSE)*; but, what is important to us is the *test mean squared of errors (MSE)*. Accordingly, we care about how well our model can perform on unforeseen data because the main goal of machine learning is to build models that can be used for future prediction. There is no point in building a model that only performs well on the trained data set. By applying feature selection, we can reduce the dimensions of the model, leading to lower variance and a more *generalized* model which would not *over fit* the data. Note that if all variables are of importance, in the process of feature selection no features will be discarded.

There are multiple feature selection methods, probably the best (as the name says) is the *best subset selection* method. Below is the best subset selection for the red wine data set:

```
library(leaps)
regfit.full_red <- regsubsets(quality ~ ., red, nvmax = 11, method = "exhaustive")
summary (regfit.full_red)
## Subset selection object
## Call: regsubsets.formula(quality ~ ., red, nvmax = 11, method = "exhaustive")
## 11 Variables  (and intercept)
##                        Forced in Forced out
## `fixed acidity`            FALSE      FALSE
## `volatile acidity`         FALSE      FALSE
## `citric acid`              FALSE      FALSE
## `residual sugar`           FALSE      FALSE
## chlorides                  FALSE      FALSE
## `free sulfur dioxide`      FALSE      FALSE
## `total sulfur dioxide`     FALSE      FALSE
## density                    FALSE      FALSE
## pH                         FALSE      FALSE
## sulphates                  FALSE      FALSE
## alcohol                    FALSE      FALSE
## 1 subsets of each size up to 11
## Selection Algorithm: exhaustive
##          `fixed acidity` `volatile acidity` `citric acid` `residual sugar`
```

```
## 1  ( 1 )  " "             "*"                 " "           " "
## 2  ( 1 )  " "             "*"                 " "           " "
## 3  ( 1 )  " "             "*"                 " "           " "
## 4  ( 1 )  " "             "*"                 " "           " "
## 5  ( 1 )  " "             "*"                 " "           " "
## 6  ( 1 )  " "             "*"                 "*"           " "
## 7  ( 1 )  " "             "*"                 "*"           " "
## 8  ( 1 )  " "             "*"                 "*"           " "
## 9  ( 1 )  " "             "*"                 "*"           "*"
## 10  ( 1 ) "*"             "*"                 "*"           "*"
## 11  ( 1 ) "*"             "*"                 "*"           "*"
##           chlorides `free sulfur dioxide` `total sulfur dioxide` density pH
## 1  ( 1 )  " "       " "                   " "                   " "     " "
## 2  ( 1 )  " "       " "                   " "                   "*"     " "
## 3  ( 1 )  " "       " "                   " "                   "*"     " "
## 4  ( 1 )  "*"       " "                   " "                   "*"     " "
## 5  ( 1 )  "*"       " "                   "*"                   "*"     " "
## 6  ( 1 )  "*"       " "                   "*"                   "*"     " "
## 7  ( 1 )  "*"       "*"                   "*"                   "*"     " "
## 8  ( 1 )  "*"       "*"                   "*"                   "*"     " "
## 9  ( 1 )  "*"       "*"                   "*"                   "*"     " "
## 10  ( 1 ) "*"       "*"                   "*"                   "*"     " "
## 11  ( 1 ) "*"       "*"                   "*"                   "*"     "*"
##           sulphates alcohol
## 1  ( 1 )  " "       " "
## 2  ( 1 )  " "       " "
## 3  ( 1 )  "*"       " "
## 4  ( 1 )  "*"       " "
## 5  ( 1 )  "*"       " "
## 6  ( 1 )  "*"       " "
## 7  ( 1 )  "*"       " "
## 8  ( 1 )  "*"       "*"
## 9  ( 1 )  "*"       "*"
## 10  ( 1 ) "*"       "*"
## 11  ( 1 ) "*"       "*"
```

This is the best subset selection for the white wine data set:

```
regfit.full_white <- regsubsets(quality ~ ., white, nvmax = 11, method = "exhaustive")
summary (regfit.full_white)
## Subset selection object
## Call: regsubsets.formula(quality ~ ., white, nvmax = 11, method = "exhaustive")
## 11 Variables  (and intercept)
##                       Forced in Forced out
## `fixed acidity`           FALSE      FALSE
## `volatile acidity`        FALSE      FALSE
## `citric acid`             FALSE      FALSE
## `residual sugar`          FALSE      FALSE
## chlorides                 FALSE      FALSE
## `free sulfur dioxide`     FALSE      FALSE
## `total sulfur dioxide`    FALSE      FALSE
## density                   FALSE      FALSE
## pH                        FALSE      FALSE
## sulphates                 FALSE      FALSE
```

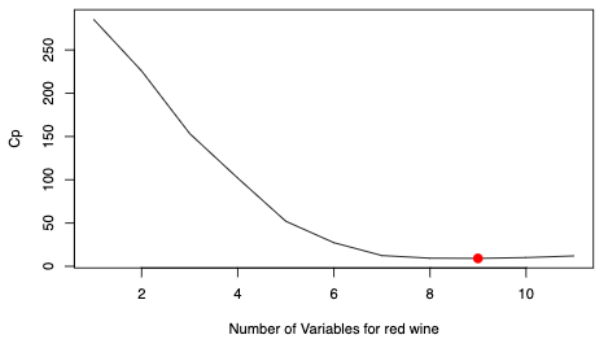
```
## alcohol                    FALSE      FALSE
## 1 subsets of each size up to 11
## Selection Algorithm: exhaustive
##          `fixed acidity` `volatile acidity` `citric acid` `residual sugar`
## 1  ( 1 ) " "             " "                " "           " "
## 2  ( 1 ) " "             " "                " "           "*"
## 3  ( 1 ) " "             "*"                " "           "*"
## 4  ( 1 ) " "             "*"                " "           "*"
## 5  ( 1 ) " "             "*"                " "           "*"
## 6  ( 1 ) " "             "*"                " "           "*"
## 7  ( 1 ) " "             "*"                "*"           "*"
## 8  ( 1 ) " "             "*"                "*"           "*"
## 9  ( 1 ) "*"             "*"                "*"           "*"
## 10  ( 1 ) "*"            "*"                "*"           "*"
## 11  ( 1 ) "*"            "*"                "*"           "*"
##          chlorides `free sulfur dioxide` `total sulfur dioxide` density pH
## 1  ( 1 ) " "       " "                   " "                    "*"     " "
## 2  ( 1 ) " "       " "                   " "                    "*"     " "
## 3  ( 1 ) " "       " "                   " "                    "*"     " "
## 4  ( 1 ) " "       " "                   " "                    "*"     " "
## 5  ( 1 ) "*"       " "                   " "                    "*"     " "
## 6  ( 1 ) "*"       " "                   "*"                    "*"     " "
## 7  ( 1 ) "*"       " "                   "*"                    "*"     " "
## 8  ( 1 ) "*"       " "                   "*"                    "*"     " "
## 9  ( 1 ) "*"       " "                   "*"                    "*"     " "
## 10  ( 1 ) "*"      " "                   "*"                    "*"     "*"
## 11  ( 1 ) "*"      "*"                   "*"                    "*"     "*"
##          sulphates alcohol
## 1  ( 1 ) " "       " "
## 2  ( 1 ) " "       " "
## 3  ( 1 ) " "       " "
## 4  ( 1 ) "*"       " "
## 5  ( 1 ) "*"       " "
## 6  ( 1 ) "*"       " "
## 7  ( 1 ) "*"       " "
## 8  ( 1 ) "*"       "*"
## 9  ( 1 ) "*"       "*"
## 10  ( 1 ) "*"      "*"
## 11  ( 1 ) "*"      "*"
```

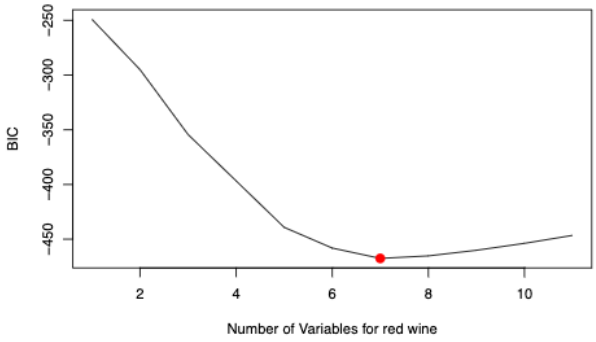
Above, the results for the best subset selection approach can be seen. Other feature selection methods are forward step-wise selection (a greedy approach), backward step-wise selection (does not work on high dimensional data), and hybrid selection (combination of both forward and backward step wise). All these approaches compared to *best subset selection* are less precise. However, when there is a large number of predictors the best subset selection method is computationally expensive and we must use an alternative approach. Nonetheless, given that the wine data set is fairly small, using best subset selection is recommended. It is worth mentioning that the *best subset selection* method takes in all the combinations of predictors and creates a multiple linear regression model for each one. In detail, it first creates all the models with one predictor, calculates the *RSS (Residual Sum of Squares)* for each model and gives out the model with one predictor with the least RSS. In this case, the model for the red wine only contains the *volatile acidity* variable and for white wine contains the *density* variable. Furthermore, it creates all the models with two predictors, calculates the RSS for each one and gives out the model with two predictors with the least RSS. In the red wine case, it consists of *volatile acidity* and *density*. This process will be repeated and in general if the data has p predictors the approach will create 2^p models. Ultimately, we will have p models containing

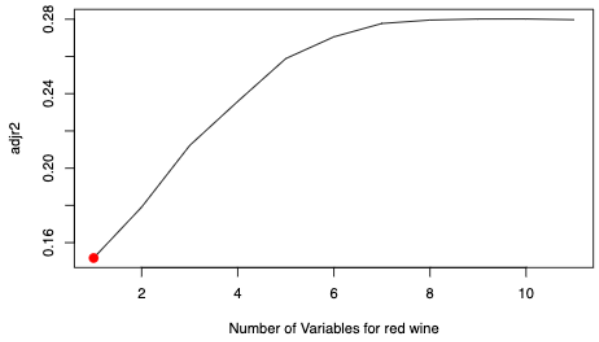
one variable, two predictors, three predictors, ..., p predictors. Having p (11) models with different number of predictors, the previous methods of finding the best model by using the maximum R^2 (or minimum RSS) cannot be used. The main reason is that when adding a feature to the model the RSS will consistently decrease; thus, RSS and R^2 are not suitable for selecting the best model among a collection of models with different numbers of predictors. Should we use RSS we will always get the model with the most number of predictors. With this regard, to compare models with different number of predictors other approaches are used. The four main approaches are: AIC, BIC, Mallows C_p, Adjusted R^2. Although they have different formula, they all have the same reasoning behind them: if a predictor is added to a model, whether the model has improved or not is indicated by a variable which consists of two terms: 1) the RSS which will obviously decrease. 2) a *penalty term* which adds a penalty with every added predictor. Therefore, a model will improve only if the the decrease in RSS is significant enough to compensate for the addition in the penalty term, resulting in an overall decrease. Note that for least regression AIC and C_p are proportional so only one will be displayed.

The below charts indicate the mentioned appraoches for the red wine data set:

```
par(mfrow = c(2, 2))
reg.summary_red <- summary (regfit.full_red)
plot(reg.summary_red$cp, xlab = "Number of Variables for red wine", ylab = "Cp", type = "l")
points(which.min(reg.summary_red$cp), reg.summary_red$cp[which.min(reg.summary_red$cp)], col = "red", c
pch = 20)


plot(reg.summary_red$bic, xlab = "Number of Variables for red wine", ylab = "BIC", type = "l")
points(which.min(reg.summary_red$bic), reg.summary_red$bic[which.min(reg.summary_red$bic)], col = "red"
pch = 20)

plot(reg.summary_red$adjr2, xlab = "Number of Variables for red wine", ylab = "adjr2", type = "l")
points(which.min(reg.summary_red$adjr2), reg.summary_red$adjr2[which.min(reg.summary_red$adjr2)], col =
pch = 20)
```

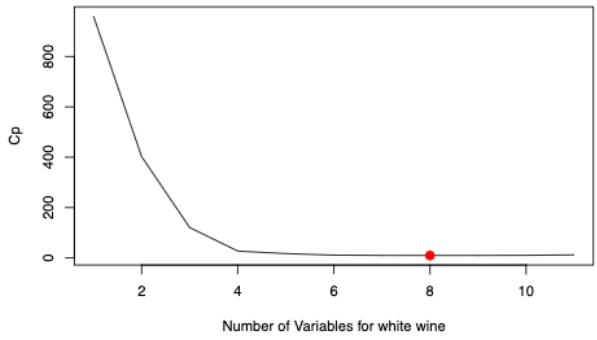


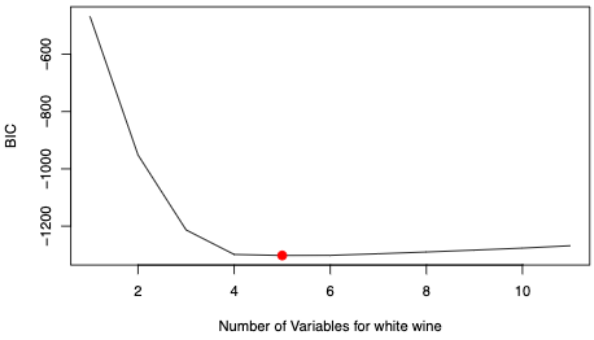


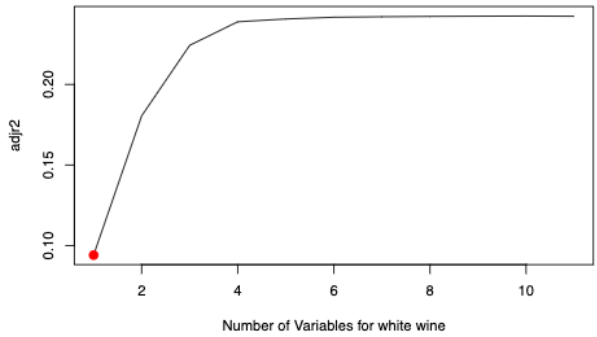


The below charts indicate the mentioned appraoches for the white wine data set:

```
par(mfrow = c(2, 2))
reg.summary_white <- summary (regfit.full_white)
plot(reg.summary_white$cp, xlab = "Number of Variables for white wine", ylab = "Cp", type = "l")
points(which.min(reg.summary_white$cp), reg.summary_white$cp[which.min(reg.summary_white$cp)], col = "r
pch = 20)


plot(reg.summary_white$bic, xlab = "Number of Variables for white wine", ylab = "BIC", type = "l")
points(which.min(reg.summary_white$bic), reg.summary_white$bic[which.min(reg.summary_white$bic)], col =
pch = 20)

plot(reg.summary_white$adjr2, xlab = "Number of Variables for white wine", ylab = "adjr2", type = "l")
points(which.min(reg.summary_white$adjr2), reg.summary_white$adjr2[which.min(reg.summary_white$adjr2)],
pch = 20)
```







red wine: As seen the three approaches have given different results. Particularly, the C_p approach has indicated that the best model has 9 variables. For BIC the predicted number of variables is 7 and for *adjusted* R it is 1 variable. The adjusted R approach, though being logical, does not have a strong theoretical background to support the method compared to BIC and C_p. Moreover, if one looks at the formula of BIC and C_p, BIC penalizes the model more when having more predictors compared to C_P. Specifically, BIC has a penalty term multiplied by $log(n)$ but C_p has a penalty term with 2 as the coefficient. Thus, it is reasonable that the BIC method has given less predictors for the final model. In general researchers prefer the result obtained from BIC as it gives the $TRUE$ model (Is There Any Reason to Prefer the AIC or BIC Over the Other?, 2010).

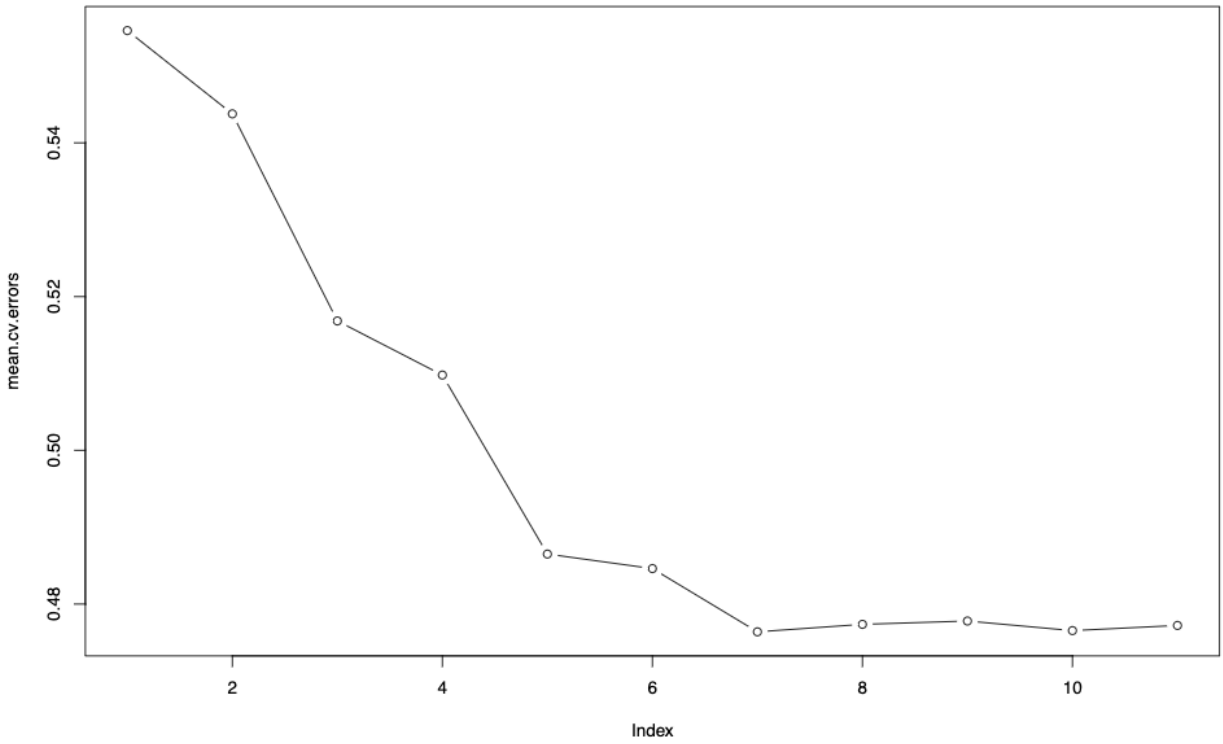
white wine: 8, 5, and 1 predictors were estimated from C_P, BIC, and $adjr2$ respectively.

For further examination 10 fold cross-validation will be used [1]. The advantage cross-validation has over BIC,

[1] bootstrapping can also be examined

C_p, and $adjr2$, is that it does not make any assumptions about the data.

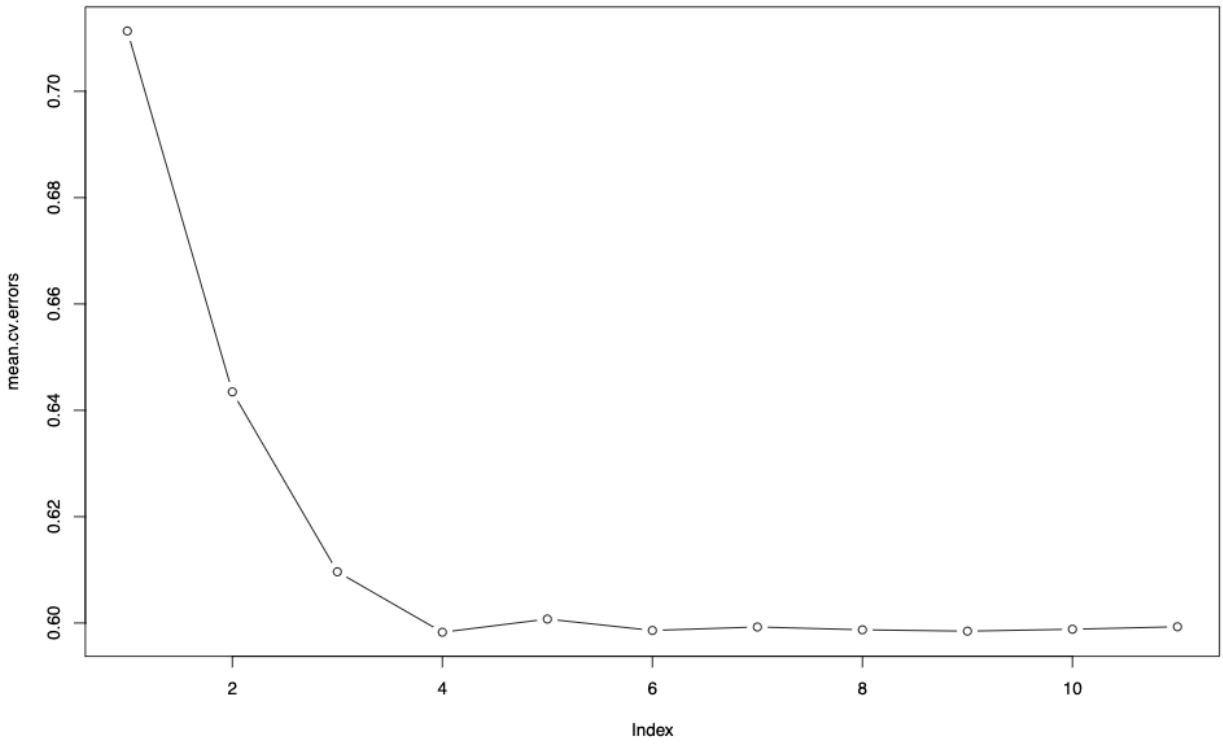
```
k <- 10
n <- nrow(red)
set.seed(11)
folds <- sample(rep(1:k, length = n))
cv.errors <- matrix(data = NA, nrow = k, ncol = 11, dimnames = list(NULL, paste(1:11)))

predict.regsubsets <- function(object, newdata, id, ...) {
  form <- as.formula(object$call[[2]])
  mat <- model.matrix(form, newdata)
  coefi <- coef(object, id = id)
  xvars <- names(coefi)
  mat[, xvars] %*% coefi
  }


for (j in 1:k) {
  best.fit <- regsubsets(quality ~ .,
        data = red[folds != j, ],
        nvmax = 11)
  for (i in 1:11) {
        pred <- predict(best.fit, red[folds == j, ], id = i)
        cv.errors[j, i] <-mean((red$quality[folds == j] - pred)^2)
  }
}
mean.cv.errors <- apply(cv.errors, 2, mean)
plot(mean.cv.errors, type = "b")
```



```
mean.cv.errors
##         1         2         3         4         5         6         7         8
## 0.5546047 0.5437701 0.5168309 0.5097931 0.4865075 0.4846174 0.4763889 0.4773634
##         9        10        11
## 0.4777946 0.4765478 0.4772178
```

Further down for the white wine data set:

```
k <- 10
n <- nrow(white)
set.seed(11)
folds <- sample(rep(1:k, length = n))
cv.errors <- matrix(data = NA, nrow = k, ncol = 11, dimnames = list(NULL, paste(1:11)))

for (j in 1:k) {
  best.fit <- regsubsets(quality ~ .,
        data = white[folds != j, ],
        nvmax = 11)
  for (i in 1:11) {
        pred <- predict(best.fit, white[folds == j, ], id = i)
        cv.errors[j, i] <-mean((white$quality[folds == j] - pred)^2)
  }
}
mean.cv.errors <- apply(cv.errors, 2, mean)
plot(mean.cv.errors, type = "b")
```




```
mean.cv.errors
##         1         2         3         4         5         6         7         8
## 0.7113359 0.6434693 0.6096291 0.5982554 0.6007385 0.5985958 0.5992456 0.5987172
##         9        10        11
## 0.5984637 0.5988243 0.5992879
```

As seen, in the 10-fold cross validation approach, the model with 7 predictors is the *best* (has the lowest *test MSE*) for the red wine and the model with 4 predictors is the best for white wine. The result for red wine is similar to the result obtained from *BIC*. Specifically, the variables *volatile acidity*, *citric acid*, *chlorides*, *free sulfur dioxide*, *total sulfur dioxide*, *density*, and *sulphates* are included in the model. If we refer to the summary of the multiple linear regression model created at the beginning, we can see that the *p-values* for the seven mentioned predictors are much smaller than the other predictors. Particularly, the initial summary already gave us the correct set of predictors, however, not always based on p-values can the best set of predictors be easily picked. For example the result from the cross validation approach for the white wine data gives 4 predictors where as looking at the p-values of the t-tests there are 7 predictors with significant values. Moreover, the result is not exactly like *BIC* given that *BIC* gives 5 predictors but cross validation approach has given 4 predictors. However, looking more closely at BIC graph for white wine, the model with 4 and 5 predictors have roughly the same value. Specifically, the variables *volatile acidity*, *residual sugar*, *density*, and *sulphates* are included in the model for white wine. From here on the machine learning techniques are applied in order. They are only applied for the *red wine* data set. However, the same can be done for the *white wine* data set as well following the same procedure as below.

```
library (boot)
set.seed(11)

cv.error.10 <- rep (0,10)
for (i in 1:10) {
  mlr <- glm (quality ~ ., data = red)
  cv.error.10[i] <- cv.glm (red, mlr, K=10)$delta[1]
}
print ("mean of test MSE from 10 fold cross validation is :")
## [1] "mean of test MSE from 10 fold cross validation is :"
mean (cv.error.10)
## [1] 0.4785797

mlr <- lm (quality ~ ., data = red)
print ("training MSE for model with 11 predictors is: ")
## [1] "training MSE for model with 11 predictors is: "
mean(summary (mlr)$residuals^2)
## [1] 0.4664348

mlr2 <- lm (quality ~ `volatile acidity` + `citric acid` + `chlorides` + `free sulfur dioxide` + `total
print ("training MSE for model with 7 predictors is: ")
## [1] "training MSE for model with 7 predictors is: "
mean(summary (mlr2)$residuals^2)
## [1] 0.4689199
```

Comparing this model (which has 7 predictors) with the initial multiple linear regression model (having 11 predictors), the **training MSE** are 0.4689199 and 0.4664348 respectively. This indicates that the new model has a much lower variance with the cost of a little bias. In other words, in the bias-variance trade off, the model has significantly improved and has a better performance. This can easily be seen with the cross-validation approach where the **test MSE** of the model with 7 predictors (0.4763889) is roughly 0.002 lower than the **test MSE** of the model with 11 predictors (0.4785797). Note that simple models are preferred over complex models if they give the same prediction accuracy. This is because these models do not over fit the data and can be inferred much easier. With regards to this statement, it is said that in the cross-validation approach for the best subset feature selection, it is better to select the smallest model for which the estimated test MSE is within one standard error of the smallest test MSE. The rationale here is that if a set of models appear to be more or less equally good, then we might as well just choose the simplest model, which is the one with the smallest number of predictors (James et al., 2022). Accordingly, we see that the model with 5 predictors will be chosen. The model with 5 predictors consists of the following variables: *volatile acidity*, *chlorides*, *total sulfur dioxide*, *density*, *sulphates*. Note that for the *white* wine data set, even considering this

approach the model will remain the same with four predictors.

```
mean.cv.errors[mean.cv.errors <= min(mean.cv.errors) + sd(mean.cv.errors)]
##         3         4         5         6         7         8         9        10
## 0.6096291 0.5982554 0.6007385 0.5985958 0.5992456 0.5987172 0.5984637 0.5988243
##        11
## 0.5992879
```

However, lets further see the *training MSE* of this model and compare it to the model with 7 predictors.

```
mlr3 <- lm (quality ~ `volatile acidity` + `chlorides` + `total sulfur dioxide` + density + sulphates,
print ("training MSE for model with 5 predictors is: ")
## [1] "training MSE for model with 5 predictors is: "
mean(summary (mlr3)$residuals^2)
## [1] 0.4817588
```

The *training MSE* is 0.13 higher than the corresponding value for the model with 11 predictors. At this point choosing between the model with 11 predictors and 7 predictors depends on the **purpose** of the model. If the model is used for **inference** then the model with 5 variables should be picked. If the model is used for **accurate prediction** then the model with 7 variables should be chosen. In general, personally, *I would choose the model with 5 predictors given the decent accuracy rate it has over the test data set.*

regression decision trees & tree ensemble methods: bagging and random forests. Now we have the five important predictors which are related to the response. Furthermore, we will expand on our linear regression model by adding non-linearity. Specifically, in this part, tree based methods will be applied which are non-linear and non-parametric methods. For this purpose, the *tree* library in *r* is used. Given that the *tree* function is sensitive on the names of the columns, they are changed to contain no spaces. If this was not done, an error would be recieved when calling the *tree* function.

```
red <- red %>%
  rename ( fixed = `fixed acidity`,
           volatile = `volatile acidity`,
           citric = `citric acid`,
           sugar = `residual sugar`,
           free = `free sulfur dioxide`,
           total = `total sulfur dioxide`,


  )
```

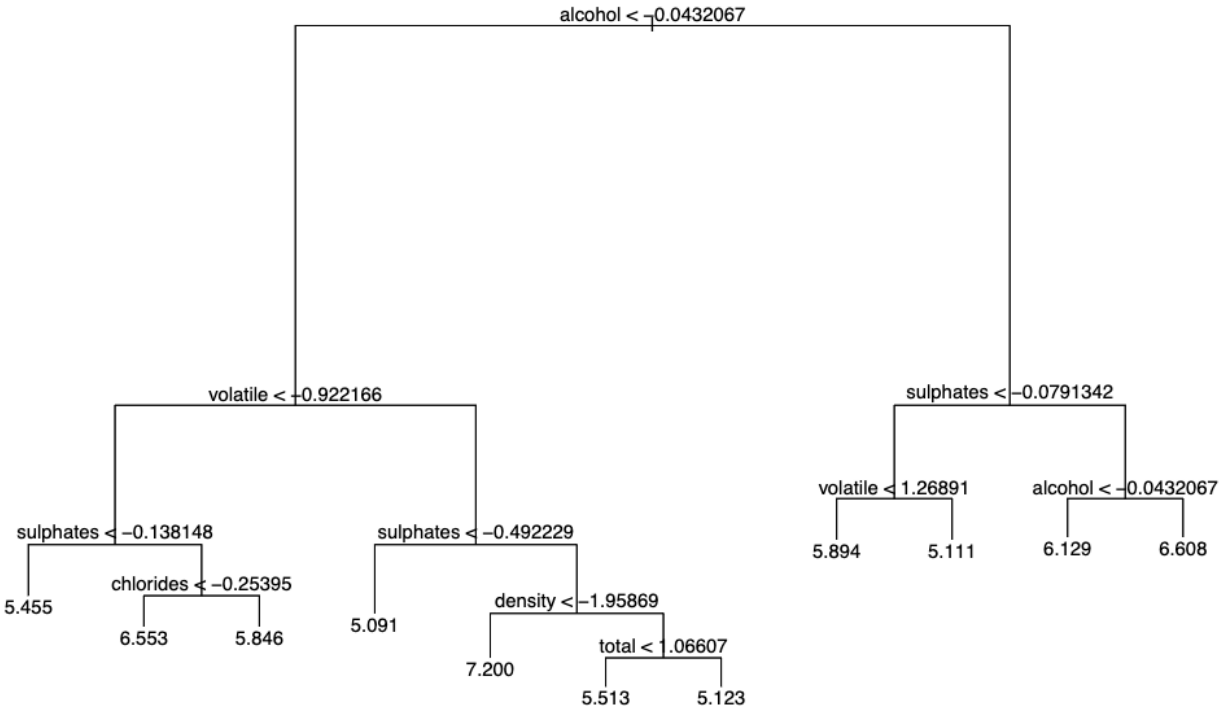
Further on, we create a train data set consisting of 80% of the observations ^ [normally in cross validation the data set is divided into 80-20 or 75-25 for train and test data sets]. As seen, when fitted the tree, the *Residual mean deviance* will be 0.4111. Compared to the *training mean squared of errors* for the multiple linear regression model (0.4664348), this shows that at least the the model is performing better on the trained data set. However, the test MSE is important to us.

```
library (tree)
set.seed(11)
train <- sample(1:nrow(red), 4 * nrow(red) / 5)
tree.red <- tree(formula = red$quality ~ ., data = red, subset = train)
summary(tree.red)
##
## Regression tree:
## tree(formula = red$quality ~ ., data = red, subset = train)
## Variables actually used in tree construction:
## [1] "alcohol"   "volatile"  "sulphates" "chlorides" "density"   "total"
## Number of terminal nodes:  11
```

```
## Residual mean deviance:  0.4111 = 520.4 / 1266
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.51300 -0.51310 -0.09122  0.00000  0.48690  1.90900
```

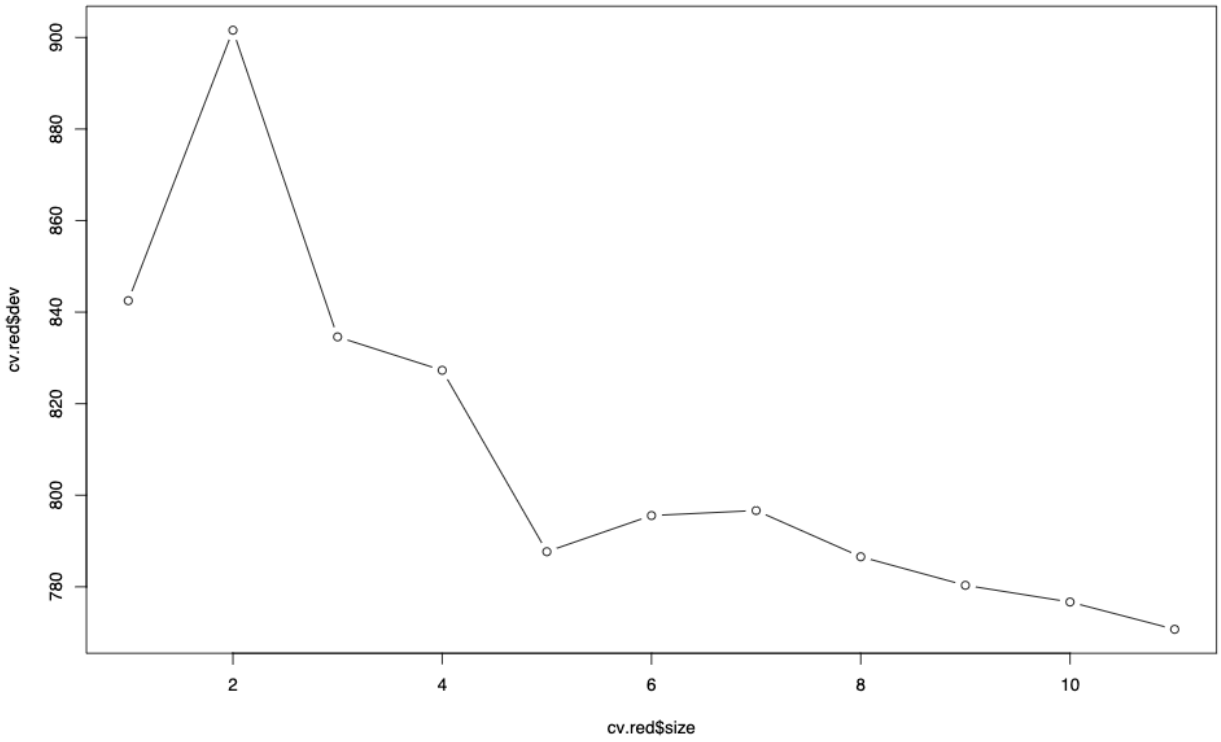
The tree is plotted in the following. It is seen that the most decisive predictor is *alcohol*. This is different from the result obtained from multiple linear regression. Specifically, in the later approach combined with best subset selection *alcohol* was only chosen as the eighth variable for the model.

```
plot (tree.red)
text(tree.red, pretty = 0)
```

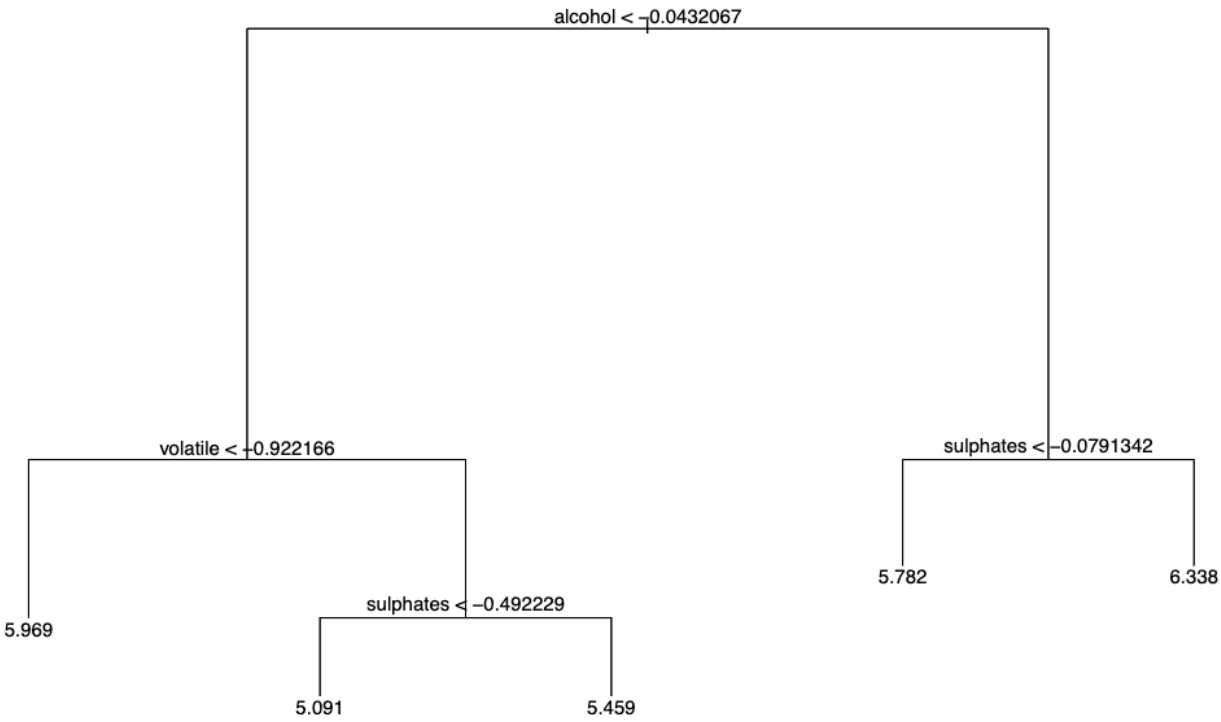


Running a K-fold cross-validation experiment shows that pruning the tree to be of size 5 will indeed result into a better model.

```
cv.red <- cv.tree(tree.red)
plot(cv.red$size, cv.red$dev, type = "b")
```




```
prune.red <- prune.tree(tree.red, best = 5)
plot(prune.red)
text(prune.red, pretty = 0)
```



Particularly, in this pruned tree, the predictors *volatile acidity*, *sulphates*, and *alcohol* are used for classification. As seen below, 0.5878 is the *test MSE* for the tree. Even though the training MSE for the tree was lower

compared to linear regression, the *test MSE* is very high. This shows that the tree-based approach has over fit the data.

```
yhat <- predict(tree.red, newdata = red[-train, ])
red.test <- red[-train,]
mean((yhat - red.test$quality)^2, na.rm = TRUE)
## [1] 0.5878912
```

Moving on, we use ensemble approaches to combine single trees in order to get a better machine learning algorithm. The first ensemble approach is *bagging*.

```
library (randomForest)
set.seed(11)
rf.red <- randomForest(quality ~ ., data = red,
subset = train, mtry = 11, importance = TRUE)
yhat.rf <- predict(rf.red, newdata = red[-train, ])
mean((yhat.rf - red.test$quality)^2)
## [1] 0.3492814
```

As seen the *test MSE* has dramatically decreased compared to using only a single tree and also using a multiple linear regression method. In below, *random forests* have been used which is the same as above, instead in *bagging* all predictors (11) will be used whereas in random forest roughly 1/3 of the predictors (4) will be used.

```
library (randomForest)
set.seed(11)
rf.red <- randomForest(quality ~ ., data = red,
subset = train, mtry = 4, importance = TRUE)
yhat.rf <- predict(rf.red, newdata = red[-train, ])
mean((yhat.rf - red.test$quality)^2)
## [1] 0.3431366
```

Again we see a slight decrease in the test MSE compared to bagging. Overall, the following regression machine learning approaches have been applied: 1) multiple linear regression 2) best subset linear regression 3) decision tree 4) bagging 5) random forest. *Best subset linear regression* is the best for inference, but random forest has the lowest *test MSE* and thus, is better for prediction. Addmitedly, given that it is non-linear and non-parametric it is not very useful for inference.

logistic regression In this section, we will apply logistic regression for the classification of the *quality* variable. As mentioned, this variable holds integer values, but the values can be treated as classes as well. Specifically, for the *red* data set there can be six classes as there are only six distinctive values for quality.

```
red %>%
  distinct (quality)
##   quality
## 1       5
## 2       6
## 3       7
## 4       4
## 5       8
## 6       3
```

First step, by changing the *class* of the *quality* variable, we will convert it to a *qualitative* variable.

```
red <- red %>%
  mutate (quality = as.factor(quality))
```

After creating the qualitative variable, we will fit the logistic regression model. To do so, the *red wine* data

set has been separated to two data sets- training data and test data. More often that not, this partition is done with a ratio of 80% to 20%. Moreover, the multinomial logestic regression model has been trained on the training data and predictions were made on the test data. This process has been done for 10 times and the mean of the model accuracy which is the proportion of test data observations classified correctly has been shown. The reason this has been done for 10 times is that randomly dividing the data set comes with variability. Therefore, the process is repeated and the mean of the accuracy are calculated so to decrease the variability. As seen, roughly 58.069% of the test data sets were classified correctly (when run for the training data set, the model classifies 56.963% of the training data observations correctly).

```
library (caret)
library (nnet)
set.seed(11)

acc <- rep (0,10)
for (i in 1:10){
  training.samples <- red$quality %>%
  createDataPartition(p = 0.8, list = FALSE)
  train.data  <- red[training.samples, ]
  test.data <- red[-training.samples, ]
  # Fit the model
  model <- multinom(quality ~., data = train.data)
  # Make predictions
  predicted.classes <- model %>% predict(train.data)
  # Model accuracy
  acc[i] <- mean(predicted.classes == train.data$quality)
}
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1541.167221
## iter  20 value 1300.781757
## iter  30 value 1265.767375
## iter  40 value 1248.698125
## iter  50 value 1245.362916
## iter  60 value 1244.893794
## final  value 1244.878385
## converged
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1478.917405
## iter  20 value 1277.774918
## iter  30 value 1250.359941
## iter  40 value 1237.390020
## iter  50 value 1234.792265
## iter  60 value 1234.108504
## iter  70 value 1234.031132
## final  value 1234.031108
## converged
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1391.730954
## iter  20 value 1268.144850
## iter  30 value 1242.409489
## iter  40 value 1228.034757
## iter  50 value 1225.259420
```

```
## iter  60 value 1225.227440
## final  value 1225.227139
## converged
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1485.522197
## iter  20 value 1279.111119
## iter  30 value 1239.351657
## iter  40 value 1229.232005
## iter  50 value 1226.214304
## iter  60 value 1226.051331
## final  value 1226.041690
## converged
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1424.951418
## iter  20 value 1282.915620
## iter  30 value 1241.022426
## iter  40 value 1224.370063
## iter  50 value 1221.829451
## iter  60 value 1221.739267
## final  value 1221.737283
## converged
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1493.259024
## iter  20 value 1281.912627
## iter  30 value 1251.067624
## iter  40 value 1238.145260
## iter  50 value 1234.105351
## iter  60 value 1233.835401
## final  value 1233.827141
## converged
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1488.787541
## iter  20 value 1310.647529
## iter  30 value 1259.368693
## iter  40 value 1241.779376
## iter  50 value 1238.630109
## iter  60 value 1238.148909
## final  value 1238.148635
## converged
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1453.729647
## iter  20 value 1298.859292
## iter  30 value 1266.165484
## iter  40 value 1247.935341
## iter  50 value 1244.827909
## iter  60 value 1244.575660
## final  value 1244.574754
## converged
```

```
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1444.918965
## iter  20 value 1295.350517
## iter  30 value 1257.556063
## iter  40 value 1242.761546
## iter  50 value 1239.097454
## iter  60 value 1238.729613
## final  value 1238.718916
## converged
## # weights:  78 (60 variable)
## initial  value 2295.243880
## iter  10 value 1418.227503
## iter  20 value 1291.545955
## iter  30 value 1260.669652
## iter  40 value 1247.897431
## iter  50 value 1244.178754
## iter  60 value 1243.357150
## final  value 1243.332682
## converged
mean (acc)
## [1] 0.5696331
```

However, the question rises that does the model have a high prediction accuracy or is it predicting correctly by chance. This question especially becomes important when looking at the number of observations at each class in the bar charts at the beginning of the report. As seen, they are not equally divided and thus a model only predicting 5 and 6 will also have a good predictive accuracy for the red wine data set.

The below confusion matrix has been drawn. As seen many observations in the test data set have been mis-classified. Specifically, no observation in class 4 or 8 has been classified correctly. There are lots of errors in classes 5 6 and 7 as well.

```
confusionMatrix(predict (model, test.data), test.data$quality,
                mode = "everything",
                positive="1")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  3  4  5  6  7  8
##          3  1  0  0  0  0  0
##          4  0  0  1  0  0  0
##          5  1  5 80 35  2  0
##          6  0  5 53 84 30  0
##          7  0  0  1  8  7  3
##          8  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5443
##                  95% CI : (0.4876, 0.6002)
##     No Information Rate : 0.4272
##     P-Value [Acc > NIR] : 1.861e-05
##
##                   Kappa : 0.2498
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.500000 0.000000   0.5926   0.6614  0.17949 0.000000
## Specificity          1.000000 0.996732   0.7624   0.5344  0.95668 1.000000
## Pos Pred Value       1.000000 0.000000   0.6504   0.4884  0.36842      NaN
## Neg Pred Value       0.996825 0.968254   0.7150   0.7014  0.89226 0.990506
## Precision            1.000000 0.000000   0.6504   0.4884  0.36842       NA
## Recall               0.500000 0.000000   0.5926   0.6614  0.17949 0.000000
## F1                   0.666667      NaN   0.6202   0.5619  0.24138       NA
## Prevalence           0.006329 0.031646   0.4272   0.4019  0.12342 0.009494
## Detection Rate       0.003165 0.000000   0.2532   0.2658  0.02215 0.000000
## Detection Prevalence 0.003165 0.003165   0.3892   0.5443  0.06013 0.000000
## Balanced Accuracy    0.750000 0.498366   0.6775   0.5979  0.56808 0.500000
```

Now we will repeat the process this time with only the five variables which had the most correlation with the response, *quality*. As seen below, the accuracy rate for the test data is 55.089%. This is just under the corresponding percentage for 11 predictors. Thus, with only a bit increase in bias we have much lower variance when using these 5 predictors. However, in general the multinomial logistic regression does not perform well on the data set and has a lot of errors. For example, the *F1-statistic* (which uses precision and recall) for no predictor exceeds 62%.

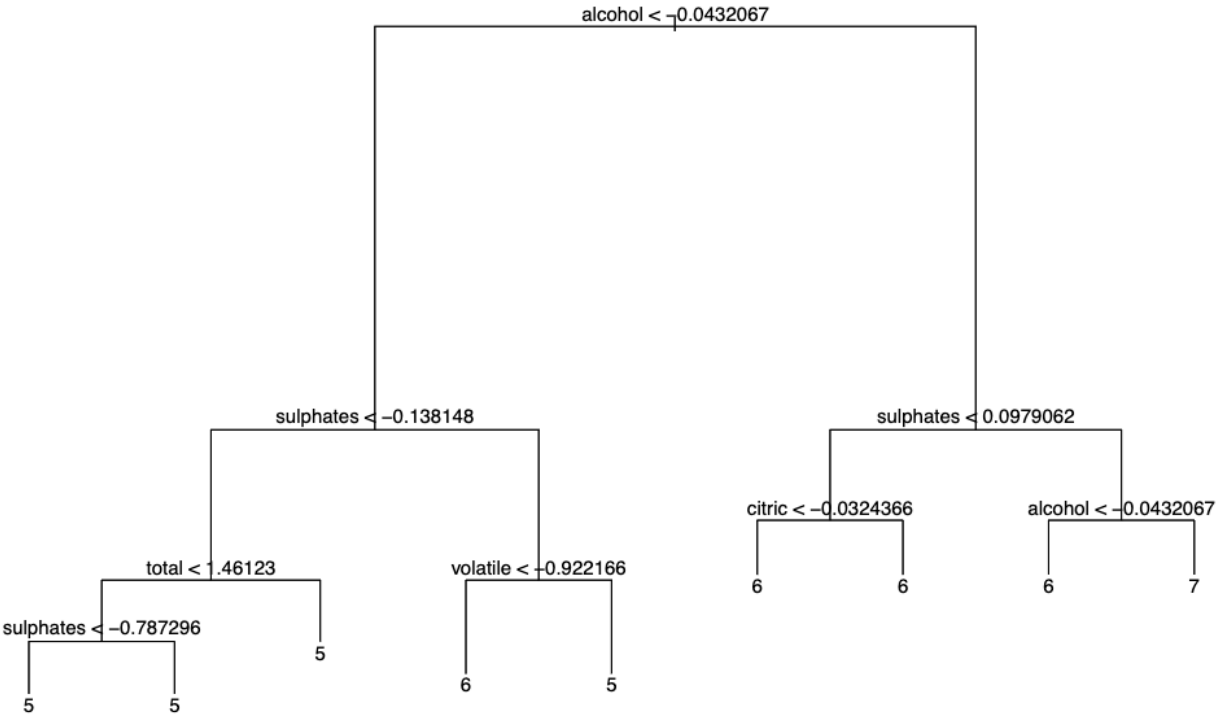
```
library (caret)
library (nnet)
set.seed(11)

acc <- rep (0,10)
for (i in 1:10){
  training.samples <- red$quality %>%
  createDataPartition(p = 0.8, list = FALSE)
  train.data  <- red[training.samples, ]
  test.data <- red[-training.samples, ]
  # Fit the model
  model <- multinom(quality ~ volatile + chlorides + total + density + sulphates, data = train.data)
  # Make predictions
  predicted.classes <- model %>% predict(train.data)
  # Model accuracy
  acc[i] <- mean(predicted.classes == train.data$quality)
}
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1509.602348
## iter  20 value 1343.455691
## iter  30 value 1306.312932
## iter  40 value 1303.802889
## final  value 1303.802254
## converged
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1438.175171
## iter  20 value 1328.817706
## iter  30 value 1281.915946
```

```
## iter  40 value 1280.739260
## final  value 1280.736840
## converged
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1450.689586
## iter  20 value 1326.341893
## iter  30 value 1304.610302
## iter  40 value 1303.302851
## iter  40 value 1303.302842
## iter  40 value 1303.302842
## final  value 1303.302842
## converged
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1475.840661
## iter  20 value 1314.928391
## iter  30 value 1285.598188
## final  value 1285.014852
## converged
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1452.895419
## iter  20 value 1324.330281
## iter  30 value 1298.037947
## iter  40 value 1297.254899
## final  value 1297.254622
## converged
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1457.079971
## iter  20 value 1316.532661
## iter  30 value 1297.217493
## iter  40 value 1296.019064
## final  value 1296.019048
## converged
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1437.817066
## iter  20 value 1314.826605
## iter  30 value 1284.516567
## iter  40 value 1282.488617
## final  value 1282.488450
## converged
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1486.085361
## iter  20 value 1310.859046
## iter  30 value 1287.984553
## iter  40 value 1286.326721
## final  value 1286.326121
## converged
## # weights:  42 (30 variable)
```

```
## initial  value 2295.243880
## iter  10 value 1433.548280
## iter  20 value 1327.991231
## iter  30 value 1293.745672
## iter  40 value 1292.608660
## final  value 1292.608567
## converged
## # weights:  42 (30 variable)
## initial  value 2295.243880
## iter  10 value 1431.591848
## iter  20 value 1312.061208
## iter  30 value 1287.964951
## iter  40 value 1286.525954
## final  value 1286.521757
## converged
mean (acc)
## [1] 0.5508977
```

classification decision trees & tree ensemble methods: bagging and random forests. The first naive approach where all predictors are used is like the following. Just like regression decision trees, *alcohol* is the *root* variable.

```
tree.red2 <- tree(quality ~ . , red)
plot (tree.red2)
text (tree.red2, pretty = 0)
```



The accuracy of such a model on the *test data set* is 0.4844 which is lower than the corresponding value in the multinomial logistic regression. Even the accuracy for the training data is lower (0.5043).

```
set.seed(11)
train <- sample(1:nrow(red), 4 *nrow(red) / 5)
```

```
red.test <- red[-train, ]
tree.red <- tree(quality ~ ., red,
subset = train)
tree.pred <- predict(tree.red, red.test,
type = "class")
confusionMatrix(tree.pred, red.test$quality,
                mode = "everything",
                positive="1")
## 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  2  7 82 66  9  1
##          6  0  3 49 73 23  5
##          7  0  0  0  0  0  0
##          8  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.4844
##                  95% CI : (0.4284, 0.5406)
##     No Information Rate : 0.4344
##     P-Value [Acc > NIR] : 0.04058
##
##                   Kappa : 0.109
##
##  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   0.6260   0.5252      0.0  0.00000
## Specificity           1.00000  1.00000   0.5503   0.5580      1.0  1.00000
## Pos Pred Value            NaN      NaN   0.4910   0.4771      NaN      NaN
## Neg Pred Value        0.99375  0.96875   0.6797   0.6048      0.9  0.98125
## Precision                 NA       NA   0.4910   0.4771       NA       NA
## Recall                0.00000  0.00000   0.6260   0.5252      0.0  0.00000
## F1                        NA       NA   0.5503   0.5000       NA       NA
## Prevalence            0.00625  0.03125   0.4094   0.4344      0.1  0.01875
## Detection Rate        0.00000  0.00000   0.2562   0.2281      0.0  0.00000
## Detection Prevalence  0.00000  0.00000   0.5219   0.4781      0.0  0.00000
## Balanced Accuracy     0.50000  0.50000   0.5881   0.5416      0.5  0.50000
```

Moving on, we use ensemble approaches to combine single trees in order to get a better machine learning algorithm. The first ensemble approach is *bagging*.

```
library (randomForest)
set.seed(11)
rf.red <- randomForest(quality ~ ., data = red,
subset = train, mtry = 11, importance = TRUE)
yhat.rf <- predict(rf.red, newdata = red[-train, ], type = "class")
confusionMatrix(yhat.rf, red.test$quality,
```

```
               mode = "everything",
               positive="1")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   0   0   1   0   0   0
##          5   1   6 104  35   3   0
##          6   1   4  25  92   9   4
##          7   0   0   1  12  20   2
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.675
##                  95% CI : (0.6207, 0.726)
##     No Information Rate : 0.4344
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4716
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000 0.000000   0.7939   0.6619   0.6250  0.00000
## Specificity           1.00000 0.996774   0.7619   0.7624   0.9479  1.00000
## Pos Pred Value            NaN 0.000000   0.6980   0.6815   0.5714      NaN
## Neg Pred Value        0.99375 0.968652   0.8421   0.7459   0.9579  0.98125
## Precision                  NA 0.000000   0.6980   0.6815   0.5714       NA
## Recall                0.00000 0.000000   0.7939   0.6619   0.6250  0.00000
## F1                         NA      NaN   0.7429   0.6715   0.5970       NA
## Prevalence            0.00625 0.031250   0.4094   0.4344   0.1000  0.01875
## Detection Rate        0.00000 0.000000   0.3250   0.2875   0.0625  0.00000
## Detection Prevalence  0.00000 0.003125   0.4656   0.4219   0.1094  0.00000
## Balanced Accuracy     0.50000 0.498387   0.7779   0.7122   0.7865  0.50000
```

As seen the *test prediction accuracy* has dramatically increased compared to using only a single tree and also compared to multiple linear regression methods. Particularly, it is 0.675 which is much larger than the accuracy for a single decxision tree. In below, *random forests* have been used which is the same as above, instead in *bagging* all predictors (11) will be used whereas in random forest roughly 1/3 of the predictors (4) will be used.

```
set.seed(11)
rf.red <- randomForest(quality ~ ., data = red,
subset = train, mtry = 4, importance = TRUE)
yhat.rf <- predict(rf.red, newdata = red[-train, ], type = "class")
confusionMatrix(yhat.rf, red.test$quality,
               mode = "everything",
               positive="1")
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   1   0   0   0   0   0
##          5   1   6 105  34   2   0
##          6   0   4  24  95  10   4
##          7   0   0   2  10  20   2
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6875
##                  95% CI : (0.6336, 0.7379)
##     No Information Rate : 0.4344
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.491
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000 0.000000   0.8015   0.6835   0.6250  0.00000
## Specificity           1.00000 0.996774   0.7725   0.7680   0.9514  1.00000
## Pos Pred Value            NaN 0.000000   0.7095   0.6934   0.5882      NaN
## Neg Pred Value        0.99375 0.968652   0.8488   0.7596   0.9580  0.98125
## Precision                 NA 0.000000   0.7095   0.6934   0.5882       NA
## Recall                0.00000 0.000000   0.8015   0.6835   0.6250  0.00000
## F1                        NA      NaN   0.7527   0.6884   0.6061       NA
## Prevalence            0.00625 0.031250   0.4094   0.4344   0.1000  0.01875
## Detection Rate        0.00000 0.000000   0.3281   0.2969   0.0625  0.00000
## Detection Prevalence  0.00000 0.003125   0.4625   0.4281   0.1062  0.00000
## Balanced Accuracy     0.50000 0.498387   0.7870   0.7257   0.7882  0.50000
```

Again a slight increase in the test accuracy prediction is seen in the *random forest* method (just like regression) with 68.75% prediction accuracy.

support vector machines (SVM) – linear & polynomial & radial In the last subsection we will run an SVM algorithm first with a *linear kernel*, followed by a *polynomial kernel* and finally a *radial kernel*. It is worth metnioning that the cost has been defined as 10 through trial and error this value has approved sufficient. However, a more systematic way might be to do cross validation.

```
library (e1071)
out <- svm(quality ~ ., data = red[train,], kernel = "linear", cost = 10)
summary (out)
##
## Call:
## svm(formula = quality ~ ., data = red[train, ], kernel = "linear",
##     cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
```

```
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  1114
##
##  ( 473 413 165 12 43 8 )
##
##
## Number of Classes:  6
##
## Levels:
##  3 4 5 6 7 8
```

```
confusionMatrix(out$fitted, red[train,]$quality)
## 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   7  32 386 173  17   1
##          6   1  11 161 326 143  11
##          7   0   0   1   0   7   0
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.563
##                  95% CI : (0.5353, 0.5905)
##     No Information Rate : 0.4291
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2623
##
##  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   0.7044   0.6533 0.041916 0.000000
## Specificity          1.000000  1.00000   0.6845   0.5797 0.999099 1.000000
## Pos Pred Value            NaN      NaN   0.6266   0.4992 0.875000      NaN
## Neg Pred Value       0.993735  0.96633   0.7549   0.7228 0.873916 0.990603
## Prevalence           0.006265  0.03367   0.4291   0.3908 0.130775 0.009397
## Detection Rate       0.000000  0.00000   0.3023   0.2553 0.005482 0.000000
## Detection Prevalence 0.000000  0.00000   0.4824   0.5114 0.006265 0.000000
## Balanced Accuracy    0.500000  0.50000   0.6944   0.6165 0.520508 0.500000
```

With regards to SVM with a *linear* kernel, the accuracy for classifying the training data based on quality is 0.563. This number for the test data set is 0.5344.

```
yhat.rf <- predict(out, newdata = red[-train, ], type = "class")

confusionMatrix(yhat.rf, red.test$quality,
                mode = "everything",
```

```
               positive="1")
## 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  2  9 90 57  3  0
##          6  0  1 40 81 29  6
##          7  0  0  1  1  0  0
##          8  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5344
##                  95% CI : (0.4781, 0.59)
##     No Information Rate : 0.4344
##     P-Value [Acc > NIR] : 0.000204
##
##                   Kappa : 0.1976
##
##  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   0.6870   0.5827  0.00000  0.00000
## Specificity           1.00000  1.00000   0.6243   0.5801  0.99306  1.00000
## Pos Pred Value            NaN      NaN   0.5590   0.5159  0.00000      NaN
## Neg Pred Value        0.99375  0.96875   0.7421   0.6442  0.89937  0.98125
## Precision                  NA       NA   0.5590   0.5159  0.00000       NA
## Recall                0.00000  0.00000   0.6870   0.5827  0.00000  0.00000
## F1                         NA       NA   0.6164   0.5473      NaN       NA
## Prevalence            0.00625  0.03125   0.4094   0.4344  0.10000  0.01875
## Detection Rate        0.00000  0.00000   0.2812   0.2531  0.00000  0.00000
## Detection Prevalence  0.00000  0.00000   0.5031   0.4906  0.00625  0.00000
## Balanced Accuracy     0.50000  0.50000   0.6557   0.5814  0.49653  0.50000
```

```
out <- svm(quality ~ ., data = red[train,], kernel = "polynomial", cost = 10)
summary (out)
##
## Call:
## svm(formula = quality ~ ., data = red[train, ], kernel = "polynomial",
##     cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  1101
```

```
##
##  ( 463 425 150 12 43 8 )
##
##
## Number of Classes:  6
##
## Levels:
##  3 4 5 6 7 8
```

```
confusionMatrix(out$fitted, red[train,]$quality)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   4   0   0   0   0   0
##          4   0   9   0   0   0   0
##          5   2  19 420 123  10   0
##          6   2  15 126 364  78   7
##          7   0   0   2  12  79   1
##          8   0   0   0   0   0   4
##
## Overall Statistics
##
##                Accuracy : 0.6891
##                  95% CI : (0.6629, 0.7144)
##     No Information Rate : 0.4291
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4954
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.500000 0.209302   0.7664   0.7295  0.47305 0.333333
## Specificity          1.000000 1.000000   0.7888   0.7069  0.98649 1.000000
## Pos Pred Value       1.000000 1.000000   0.7317   0.6149  0.84043 1.000000
## Neg Pred Value       0.996858 0.973186   0.8179   0.8029  0.92561 0.993716
## Prevalence           0.006265 0.033673   0.4291   0.3908  0.13078 0.009397
## Detection Rate       0.003132 0.007048   0.3289   0.2850  0.06186 0.003132
## Detection Prevalence 0.003132 0.007048   0.4495   0.4636  0.07361 0.003132
## Balanced Accuracy    0.750000 0.604651   0.7776   0.7182  0.72977 0.666667
```

```
yhat.rf <- predict(out, newdata = red[-train, ], type = "class")
```

```
confusionMatrix(yhat.rf, red.test$quality,
                mode = "everything",
                positive="1")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  3  4  5  6  7  8
```

```
##          3  2  0  1  0  0  0
##          4  0  1  5  1  0  0
##          5  0  5 76 51  3  1
##          6  0  4 45 76 23  2
##          7  0  0  4 11  6  3
##          8  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5031
##                  95% CI : (0.447, 0.5592)
##     No Information Rate : 0.4344
##     P-Value [Acc > NIR] : 0.007848
##
##                   Kappa : 0.191
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          1.000000 0.100000   0.5802   0.5468  0.18750  0.00000
## Specificity          0.996855 0.980645   0.6825   0.5912  0.93750  1.00000
## Pos Pred Value       0.666667 0.142857   0.5588   0.5067  0.25000      NaN
## Neg Pred Value       1.000000 0.971246   0.7011   0.6294  0.91216  0.98125
## Precision            0.666667 0.142857   0.5588   0.5067  0.25000       NA
## Recall               1.000000 0.100000   0.5802   0.5468  0.18750  0.00000
## F1                   0.800000 0.117647   0.5693   0.5260  0.21429       NA
## Prevalence           0.006250 0.031250   0.4094   0.4344  0.10000  0.01875
## Detection Rate       0.006250 0.003125   0.2375   0.2375  0.01875  0.00000
## Detection Prevalence 0.009375 0.021875   0.4250   0.4688  0.07500  0.00000
## Balanced Accuracy    0.998428 0.540323   0.6313   0.5690  0.56250  0.50000
```

With regards to SVM with a *polynomial* kernel, the accuracy for classifying the training data based on quality is 0.6891 This number for the test data set is 0.5031. These values indicate that the model overfits the data.

Ultimately, below SVM with a *radial* kernel can be seen:

```
out <- svm(quality ~ ., data = red[train,], kernel = "radial", cost = 10)
summary (out)
##
## Call:
## svm(formula = quality ~ ., data = red[train, ], kernel = "radial",
##     cost = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  1061
##
##  ( 451 396 151 12 43 8 )
##
```

```
##
## Number of Classes:  6
##
## Levels:
##  3 4 5 6 7 8
```

```
confusionMatrix(out$fitted, red[train,]$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   2   0   0   0   0   0
##          4   0   9   0   0   0   0
##          5   4  24 453 112   8   0
##          6   2  10  90 375  69   8
##          7   0   0   5  12  90   1
##          8   0   0   0   0   0   3
##
## Overall Statistics
##
##                Accuracy : 0.7298
##                  95% CI : (0.7046, 0.754)
##     No Information Rate : 0.4291
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5623
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.250000 0.209302   0.8266   0.7515  0.53892 0.250000
## Specificity          1.000000 1.000000   0.7970   0.7699  0.98378 1.000000
## Pos Pred Value       1.000000 1.000000   0.7537   0.6769  0.83333 1.000000
## Neg Pred Value       0.995294 0.973186   0.8595   0.8285  0.93413 0.992936
## Prevalence           0.006265 0.033673   0.4291   0.3908  0.13078 0.009397
## Detection Rate       0.001566 0.007048   0.3547   0.2937  0.07048 0.002349
## Detection Prevalence 0.001566 0.007048   0.4706   0.4338  0.08457 0.002349
## Balanced Accuracy    0.625000 0.604651   0.8118   0.7607  0.76135 0.625000
```

```
yhat.rf <- predict(out, newdata = red[-train, ], type = "class")
```

```
confusionMatrix(yhat.rf, red.test$quality,
                mode = "everything",
                positive="1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  1  1  4  1  0  0
##          5  1  6 91 49  2  1
```

```
##          6  0  3 31 77 18  1
##          7  0  0  5 12 12  4
##          8  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5656
##                  95% CI : (0.5094, 0.6207)
##     No Information Rate : 0.4344
##     P-Value [Acc > NIR] : 1.624e-06
##
##                   Kappa : 0.3001
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000 0.100000   0.6947   0.5540   0.3750  0.00000
## Specificity           1.00000 0.980645   0.6878   0.7072   0.9271  1.00000
## Pos Pred Value            NaN 0.142857   0.6067   0.5923   0.3636      NaN
## Neg Pred Value        0.99375 0.971246   0.7647   0.6737   0.9303  0.98125
## Precision                 NA 0.142857   0.6067   0.5923   0.3636       NA
## Recall                0.00000 0.100000   0.6947   0.5540   0.3750  0.00000
## F1                        NA 0.117647   0.6477   0.5725   0.3692       NA
## Prevalence            0.00625 0.031250   0.4094   0.4344   0.1000  0.01875
## Detection Rate        0.00000 0.003125   0.2844   0.2406   0.0375  0.00000
## Detection Prevalence  0.00000 0.021875   0.4688   0.4062   0.1031  0.00000
## Balanced Accuracy     0.50000 0.540323   0.6912   0.6306   0.6510  0.50000
```

Conclusions

- the main attributes correlated to the *quality* of *white wine* are *volatile acidity*, *residual sugar*, *density*, and *sulphates*.
- the main attributes correlated to the *quality* of *red wine* are *volatile acidity*, *chlorides*, *total sulfur dioxide*, *density*, and *sulphates*.
- *density*, *sulphates*, and *volatile acidity* are the main attributes which are correlated to the quality of both wines.
- Among the regression models used in this project, the multiple linear regression with 5 predictors has a reasonable prediction accuracy and is best used for *inference*.
- Among the regression models, the *random forests* is the best model for prediction.
- Should the quality be treated as a qualitative variable, again using *random forest* has proved to give a better accuracy compared to other approaches with roughly 6. This is contrary to the finding to the article Modeling wine preferences by data mining from physicochemical properties where the authors found *SVM* superior.

References

1. 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.
2. Kaggle: Your Machine Learning and Data Science Community. (n.d.).
3. UCI Machine Learning Repository: Wine Quality Data Set. (n.d.).

Final note There can be much more detailed analysis done on the data set (like applying reinforcement learning, KNN, and neural networks (NN)), but we will stop at this point.