Decision Trees and Random Forests

Taylor Shrode

9/20/2020

Introduction

Classification is used to identify the category of a given observation from a category of new observations (Yu-Wei, 2015). This classification is based on a classification model that is built from a trained dataset, where the categories are known. The classification method we will be using is a tree based classification method using decision trees and random forests (Yu-Wei, 2015). A decision tree is used to visualize and represent decisions and decision making (Gupta, 2017). A decision tree is drawn upside down with its root at the top. The tree is composed of nodes (conditions), edges (tree branches), and leaves, (decisions) (Gupta, 2017). A random forest is an ensemble tree-based learning algorithm, which is a set of decision trees from a randomly selected subset of the training set (Chakure, 2020).

For this assignment, we will be using a dataset that contains red vinho verde wine samples from the north of Portugal (Cortez et al, n.d.). The variables included in this dataset are:

1. Fixed acidity
2. Volatile acidity
3. Citric acid
4. Residual sugar
5. Chlorides
6. Free sulfur dioxide
7. Total sulfur dioxide
8. Density
9. pH
10. Sulphates
11. Alcohol
12. Quality (score between 0 and 10)

where *quality* is the outcome (response variable). The objective is to predict wine quality rankings from the chemical properties using tree-based classification methods. This provides guidance to vineyards regarding the quality of wine and the expected price without heavily relying on wine tasters.

Libraries

Before we begin building our classification trees, we need to load the necessary libraries into R.

```
library(DataExplorer)
library(caret)
library(rpart.plot)
library(rpart)
library(rattle)
library("randomForest")
library(ggplot2)
```

The **DataExplorer** library allows us to perform data exploration analysis. The **caret** package provides feature selection tools so that we can rank features by their importance and to find attributes that are highly correlated. The **rpart** package allows us to create our decision tree and the **rpart.plot** package allows us to plot our decision tree (Yu-Wei, 2015). The **rattle** package allows us to create a "fancy" **rpart** plot (Dinov, 2020). The **randomForest** package is loaded to classify our wine data using Random Forest classification. Finally, we load the **ggplot2** package so we can plot out variable importance output for each decision tree.

Load Dataset

To load our data into R, we can load the data directly from the URL using the **read.csv()** function (Read csv from the web, n.d.). We add the argument **sep = ";"** to separate the columns in the dataset properly.

```
wine_df <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-red.csv", sep = ';')
head(wine_df)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.4             0.70        0.00            1.9     0.076
## 2           7.8             0.88        0.00            2.6     0.098
## 3           7.8             0.76        0.04            2.3     0.092
## 4          11.2             0.28        0.56            1.9     0.075
## 5           7.4             0.70        0.00            1.9     0.076
## 6           7.4             0.66        0.00            1.8     0.075
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  11                   34  0.9978 3.51      0.56     9.4
## 2                  25                   67  0.9968 3.20      0.68     9.8
## 3                  15                   54  0.9970 3.26      0.65     9.8
## 4                  17                   60  0.9980 3.16      0.58     9.8
## 5                  11                   34  0.9978 3.51      0.56     9.4
## 6                  13                   40  0.9978 3.51      0.56     9.4
##   quality
## 1       5
## 2       5
## 3       5
## 4       6
## 5       5
## 6       5
```

Our dataset is stored in the **wine_df** dataframe.

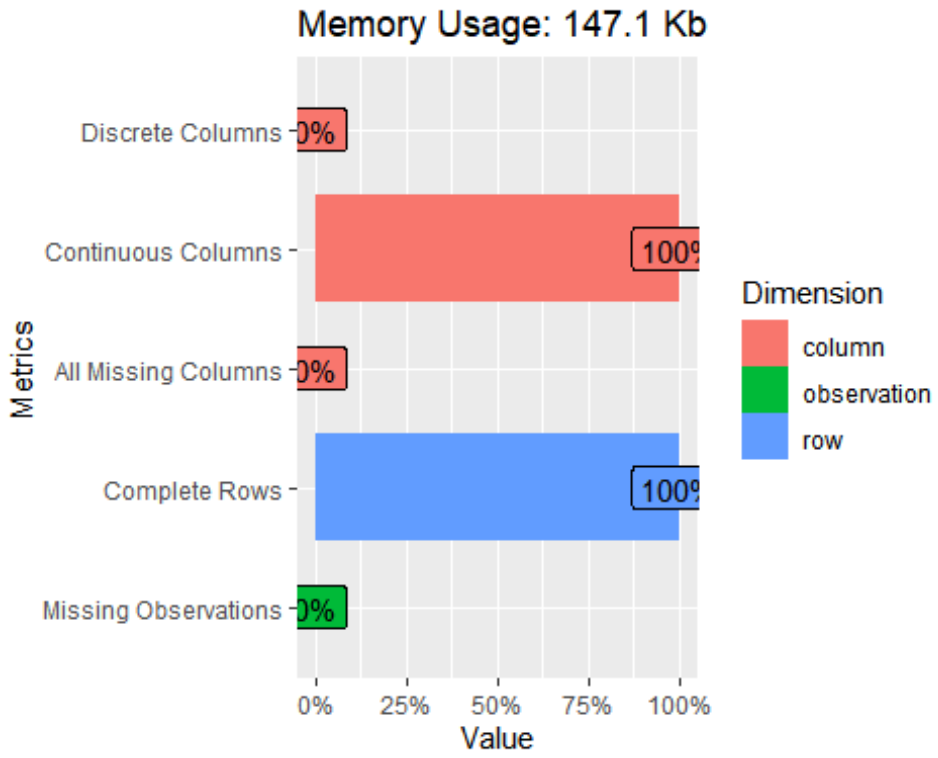
Data Exploration

To begin exploring our data, we will use the **introduce()** function and plot the results.

```
introduce(wine_df)
```

```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 1599      12                0                  12                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                    0          1599              19188       150608
```

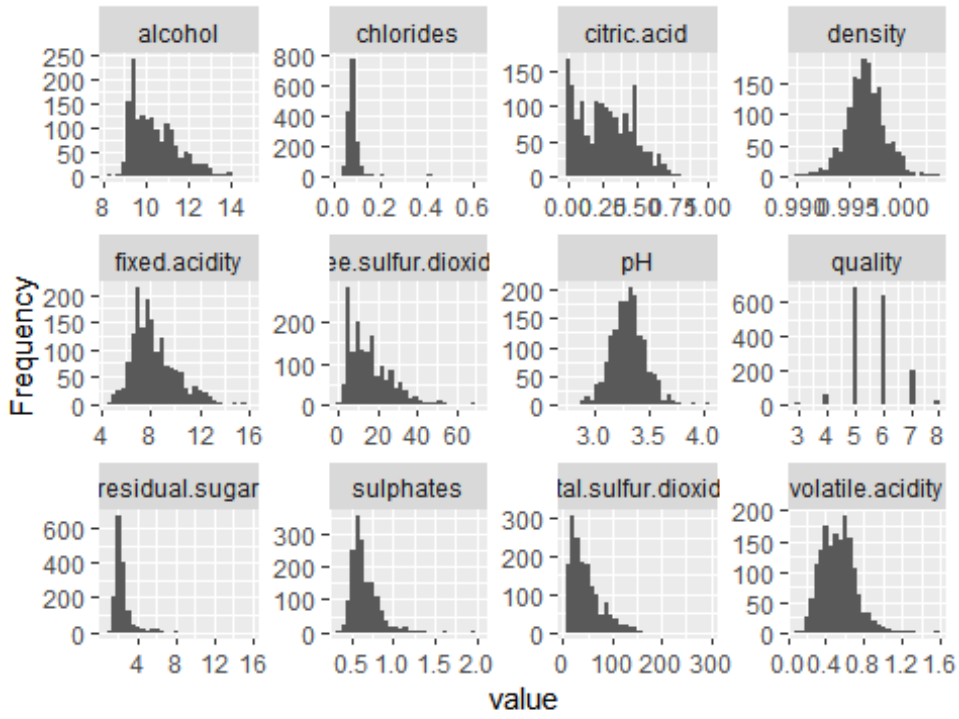
```
plot_intro(wine_df)
```




```
#plot_bar(wine_df) #discrete features
```

First, we can see that our wine dataset does not contain any missing values. Further, we can see that all of the columns in the dataset have been labeled as "continuous". Now, since all of the columns contain continuous values, we will use a histogram to plot our features.

```
plot_histogram(wine_df) #continuous features
```



From our histogram above, we see that that the wine quality ranks from 3 to 8 and most of the quality rankings are either 5 or 6.

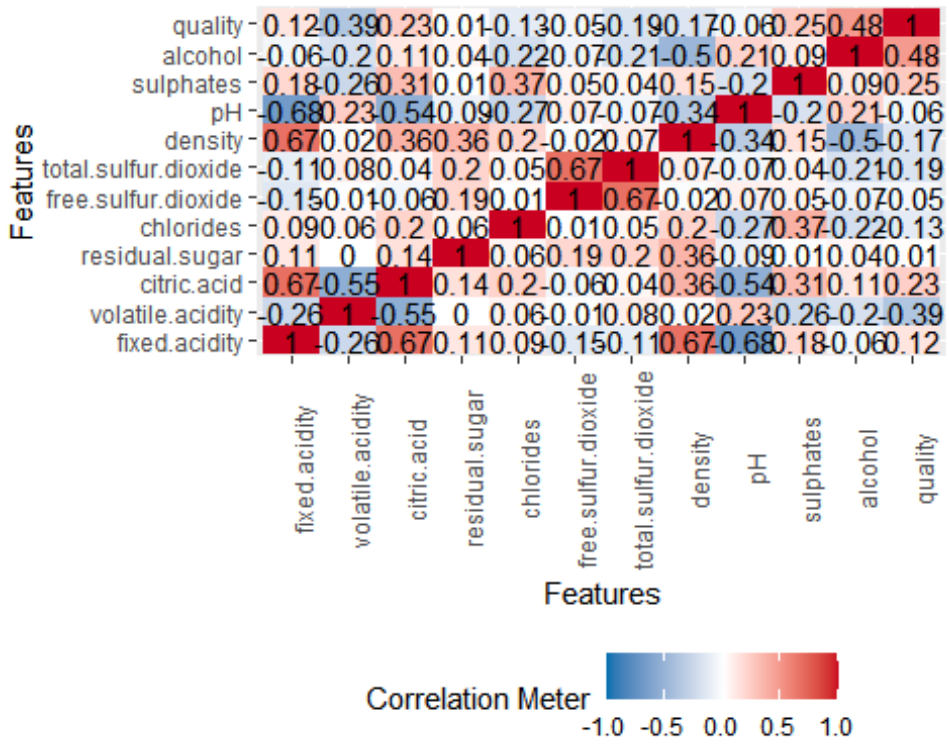
```
table(wine_df$quality)
```

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

```
## Percentage of Quality Ranks of 5 and 6: 82.48906
```

Notice, ~82% of our quality rankings are either a 5 or 6. Thus, we will group the ranks in the following section. To continue our data exploration, we will view a corrplot of our data.

```
plot_correlation(wine_df)
```



At first glance, we can see a few variables highly correlate with other variables, like **fixed.acidity**, **citric.acid**, **free.sulfur.dioxide**, and **pH**. We can use the **FindCorrelation()** function to determine the attributes that are highly correlated.

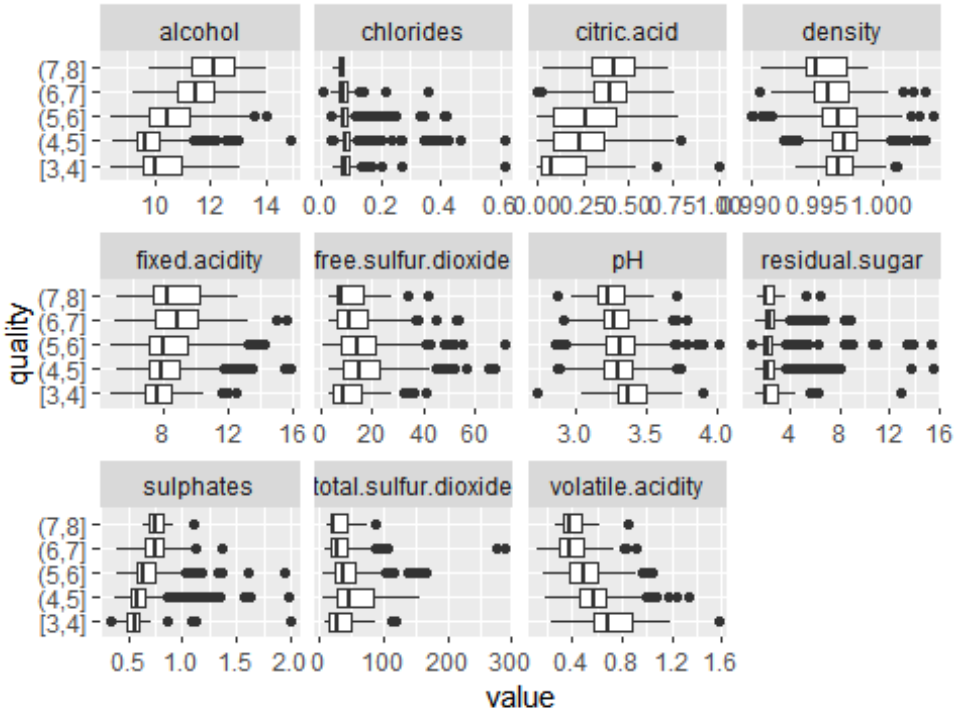
```
cor_data <- subset(wine_df[,1:11])
cor <- cor(cor_data)
findCorrelation(cor, cutoff=0.5)

## [1] 1 3 7
```

The output above suggests that the following columns are highly correlated:

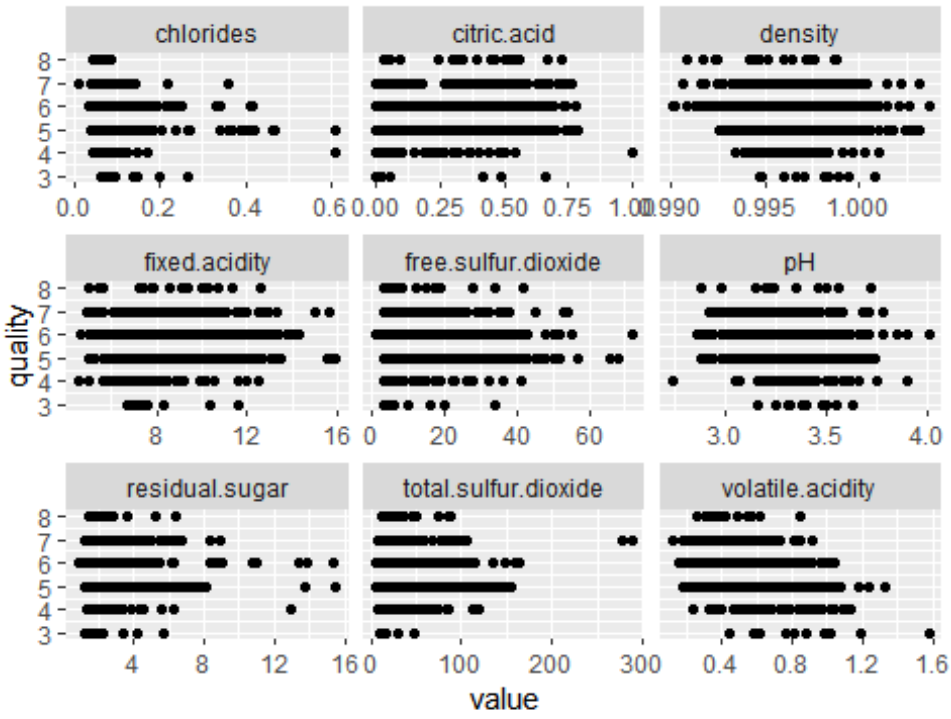
```
## Column 1: fixed.acidity

## Column 3: citric.acid

## Column 7: total.sulfur.dioxide
```

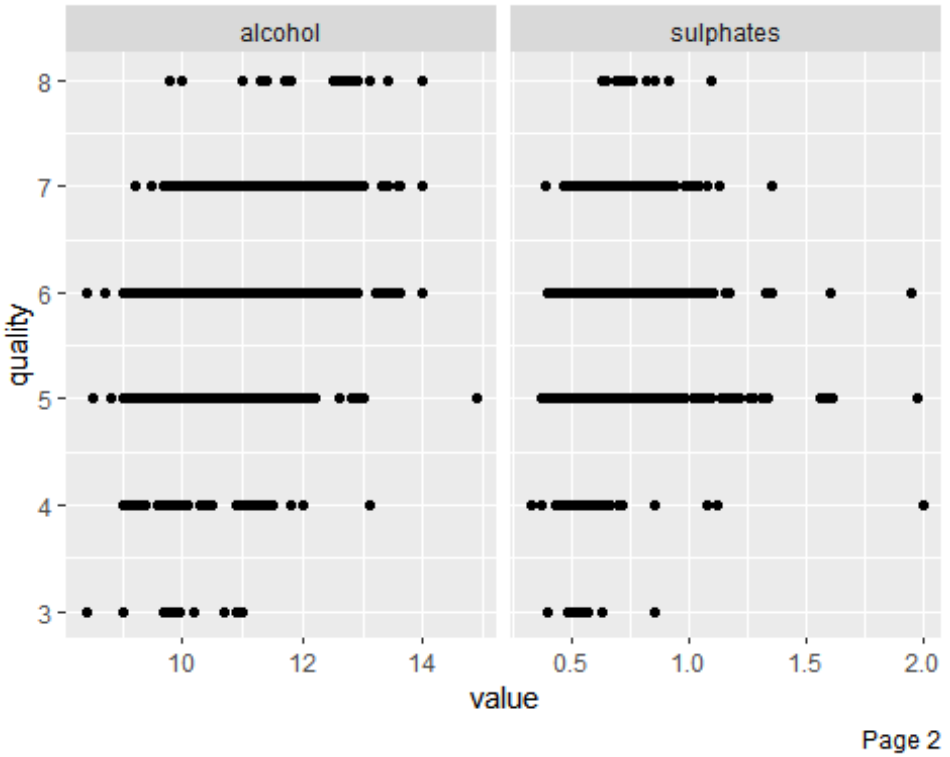
Now, we will view a boxplot and scatterplot of our data by the quality rankings of the wine.

```
plot_boxplot(wine_df, by = "quality")
```




```
plot_scatterplot(wine_df, by = "quality")
```





We can see from our boxplots, that many of our features are distributed evenly across rankings.

Convert Response Variable to Factor and Group Wine Quality Levels

As noted in the section above, most of our quality rankings are 5 or 6. To avoid any possible issues, we will group our rankings into two groups: above average and below average. The average of our wine quality rankings is 5.5. Thus, a ranking of 3, 4, or 5 will be converted to a value of 0, which indicates a below average wine quality. A ranking of 6, 7, or 8 will be converted to a value of 1, which indicates an above average wine quality.

Recall, the previous distribution of our ranking values is:

```
table(wine_df$quality)

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

To convert our quality values, we can utilize the **ifelse()** function. We will also need to convert our **quality** variable to a factor because we are using a classification decision tree in a later section.

```
wine_df$quality <- as.factor(with(wine_df, ifelse(quality >=6, 1, 0))) #group quality variable; convert response variable to factor (this is a classification tree)
str(wine_df$quality)

##  Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 2 2 1 ...
```

Now, the distribution of our ranking values is:

```
## Total Below Average Wine Quality Rankings (Value = 0): 744

## Total Above Average Wine Quality Rankings (Value = 1): 855
```

Notice, our ranking values are more evenly distributed.

Normalize and Combine Data

To avoid *dominating* features, which can result in skewed classifications, we will normalize our features. Excluding our response variable.

```
norm <- function(x) {return ((x - min(x)) / (max(x) - min(x)))}
wine_norm <- as.data.frame(lapply(wine_df[,c(1:11)],norm)) #want to normalize numerical columns
```

Now, to combine our normalized features with our converted response variable, we will use the **cbind()** function.

```
final_df <- cbind(wine_norm, wine_df$quality)
head(final_df)

##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1     0.2477876        0.3972603        0.00     0.06849315 0.1068447
```

```
## 2     0.2831858        0.5205479        0.00     0.11643836 0.1435726
## 3     0.2831858        0.4383562        0.04     0.09589041 0.1335559
## 4     0.5840708        0.1095890        0.56     0.06849315 0.1051753
## 5     0.2477876        0.3972603        0.00     0.06849315 0.1068447
## 6     0.2477876        0.3698630        0.00     0.06164384 0.1051753
##   free.sulfur.dioxide total.sulfur.dioxide   density        pH sulphates
## 1           0.1408451           0.09893993 0.5675477 0.6062992 0.1377246
## 2           0.3380282           0.21554770 0.4941263 0.3622047 0.2095808
## 3           0.1971831           0.16961131 0.5088106 0.4094488 0.1916168
## 4           0.2253521           0.19081272 0.5822320 0.3307087 0.1497006
## 5           0.1408451           0.09893993 0.5675477 0.6062992 0.1377246
## 6           0.1690141           0.12014134 0.5675477 0.6062992 0.1377246
##     alcohol wine_df$quality
## 1 0.1538462               0
## 2 0.2153846               0
## 3 0.2153846               0
## 4 0.2153846               1
## 5 0.1538462               0
## 6 0.1538462               0
```

Notice, the values for all our features are between the values 0 and 1. Also, notice that our response variable column needs to be renamed. We can do this by using the **names()** function (Rename data frame columns in r, n.d.).

```
names(final_df)[names(final_df) == "wine_df$quality"] <- "quality"
colnames(final_df)
```

```
##  [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"
##  [4] "residual.sugar"       "chlorides"            "free.sulfur.dioxide"
##  [7] "total.sulfur.dioxide" "density"              "pH"
## [10] "sulphates"            "alcohol"              "quality"
```

Now, we can proceed and prepare our training and testing datasets.

Create Training and Testing Datasets

Classification models require training datasets to train the classification model and a testing dataset that is used to validate the prediction performance (Yu-Wei, 2015). We will split 70% of the data into the training dataset and 30% of the data into the testing dataset. To do this, we will use the **createDataPartition()** function, which is located in the **caret** library. This allows us to create balanced splits of our data. We will also need to set the seed for the random generator so our results can be reproduced. Lastly, we will also create the labels that will be used to train and test the model.

```
set.seed(789)
index <- createDataPartition(final_df$quality, p =0.7, list = FALSE)
wineTrain <- final_df[index,] #index for training set
train_labels <- final_df[12][index,]
```

```
wineTest <- final_df[-index,] #not index for test set
test_labels <- final_df[-index,12]
```

To confirm the split for the training and testing datasets, we will use the **dim()** function.

```
dim(wineTrain)

## [1] 1120   12

dim(wineTest)

## [1] 479  12
```

Now, we can begin building our classification model with recursive partitioning trees (Yu-Wei, 2015).

Train, Predict, Evaluate, and Plot Models

Classification using recursive partitioning trees is a fundamental tool in data mining (Kabacoff, 2017). Tree based models help us explore the structure of a dataset while also developing an easy method to visualize decision rules for predicting a categorical variable.

Decision Tree

First, we will grow a decision tree using the **rpart** package. We will use the **rpart()** function with the argument **method = "class"** because we are predicting a categorical outcome (Kabacoff, 2017). Also, the argument **cp = 0** is added to control the complexity parameter. The complexity parameter is the minimum improvement in the model that is needed at each node (Decision trees in r, n.d.). In other words, **cp** is a stopping parameter because it helps speed up the search for splits. This allows the model to identify splits that don't meet this criteria and it will prune them before doing unnecessary work (Decision trees in r, n.d.). We will be pruning the tree in the next session, so we will set **cp** to 0 now and not put a stop on our tree.

```
wine_dtree <- rpart(quality ~., wineTrain, method = "class", cp = 0)
```

Now that we have our decision tree, we can display our **cp** table with the command **printcp()**.

```
printcp(wine_dtree)

##
## Classification tree:
## rpart(formula = quality ~ ., data = wineTrain, method = "class",
##     cp = 0)
##
## Variables actually used in tree construction:
##  [1] alcohol              chlorides            citric.acid
##  [4] density              fixed.acidity        free.sulfur.dioxide
##  [7] pH                   residual.sugar       sulphates
## [10] total.sulfur.dioxide volatile.acidity
```

```
##
## Root node error: 521/1120 = 0.46518
##
## n= 1120
##
##           CP nsplit rel error  xerror     xstd
## 1  0.3435701      0   1.00000 1.00000 0.032039
## 2  0.0220729      1   0.65643 0.69098 0.029999
## 3  0.0201536      5   0.55086 0.65259 0.029535
## 4  0.0115163      7   0.51056 0.63724 0.029335
## 5  0.0105566      8   0.49904 0.60653 0.028908
## 6  0.0076775     11   0.46641 0.59309 0.028711
## 7  0.0063980     14   0.44338 0.58925 0.028653
## 8  0.0057582     17   0.42418 0.57006 0.028355
## 9  0.0053743     21   0.40115 0.57198 0.028386
## 10 0.0052783     26   0.37428 0.56622 0.028294
## 11 0.0047985     31   0.34357 0.56622 0.028294
## 12 0.0038388     38   0.30902 0.56238 0.028232
## 13 0.0028791     41   0.29750 0.56238 0.028232
## 14 0.0019194     43   0.29175 0.56046 0.028201
## 15 0.0000000     45   0.28791 0.57198 0.028386
```

To print our results, we can use the **print()** function or we can use the **summary()** function to view the details of our tree, like surrogate splits. Only a partial portion of the **print()** output is shown below due to length of output.

```
#summary(wine_dtree) #detailed results
print(wine_dtree) #print results
```

```
## n= 1120
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 1120 521 1 (0.46517857 0.53482143)
##      2) alcohol< 0.3 619 220 0 (0.64458805 0.35541195)
##        4) sulphates< 0.1526946 285  63 0 (0.77894737 0.22105263)
##          8) sulphates< 0.1227545 158  24 0 (0.84810127 0.15189873)
##           16) volatile.acidity>=0.1952055 136  16 0 (0.88235294
0.11764706)
##             32) citric.acid>=0.195 66   3 0 (0.95454545 0.04545455) *
##             33) citric.acid< 0.195 70  13 0 (0.81428571 0.18571429)
##               66) citric.acid< 0.115 54   4 0 (0.92592593 0.07407407) *
##               67) citric.acid>=0.115 16   7 1 (0.43750000 0.56250000) *
##           17) volatile.acidity< 0.1952055 22   8 0 (0.63636364 0.36363636)
##             34) chlorides>=0.1118531 15   2 0 (0.86666667 0.13333333) *
##             35) chlorides< 0.1118531 7   1 1 (0.14285714 0.85714286) *
##          9) sulphates>=0.1227545 127  39 0 (0.69291339 0.30708661)
##           18) density>=0.5062408 63  10 0 (0.84126984 0.15873016)
##             36) alcohol< 0.2384615 55   5 0 (0.90909091 0.09090909) *
```

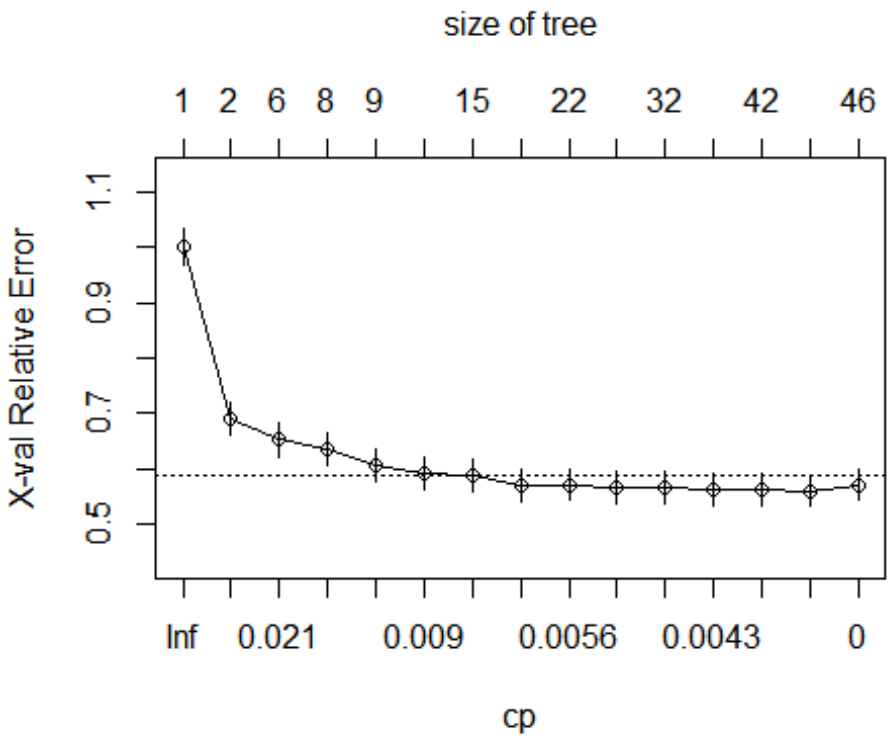
```
##             37) alcohol>=0.2384615 8   3 1 (0.37500000 0.62500000) *
##           19) density< 0.5062408 64  29 0 (0.54687500 0.45312500)
##             38) total.sulfur.dioxide>=0.319788 9   0 0 (1.00000000
0.00000000) *
##             39) total.sulfur.dioxide< 0.319788 55  26 1 (0.47272727
0.52727273)
##               78) total.sulfur.dioxide< 0.1943463 42  17 0 (0.59523810
0.40476190)
##                156) total.sulfur.dioxide>=0.114841 17   2 0 (0.88235294
0.11764706) *
##                157) total.sulfur.dioxide< 0.114841 25  10 1 (0.40000000
0.60000000)
##                  314) free.sulfur.dioxide< 0.1619718 18   8 0 (0.55555556
0.44444444) *
##                  315) free.sulfur.dioxide>=0.1619718 7   0 1 (0.00000000
1.00000000) *
##               79) total.sulfur.dioxide>=0.1943463 13   1 1 (0.07692308
0.92307692) *
##        5) sulphates>=0.1526946 334 157 0 (0.52994012 0.47005988)
##         10) total.sulfur.dioxide>=0.2632509 61   9 0 (0.85245902
0.14754098) *
##         11) total.sulfur.dioxide< 0.2632509 273 125 1 (0.45787546
0.54212454)
##           22) volatile.acidity>=0.2089041 186  84 0 (0.54838710
0.45161290)
##             44) free.sulfur.dioxide>=0.1056338 126  47 0 (0.62698413
0.37301587)
##               88) chlorides>=0.1410684 28   3 0 (0.89285714 0.10714286) *
##               89) chlorides< 0.1410684 98  44 0 (0.55102041 0.44897959)
##                178) density< 0.5517621 61  20 0 (0.67213115 0.32786885)
##                  356) volatile.acidity>=0.3630137 20   2 0 (0.90000000
0.10000000) *
##                  357) volatile.acidity< 0.3630137 41  18 0 (0.56097561
0.43902439)
##                    714) citric.acid>=0.09 23   6 0 (0.73913043 0.26086957)
*
##                    715) citric.acid< 0.09 18   6 1 (0.33333333 0.66666667)
*
##                179) density>=0.5517621 37  13 1 (0.35135135 0.64864865)
##                  358) sulphates< 0.1826347 14   4 0 (0.71428571
0.28571429) *
##                  359) sulphates>=0.1826347 23   3 1 (0.13043478
0.86956522) *
##             45) free.sulfur.dioxide< 0.1056338 60  23 1 (0.38333333
0.61666667)
##               90) fixed.acidity< 0.199115 8   2 0 (0.75000000 0.25000000)
*
##               91) fixed.acidity>=0.199115 52  17 1 (0.32692308 0.67307692)
##                182) pH< 0.4212598 31  15 1 (0.48387097 0.51612903)
##                  364) sulphates>=0.2275449 10   2 0 (0.80000000
```

```
0.20000000) *
##                  365) sulphates< 0.2275449 21   7 1 (0.33333333
0.66666667)
##                    730) density>=0.5859031 10   4 0 (0.60000000
0.40000000) *
##                    731) density< 0.5859031 11   1 1 (0.09090909
0.90909091) *
##                183) pH>=0.4212598 21   2 1 (0.09523810 0.90476190) *
##           23) volatile.acidity< 0.2089041 87  23 1 (0.26436782 0.73563218)
##             46) chlorides>=0.124374 28  13 1 (0.46428571 0.53571429)
##               92) free.sulfur.dioxide< 0.2183099 21   8 0 (0.61904762
0.38095238)
##                184) alcohol< 0.2076923 12   2 0 (0.83333333 0.16666667) *
##                185) alcohol>=0.2076923 9   3 1 (0.33333333 0.66666667) *
##               93) free.sulfur.dioxide>=0.2183099 7   0 1 (0.00000000
1.00000000) *
##             47) chlorides< 0.124374 59  10 1 (0.16949153 0.83050847)
##               94) citric.acid< 0.405 31   9 1 (0.29032258 0.70967742)
##                188) alcohol< 0.1538462 7   2 0 (0.71428571 0.28571429) *
##                189) alcohol>=0.1538462 24   4 1 (0.16666667 0.83333333) *
##               95) citric.acid>=0.405 28   1 1 (0.03571429 0.96428571) *
##      3) alcohol>=0.3 501 122 1 (0.24351297 0.75648703)
##        6) sulphates< 0.1526946 131  59 1 (0.45038168 0.54961832)
##         12) volatile.acidity>=0.2705479 73  26 0 (0.64383562 0.35616438)
##           24) free.sulfur.dioxide< 0.07746479 26   3 0 (0.88461538
0.11538462) *
##           25) free.sulfur.dioxide>=0.07746479 47  23 0 (0.51063830
0.48936170)
##             50) citric.acid>=0.095 13   2 0 (0.84615385 0.15384615) *
##             51) citric.acid< 0.095 34  13 1 (0.38235294 0.61764706)
##              102) pH>=0.5 26  13 0 (0.50000000 0.50000000)
##                204) fixed.acidity< 0.2123894 19   7 0 (0.63157895
0.36842105) *
##                205) fixed.acidity>=0.2123894 7   1 1 (0.14285714
0.85714286) *
##               103) pH< 0.5 8   0 1 (0.00000000 1.00000000) *
##         13) volatile.acidity< 0.2705479 58  12 1 (0.20689655 0.79310345) *
##        7) sulphates>=0.1526946 370  63 1 (0.17027027 0.82972973)
##         14) total.sulfur.dioxide>=0.3657244 8   1 0 (0.87500000
0.12500000) *
##         15) total.sulfur.dioxide< 0.3657244 362  56 1 (0.15469613
0.84530387)
##           30) alcohol< 0.4538462 210  47 1 (0.22380952 0.77619048)
##             60) pH>=0.5826772 24  11 0 (0.54166667 0.45833333)
##              120) residual.sugar< 0.07191781 10   0 0 (1.00000000
0.00000000) *
##              121) residual.sugar>=0.07191781 14   3 1 (0.21428571
0.78571429) *
##             61) pH< 0.5826772 186  34 1 (0.18279570 0.81720430)
##              122) citric.acid>=0.215 137  32 1 (0.23357664 0.76642336)
```

```
##                 244) alcohol< 0.3153846 11   4 0 (0.63636364 0.36363636) *
##                 245) alcohol>=0.3153846 126  25 1 (0.19841270 0.80158730)
##                   490) chlorides>=0.1427379 15   6 0 (0.60000000
0.40000000) *
##                   491) chlorides< 0.1427379 111  16 1 (0.14414414
0.85585586)
##                     982) total.sulfur.dioxide>=0.1784452 23   9 1
(0.39130435 0.60869565)
##                      1964) pH>=0.4606299 10   1 0 (0.90000000 0.10000000) *
##                      1965) pH< 0.4606299 13   0 1 (0.00000000 1.00000000) *
##                     983) total.sulfur.dioxide< 0.1784452 88   7 1
(0.07954545 0.92045455) *
##               123) citric.acid< 0.215 49   2 1 (0.04081633 0.95918367) *
##            31) alcohol>=0.4538462 152   9 1 (0.05921053 0.94078947) *
```

Now, to plot the complexity parameter, we can use the **plotcp()** function (Yu-Wei, 2015).

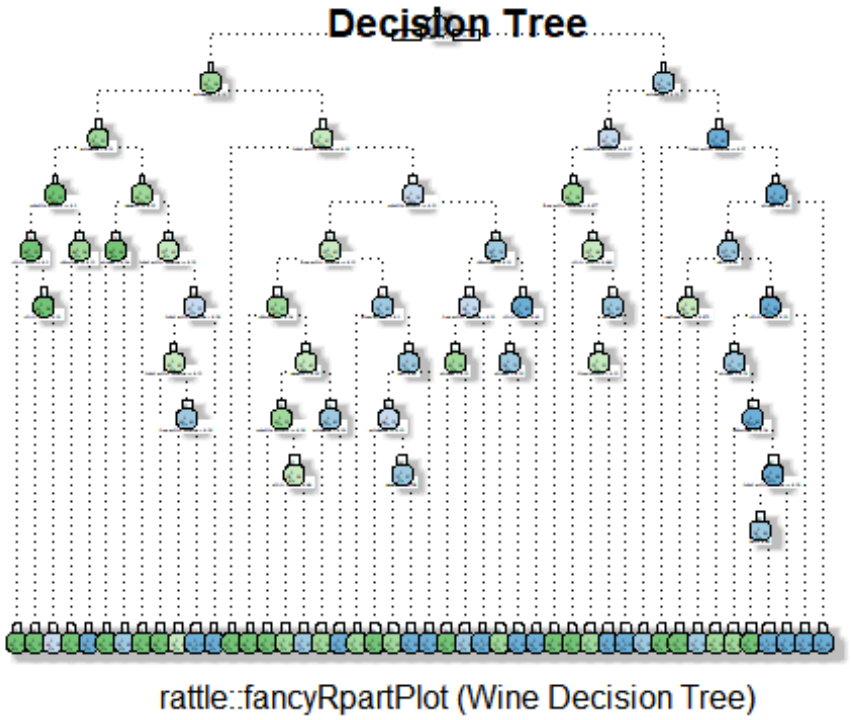
```
plotcp(wine_dtree)
```



This allows us to easily read the **cp** values. "The x-axis at the bottom illustrates the **cp** value, the y-axis illustrates the relative error, and the upper x-axis displays the size of the tree (Yu-Wei, 2015)."

There are a number of ways to plot our decision tree: the **prp()** function, the **rpart.plot()** function, the **fancyRpartPlot()** function, etc. A plot using the **fancyRpartPlot()** function can be found below (Dinov, 2020).

```
#prp(wine_dtree, box.palette = "Purples", tweak = 1.0)
#rpart.plot(wine_dtree, box.palette = "Reds", tweak = 1.5)
fancyRpartPlot(wine_dtree, cex = 0.1, main="Decision Tree", caption =
"rattle::fancyRpartPlot (Wine Decision Tree)")

## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



The plot above looks messy, and overfitted. To avoid this problem, we can prune the tree, which is shown in the next section. Before we prune the tree, we need to measure the prediction performance of our model above. To validate the prediction power of our tree, we will use the **predict()** function and use the our testing data as an argument. Then, we will check the accuracy by creating a confusion matrix.

```
wine_test_pred <- predict(wine_dtree, newdata = wineTest, type = "class")
confusionMatrix(wine_test_pred, test_labels)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction   0   1
##          0 156  54
##          1  67 202
##
##                Accuracy : 0.7474
##                  95% CI : (0.706, 0.7857)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
```

```
##
##                  Kappa : 0.4905
##
##  Mcnemar's Test P-Value : 0.2753
##
##             Sensitivity : 0.6996
##             Specificity : 0.7891
##          Pos Pred Value : 0.7429
##          Neg Pred Value : 0.7509
##              Prevalence : 0.4656
##          Detection Rate : 0.3257
##    Detection Prevalence : 0.4384
##       Balanced Accuracy : 0.7443
##
##        'Positive' Class : 0
##
```

The output above shows that our model made 358 correct predictions and made 121 false predictions, making the model ~74% accurate.

Prune Decision Tree

As shown above, our model above looks to be complex and overfitted. To avoid this, we can prune the tree which removes sections of the tree that are not powerful when making classification decisions (Yu-Wei, 2015). To prune the tree, we need to fine the minimum cross-validation error value (**xerror** column in the **printcp()** output).

```
min(wine_dtree$cptable[,'xerror']) #Find the minimum cross-validation error
of the classification tree model
```

```
## [1] 0.5604607
```

Now, we need to locate the record with the minimum cross-validation error value (Yu-Wei, 2015).

```
which.min(wine_dtree$cptable[,'xerror'])
```

```
## 14
## 14
```

Using this information, we can get our **cp** value with the minimum cross-validation error.

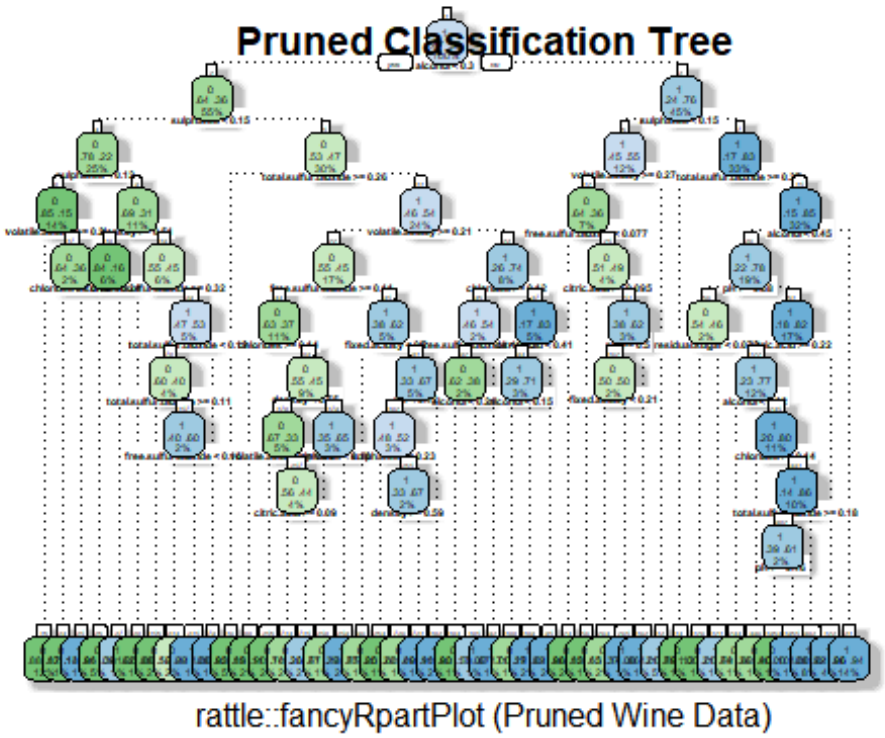
```
min_cp = wine_dtree$cptable[which.min(wine_dtree$cptable[,'xerror']),'CP']
```

To prune the tree, we will set the **cp** parameter to the **cp** value above. We use this value because we typically want to select a tree size that minimizes the cross-validated error (Kabacoff, 2017).

```
prune_wine_dtree <- prune(wine_dtree, cp = min_cp)
```

Similarly to our tree in the last section, we can plot our pruned tree with the **fancyRpartPlot()** function.

```
fancyRpartPlot(prune_wine_dtree, cex = 0.3, uniform=TRUE, main="Pruned
Classification Tree", caption = "rattle::fancyRpartPlot (Pruned Wine Data)")

## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Now, we will generate a confusion matrix for our pruned tree.

```
wine_prune_test_pred <- predict(prune_wine_dtree, newdata = wineTest, type =
"class")
confusionMatrix(wine_prune_test_pred, test_labels)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction   0   1
##          0 160  55
##          1  63 201
##
##                Accuracy : 0.7537
##                  95% CI : (0.7125, 0.7916)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5038
##
```

```
##  Mcnemar's Test P-Value : 0.5193
##
##             Sensitivity : 0.7175
##             Specificity : 0.7852
##          Pos Pred Value : 0.7442
##          Neg Pred Value : 0.7614
##              Prevalence : 0.4656
##          Detection Rate : 0.3340
##    Detection Prevalence : 0.4489
##       Balanced Accuracy : 0.7513
##
##        'Positive' Class : 0
##
```

Our pruned tree has a slightly higher accuracy than the non-pruned model.

Random Forest

We can further improve our predictive accuracy by generating a large number of bootstrapped trees (based on random samples of variables). Also known as random forests. This large number of bootstrapped trees classifies a case using each tree in the new "forest" and decides a final predicted outcome by combining the results across all the trees (Kabacoff, 2017). We can build a random forest model similarly to our decision trees above, but the **randomForest()** function is used instead.

```
wine_forest <- randomForest(quality ~., wineTrain, method = "class")
print(wine_forest)
```

```
##
## Call:
##  randomForest(formula = quality ~ ., data = wineTrain, method = "class")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 20.54%
## Confusion matrix:
##     0   1 class.error
## 0 408 113   0.2168906
## 1 117 482   0.1953255
```

Creating a confusion matrix,

```
wine_forest_test_pred <- predict(wine_forest, newdata = wineTest, type =
"class")
confusionMatrix(wine_forest_test_pred, test_labels)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

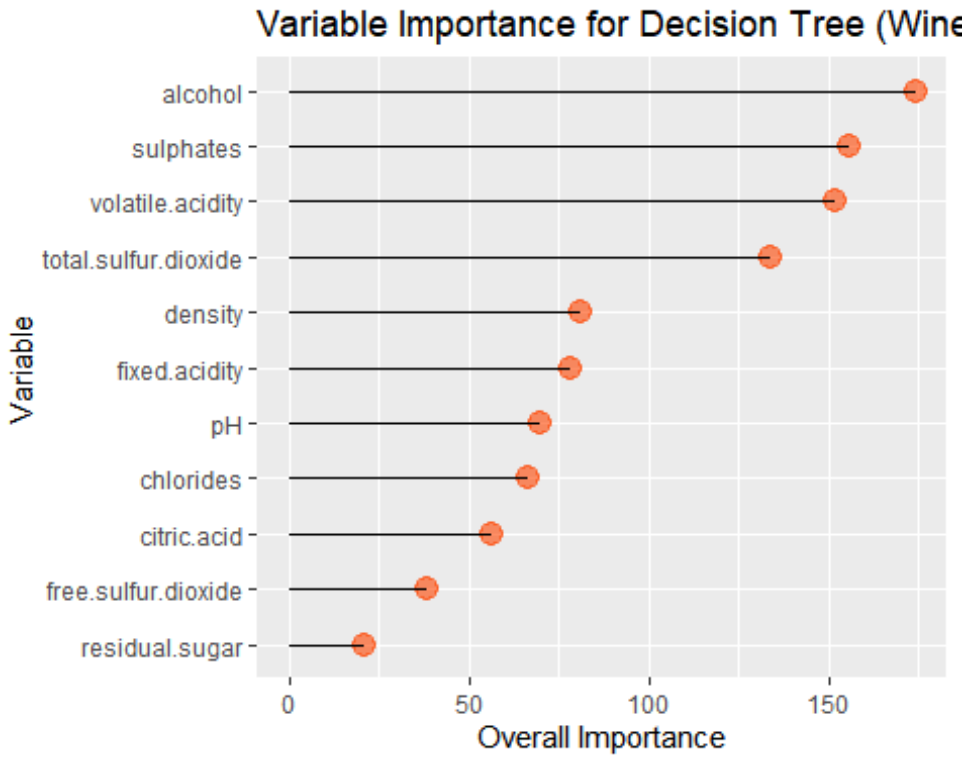
```
##          0 181  32
##          1  42 224
##
##                Accuracy : 0.8455
##                  95% CI : (0.81, 0.8767)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6886
##
##  Mcnemar's Test P-Value : 0.2955
##
##             Sensitivity : 0.8117
##             Specificity : 0.8750
##          Pos Pred Value : 0.8498
##          Neg Pred Value : 0.8421
##              Prevalence : 0.4656
##          Detection Rate : 0.3779
##    Detection Prevalence : 0.4447
##       Balanced Accuracy : 0.8433
##
##        'Positive' Class : 0
##
```

we see that our predictive accuracy is now ~84%.

Variable Importance

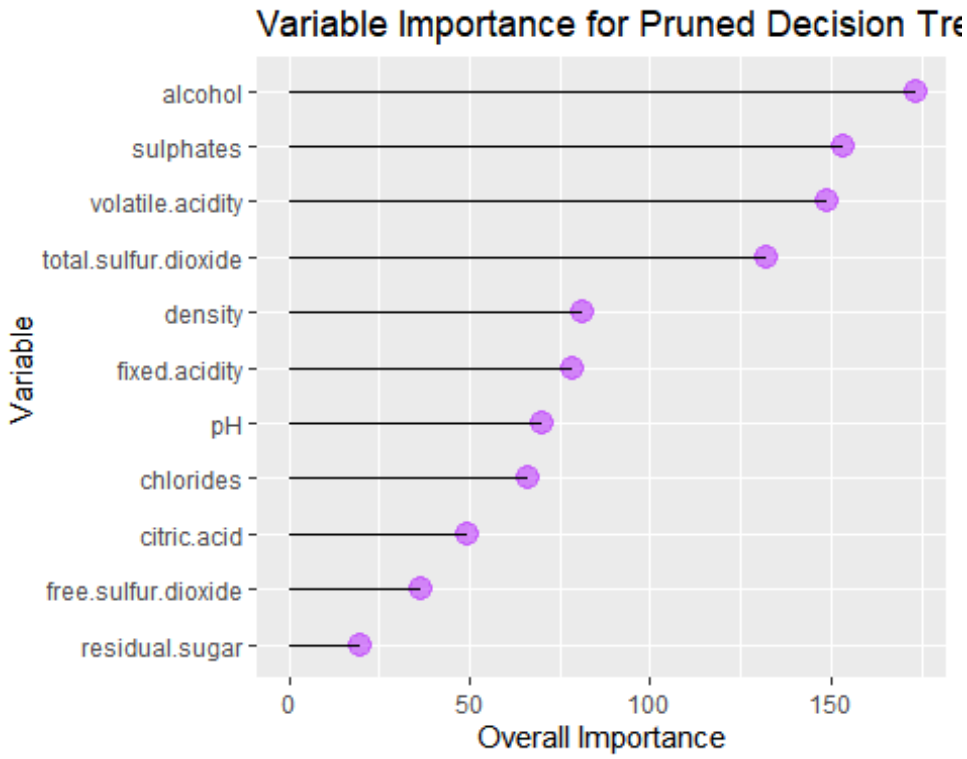
Now that we have our decision tree, pruned tree, and random forest, we can test each model for variable importance. This shows the rank orders of importance of each predictor. For the decision tree and pruned tree, we will use the **varImp()** function and we will use the **importance()** function for the random forest. We will plot the variable importance for each mode. First, we will look at the variable importance for our decision tree. To plot the **varImp()** function, we will code adapted from Lahouir (2020) which utilizes the **ggplot2** package.

```
wine_dtree_imp = varImp(wine_dtree, scale=FALSE)
ggplot(wine_dtree_imp, aes(x=reorder(rownames(wine_dtree_imp),Overall),
y=Overall)) +
  geom_point( color="orangered1", size=4, alpha=0.6)+
  geom_segment(aes(x=rownames(wine_dtree_imp), xend=rownames(wine_dtree_imp),
y=0, yend=Overall),
               color='black') +
  xlab('Variable')+
  ylab('Overall Importance')+ coord_flip()+ggtitle('Variable Importance for
Decision Tree (Wine Data)')
```



The variable with the highest importance is **alcohol** while **residual.sugar** has the lowest importance. Other important predictors include **sulphates**, **volatile.acidity**, and **total.sulfur.dioxide**. Now, we will look at the pruned tree.

```
wine_prun_tree_imp = varImp(prune_wine_dtree, scale=FALSE)

ggplot(wine_prun_tree_imp,
aes(x=reorder(rownames(wine_prun_tree_imp),Overall), y=Overall)) +
geom_point( color="darkorchid1", size=4, alpha=0.6) +
geom_segment(aes(x=rownames(wine_prun_tree_imp),
xend=rownames(wine_prun_tree_imp), y=0, yend=Overall), color='black')   +
xlab('Variable')+  ylab('Overall Importance')+ coord_flip() +
ggtitle('Variable Importance for Pruned Decision Tree (Wine Data)')
```

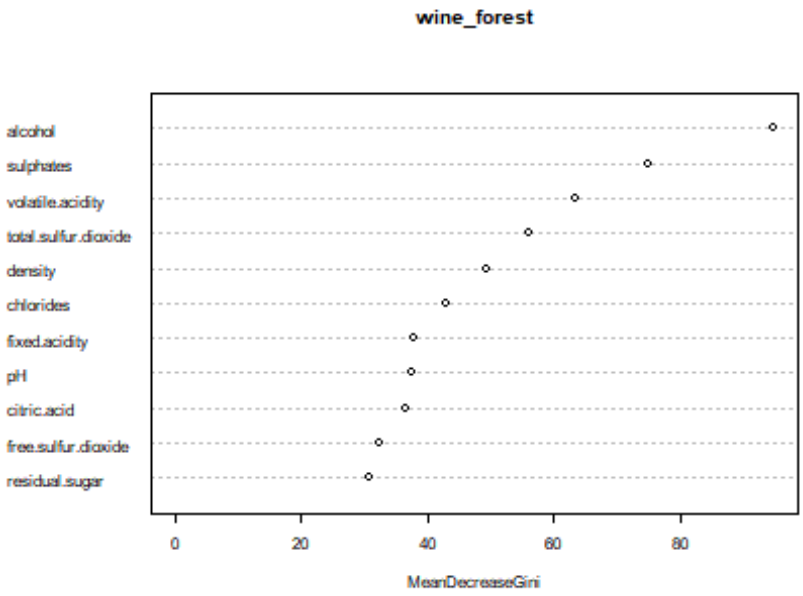


We see similar results for our pruned tree. Finally, we will look at the variable importance for our random forest.

```
importance(wine_forest)
```

```
##                       MeanDecreaseGini
## fixed.acidity                 37.80439
## volatile.acidity              63.36755
## citric.acid                   36.41994
## residual.sugar                30.91046
## chlorides                     42.88213
## free.sulfur.dioxide           32.52809
## total.sulfur.dioxide          56.14479
## density                       49.32184
## pH                            37.43713
## sulphates                     74.87935
## alcohol                       94.82164
```

```
varImpPlot(wine_forest, cex = 0.5)
```



From our outputs above, we can safely assume that the most important predictors when determining the quality of wine include **alcohol**, **sulphates**, **volatile.acidity**, and **total.sulfur.dioxide**.

Conclusion

For this assignment, we used tree-based classification models to predict wine quality rankings from chemical properties. Before we built our models, we loaded our data into R and performed data exploration. From here, we grouped our quality rankings into two groups rather than having individual rankings. A ranking value of 5 or less was converted to a value of 0, indicating a below average ranking. A ranking value of 6 or higher was converted to a value of 1, indicating an above average ranking. Then, we split our data into training and testing datasets so we could train our models with the training dataset and then calculate the predictive accuracy of the model using the testing dataset. Using our training dataset, we created a decision tree and calculated the predictive accuracy. After this, we pruned the tree, which means we simplified/optimized our decision tree by removing unnecessary sections of the tree. Then, we created and tested our random forest model. Of all the models, the most accurate model was the random forest with a predictive accuracy of ~84%. Finally, we tested each model for variable importance and found that the most important predictors included **alcohol**, **sulphates**, **volatile.acidity**, and **total.sulfur.dioxide**.

Resources

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (n.d.). Uci machine learning repository: Wine quality data set. Wine Quality Data Set. Retrieved September 21, 2020, from https://archive.ics.uci.edu/ml/datasets/wine+quality

Chakure, A. (2020, June 7). Random Forest Classification and its implementation. Medium. https://towardsdatascience.com/random-forest-classification-and-its-implementation-d5d840dbead0

Decision trees in r. (n.d.). Retrieved September 21, 2020, from http://www.learnbymarketing.com/tutorials/rpart-decision-trees-in-r/

Dinov, I. (2020, August). Data science and predictive analytics. http://www.socr.umich.edu/people/dinov/courses/DSPA_notes/08_DecisionTreeClass.html#3_Decision_Tree_Overview

Gupta, P. (2017, November 12). Decision trees in machine learning. Medium. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Kabacoff, Robert I. . (2017). Quick-r: Tree-based models. https://www.statmethods.net/advstats/cart.html

Lahouir, Y. (2020, May 31). Plotting varImp in R. Stack Overflow. https://stackoverflow.com/questions/36228559/plotting-varimp-in-r

Read csv from the web. (n.d.). ProgrammingR. Retrieved September 21, 2020, from https://www.programmingr.com/examples/read-csv-web/

Rename data frame columns in r. (n.d.). Datanovia. Retrieved September 21, 2020, from https://www.datanovia.com/en/lessons/rename-data-frame-columns-in-r/

Yu-Wei, C. (2015). Machine Learning with R Cookbook. Packt Publishing.