MSDS 680 – Machine Learning
Alla Topp

Predicting Quality of Wine

Objective is to predict wine quality ranking from its chemical properties. This provide guidance to vineyards regarding quality of wine and price expected without heavy reliance on the tasters applying Decision Trees Algorithm.

First we'll upload the data and explore what it looks like from the website: https://archive.ics.uci.edu/ml/datasets/wine+quality . Also, in this assignment we will need to install and use a few packages like:

```
#Installing packages and libraries
install.packages("rpart")
install.packages("rpart.plot")
install.packages("caret")
install.packages("randomForest")

library(randomForest)
library(rpart)
library(rpart.plot)
library(caret)
```

```
#Importing the dataset
wine <- read.csv(url("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequa
                 header = TRUE, sep = ";")
```

Next step is some data exploration:

```
> str(wine)
'data.frame':   1599 obs. of  12 variables:
 $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
 $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
 $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
 $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
 $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
 $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
 $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
 $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
 $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
 $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
 $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
 $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
> table(wine$quality)

  3   4   5   6   7   8
 10  53 681 638 199  18
```

The data set contains 1599 observations of 12 variables. 11 variables are numeric, and the wine quality variable is an integer rating - all wines are rated as an integer ranging from 3 to 8.

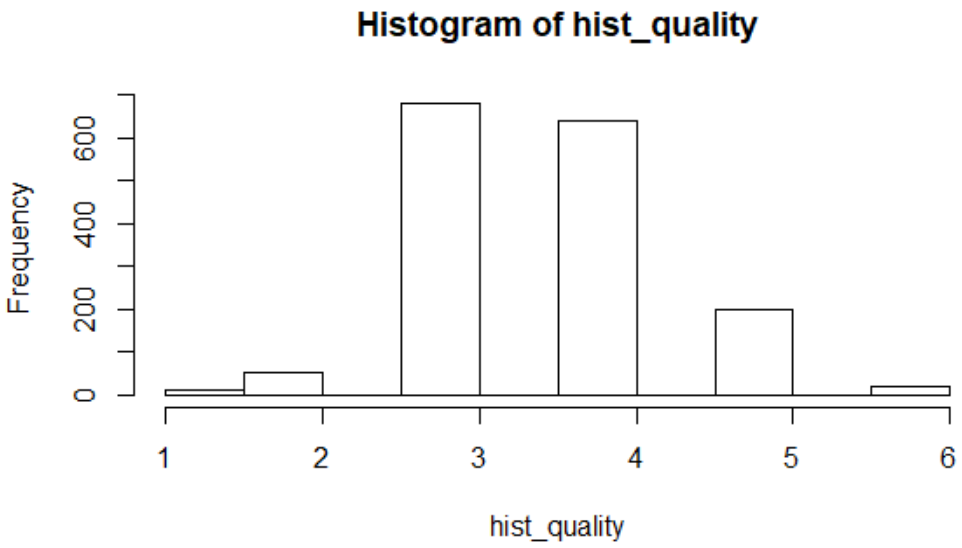
```
> names(wine)
 [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"          "residual.sugar"       "chlorides"

 [6] "free.sulfur.dioxide"  "total.sulfur.dioxide" "density"              "pH"                   "sulphates"

[11] "alcohol"              "quality"
> sum(is.na(wine))
[1] 0
```

The names look sufficient, and there are zero NA values, let's change the predictor variable "quality" to a factor.

```
> wine$quality <- as.factor(wine$quality) #changing predictor quakity to a factor
> str(wine$quality)
 Factor w/ 6 levels "3","4","5","6",..: 3 3 3 4 3 3 3 5 5 3 ...
>
```

Let's first look at a histogram of the frequency of wine quality ratings. It should be mentioned that the levels of the histogram don't represent the integers in the data frame, but instead the 6 levels that're used.

```
hist_quality <- as.numeric(wine$quality)
hist(hist_quality)
```



The majority of ratings are levels 3 and 4, which would be ratings 5 and 6 in the data frame.

We will first use the decision tree classification method for classifying wine into the 6 levels based on its properties. We'll use the rpart() library to classify. The first step is to split the data into training and testing sets. To be safe, we'll randomize these samples. We will use 80% of the data for training and 20% for testing.

To figure out 80% of 1599 we should do a quick calculation: 0.7*1599=1119.3 would be 70% training set and the rest in test set. Below we can see that 30% is 480.

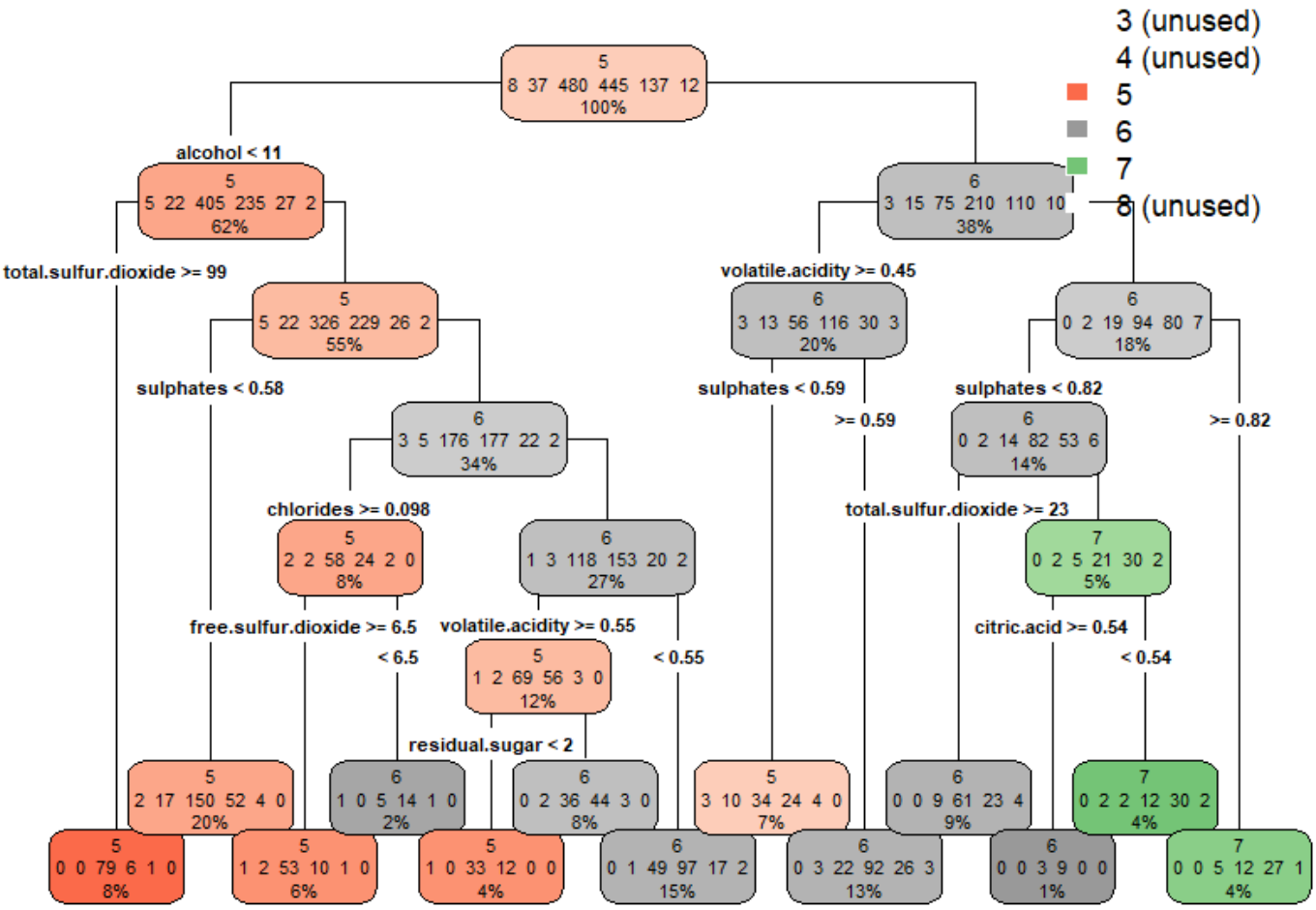
```
> # Splitting data into Training and test set
> s <- sample(1599, 1119) #devided dataset into 70/30
> wine_train <- wine[s, ]
> wine_test <- wine[-s, ]
> dim(wine_train)
[1] 1119   12
> dim(wine_test)
[1] 480  12
```

We now have two randomized samples of the data. Let's create the decision tree model using rpart(). Our decision tree model`s name is tm.

```
#working with 'rpart" library and building decision tree model
tm = rpart(quality~., data = wine_train, method = "class") # build a classification tree model
tm # retrieve the node detail of the classification tree
printcp(tm) #examine the complexity parameter
plotcp(tm)
summary(tm) #examine the built model
rpart.plot(tm, uniform=TRUE, tweak = 1.5, extra = 101, type = 4) # visualzing the tree
```

```
> printcp(tm) #examine the complexity parameter

Classification tree:
rpart(formula = quality ~ ., data = wine_train, method = "class")

Variables actually used in tree construction:
[1] alcohol              chlorides            citric.acid          free.sulfur.dioxide
[5] residual.sugar       sulphates            total.sulfur.dioxide volatile.acidity

Root node error: 639/1119 = 0.57105

n= 1119

        CP nsplit rel error  xerror     xstd
1 0.211268      0   1.00000 1.00000 0.025909
2 0.018258      1   0.78873 0.80438 0.026088
3 0.014085      5   0.71362 0.76839 0.025978
4 0.013041      6   0.69953 0.77152 0.025989
5 0.012520     11   0.63224 0.76213 0.025954
6 0.010000     12   0.61972 0.72770 0.025799
```

Here is our decision tree:



In this graph, yes is always to the left and no is always to the right. Each branch is a decision for splitting the data into a new classification. The decision tree split the data into only 3 of the 6 available classifications: 5, 6 and 7. The furthest branches show that this prediction made quite a lot of errors. Let's go on to test its prediction on the unseen data.

```
#prediction performance
predictions <- predict(tm, wine_test, type = "class") #generate a predicted label of testing the
table(wine_test$quality, predictions)
```

After this we should look at the confusion matrix where we will see the accuracy rate:

```
> #working with library caret and generating confusion matrix
> confusionMatrix(table(predictions, wine_test$quality)) # generate a confusion m
ification table
Confusion Matrix and Statistics


predictions   3   4   5   6   7   8
          3   0   0   0   0   0   0
          4   0   0   0   0   0   0
          5   2  11 129  59   2   0
          6   0   5  67 115  41   2
          7   0   0   5  19  19   4
          8   0   0   0   0   0   0

Overall Statistics

               Accuracy : 0.5479
                 95% CI : (0.5022, 0.5931)
    No Information Rate : 0.4188
    P-Value [Acc > NIR] : 8.357e-09

                  Kappa : 0.268
 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   0.6418   0.5959  0.30645   0.0000
Specificity          1.000000  1.00000   0.7348   0.5993  0.93301   1.0000
Pos Pred Value            NaN      NaN   0.6355   0.5000  0.40426      NaN
Neg Pred Value       0.995833  0.96667   0.7401   0.6880  0.90069   0.9875
Prevalence           0.004167  0.03333   0.4188   0.4021  0.12917   0.0125
Detection Rate       0.000000  0.00000   0.2687   0.2396  0.03958   0.0000
Detection Prevalence 0.000000  0.00000   0.4229   0.4792  0.09792   0.0000
Balanced Accuracy    0.500000  0.50000   0.6883   0.5976  0.61973   0.5000
```

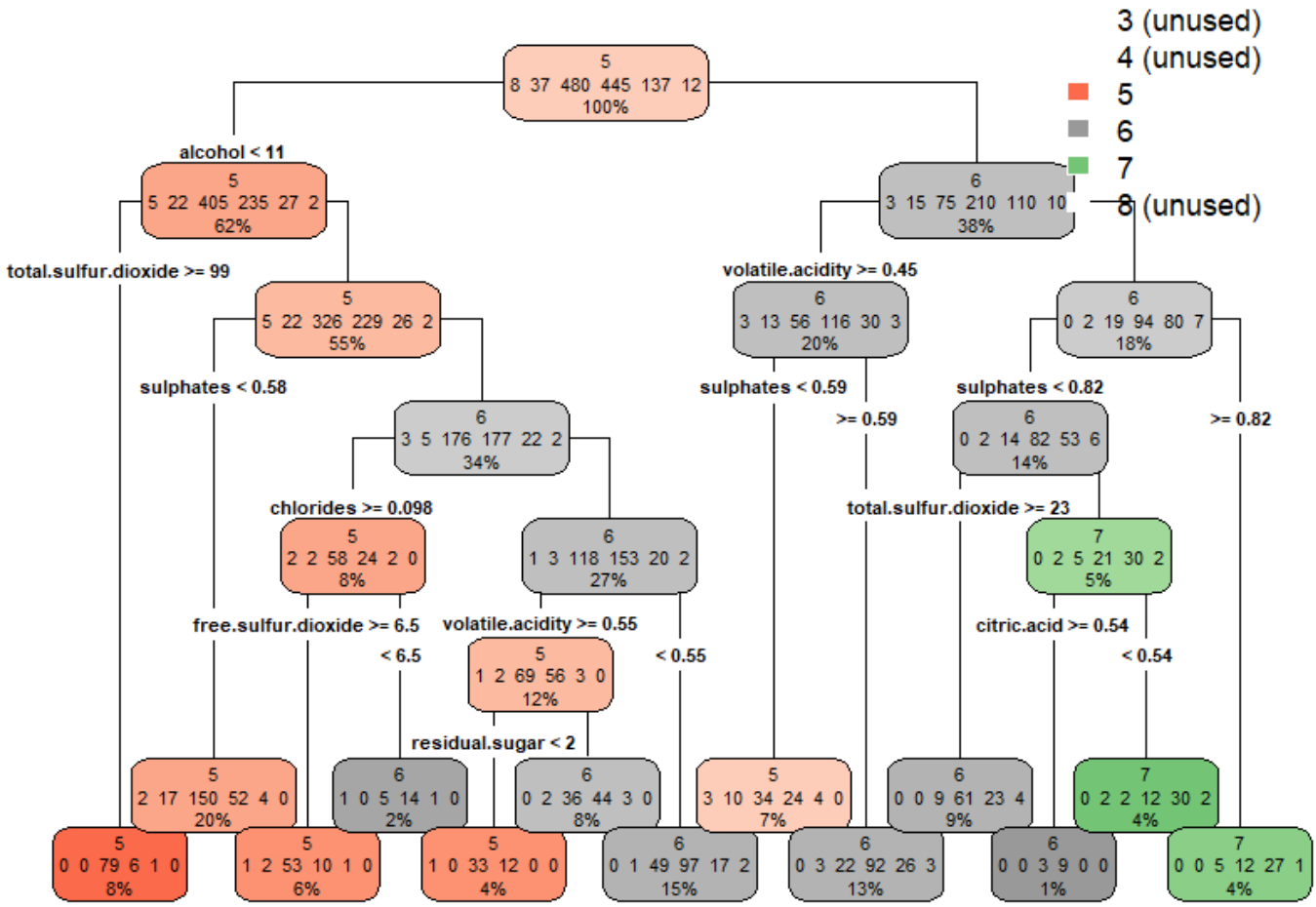
We can see that the model`s accuracy is only 54.79% which is pretty low.

After this step we are going to try to improve the model with pruning the tree by applying the function prun(). Our prune tree name will be prune.tree.tm

```
#Pruning
min(tm$cptable[,"xerror"])
which.min(tm$cptable[,"xerror"])
tm.cp = tm$cptable[7,"CP"]
tm.cp
prune.tree.tm = prune(tm, cp= tm.cp)
rpart.plot(prune.tree.tm, uniform=TRUE, tweak = 1.5, extra = 101, type = 4)
```

```
> #Pruning
> min(tm$cptable[,"xerror"]) #Find the minimum cross-validation error of the classification tree
[1] 0.7276995
> which.min(tm$cptable[,"xerror"]) #Locate the record with the minimum cross-validation errors
6
6
> tm.cp = tm$cptable[6,"CP"] #Get the cost complexity parameter of the record with the minimum c
dation errors:
> tm.cp
[1] 0.01
> prune.tree.tm = prune(tm, cp= tm.cp) #Prune the tree by setting the cp parameter to the CP val
 record with minimum cross-validation errors
> rpart.plot(prune.tree.tm, uniform=TRUE, tweak = 1.5, extra = 101, type = 4) #Visualize the cla
on tree
```

And generate the tree after pruning:



We are going to produce the prediction model for the prune tree and the confusion matrix to identify the accuracy of the tree.

```
#predictions for pruning tree
predictions.prune <- predict(prune.tree.tm, wine_test, type = "class")
table(wine_test$quality, predictions.prune)
confusionMatrix(table(predictions.prune, wine_test$quality))
```

```
> confusionMatrix(table(predictions.prune, wine_test$quality)) #generate a confusion matri
classification table
Confusion Matrix and Statistics


predictions.prune   3   4   5   6   7   8
                3   0   0   0   0   0   0
                4   0   0   0   0   0   0
                5   2  11 129  59   2   0
                6   0   5  67 115  41   2
                7   0   0   5  19  19   4
                8   0   0   0   0   0   0

Overall Statistics

               Accuracy : 0.5479
                 95% CI : (0.5022, 0.5931)
    No Information Rate : 0.4188
    P-Value [Acc > NIR] : 8.357e-09

                  Kappa : 0.268
 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   0.6418   0.5959  0.30645   0.0000
Specificity          1.000000  1.00000   0.7348   0.5993  0.93301   1.0000
Pos Pred Value            NaN      NaN   0.6355   0.5000  0.40426      NaN
Neg Pred Value       0.995833  0.96667   0.7401   0.6880  0.90069   0.9875
Prevalence           0.004167  0.03333   0.4188   0.4021  0.12917   0.0125
Detection Rate       0.000000  0.00000   0.2687   0.2396  0.03958   0.0000
Detection Prevalence 0.000000  0.00000   0.4229   0.4792  0.09792   0.0000
Balanced Accuracy    0.500000  0.50000   0.6883   0.5976  0.61973   0.5000
```

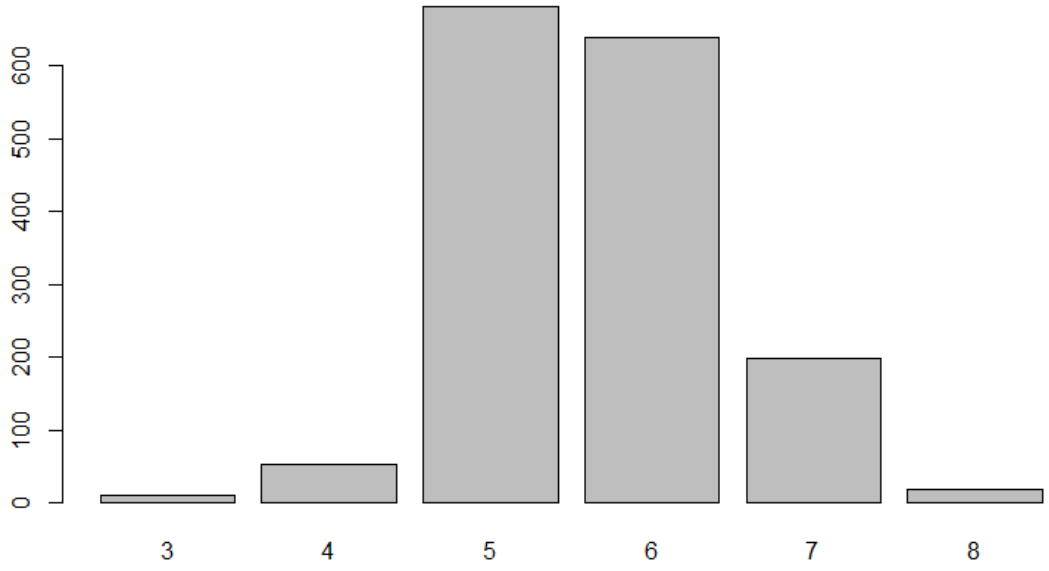
As shown above, the predictions were only 54.79% accurate, which isn't very good. If we compare pruned tree with our original tree, we can see it is the same! Tree is the same and accuracy level is the same as well. The reason for them to be the same could be that for the decision tree we used levels 5, 6 and 7 without 3, 4 and 8. So our decision tree had already reduced parts that did not provide power to classify instances.

Now we will try to improve the model by applying the random forest algorithm.

To help improve the power of the model we reduce the levels of classification from 6 to 3. Wines ranked at 7 and 8 become "good", 5 and 6 became "normal", and then 3 and 4 become "bad".

Let's look again at the distribution of wine rankings, this time with a bar plot. And then we can reload the data in its original form and will call it wine2 and build random forest model.

```
#building random forest model
barplot(table(wine$quality))
```

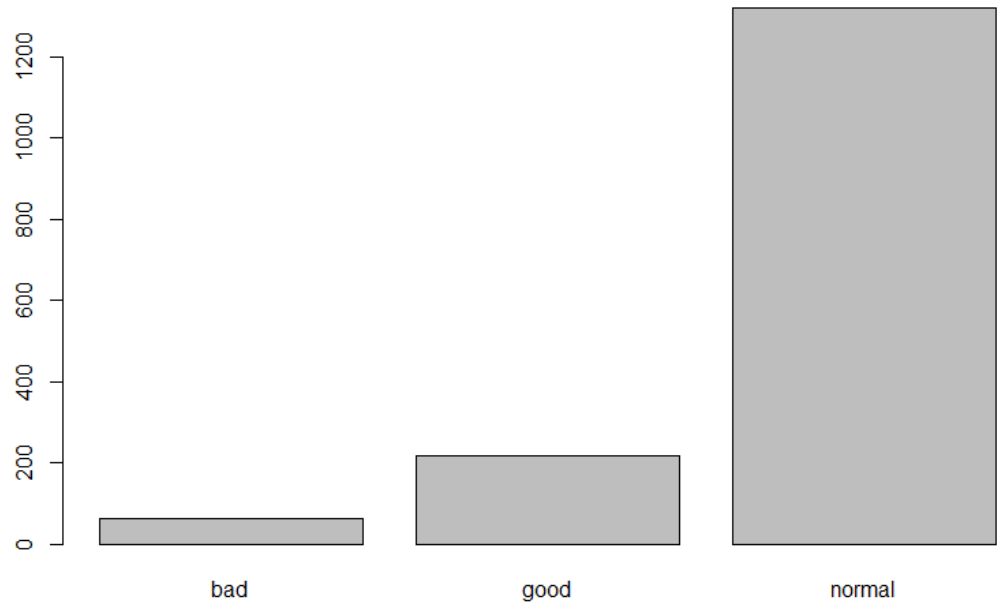


We can look at the distribution:

```
> table(wine2$taste)

   bad   good normal
    63    217   1319
```

We'd like to classify the wines ranked as 5 and 6 as "normal", the lower ranked wines as "bad", and the wines ranked above as "good".

```
#grouping wine ranks into 3 groups
wine2 <- read.csv(url("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/w
                  header = TRUE, sep = ";")
wine2$taste <- ifelse(wine2$quality < 5, "bad", "good")
wine2$taste[wine2$quality == 5] <- "normal"
wine2$taste[wine2$quality == 6] <- "normal"
wine2$taste <- as.factor(wine2$taste)
str(wine2$taste)
barplot(table(wine2$taste))
```



As seen above, there are a lot more normal wines in the dataset then there are bad or good. We can now proceed to splitting our data into training and testing sets like we did for decision tree model. We'll use 70% for training again for the random forest approach and 30% for the testing.

```
79  #random forest train and test sets
80  samp <- sample(1599, 1119)
81  wine_train2 <- wine2[samp, ]
82  wine_test2 <- wine2[-samp, ]
83  dim(wine_train2)
84  dim(wine_test2)
85
```

The final step is to build the random forest model and figure out the accuracy of that model:

```
model <- randomForest(taste ~ . - quality, data = wine_train2)
model
prediction <- predict(model, newdata = wine_test2)
table(prediction, wine_test2$taste)
```

```
> #random forest model and table with predictions
> model <- randomForest(taste ~ . - quality, data = wine_train2)
> model

Call:
 randomForest(formula = taste ~ . - quality, data = wine_train2)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 14.92%
Confusion matrix:
       bad good normal class.error
bad      1    1     43   0.9777778
good     0   74     83   0.5286624
normal   2   38    877   0.0436205
```

We can test the accuracy as follows:

```
> prediction <- predict(model, newdata = wine_test2)
> table(prediction, wine_test2$taste)

prediction bad good normal
    bad      0    0      0
    good     0   34      8
    normal  18   26    394
> (0+34+394)/nrow(wine_test2)
[1] 0.8916667
```

As seen above, our model was approx. 89% accurate - a major improvement from our decision tree. If we compare with two previous models (decision tree and the pruning tree) the random forests model has the highest accuracy! When we build a random forest, we don't have a visual tree, but we have accuracy rate that we can look at. I think 89% accuracy is amazing result of the model!

```
> tm$variable.importance
            alcohol           sulphates total.sulfur.dioxide             density
          64.263153           35.609871           32.173855           31.229609
    volatile.acidity           chlorides         citric.acid free.sulfur.dioxide
          28.958596           24.287246           16.992160           16.204977
       fixed.acidity                  pH      residual.sugar
          15.305493            9.179071            6.021613
```