Wine Recommendation

Decision tree and Random Forest Algorithms

Sean O’Malley

Objective : Predict wine quality ranking from the its chemical properties.

Methodology 1 : Decision Tree

Methodology 2 : Random Forest

Ingest and Explore Data

```
wine <- read.delim("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequa
lity-red.csv",
                   sep = ";",
                   header = T)

wine$quality <- as.factor(wine$quality)

glimpse(wine)
```

```
## Observations: 1,599
## Variables: 12
## $ fixed.acidity        <dbl> 7.4, 7.8, 7.8, 11.2, 7.4, 7.4, 7.9, 7.3, ...
## $ volatile.acidity     <dbl> 0.700, 0.880, 0.760, 0.280, 0.700, 0.660,...
## $ citric.acid          <dbl> 0.00, 0.00, 0.04, 0.56, 0.00, 0.00, 0.06,...
## $ residual.sugar       <dbl> 1.9, 2.6, 2.3, 1.9, 1.9, 1.8, 1.6, 1.2, 2...
## $ chlorides            <dbl> 0.076, 0.098, 0.092, 0.075, 0.076, 0.075,...
## $ free.sulfur.dioxide  <dbl> 11, 25, 15, 17, 11, 13, 15, 15, 9, 17, 15...
## $ total.sulfur.dioxide <dbl> 34, 67, 54, 60, 34, 40, 59, 21, 18, 102, ...
## $ density              <dbl> 0.9978, 0.9968, 0.9970, 0.9980, 0.9978, 0...
## $ pH                   <dbl> 3.51, 3.20, 3.26, 3.16, 3.51, 3.51, 3.30,...
## $ sulphates            <dbl> 0.56, 0.68, 0.65, 0.58, 0.56, 0.56, 0.46,...
## $ alcohol              <dbl> 9.4, 9.8, 9.8, 9.8, 9.4, 9.4, 9.4, 10.0, ...
## $ quality              <fctr> 5, 5, 5, 6, 5, 5, 5, 7, 7, 5, 5, 5, 5, 5...
```

```
summary(wine)
```

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00
##  Median :0.07900   Median :14.00       Median : 38.00
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00
##     density             pH          sulphates        alcohol      quality
##  Min.   :0.9901   Min.   :2.740   Min.   :0.3300   Min.   : 8.40   3: 10
##  1st Qu.:0.9956   1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   4: 53
##  Median :0.9968   Median :3.310   Median :0.6200   Median :10.20   5:681
##  Mean   :0.9967   Mean   :3.311   Mean   :0.6581   Mean   :10.42   6:638
##  3rd Qu.:0.9978   3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   7:199
##  Max.   :1.0037   Max.   :4.010   Max.   :2.0000   Max.   :14.90   8: 18
```

Create Test and Training Datasets

To do so I created a function that splits the data into a 70/30 split and sets a seed. I then ran the wine dataset through my function.

```
split_data <- function(x, p = 0.7, s = 777) {
  set.seed(s)
  index = sample(1:dim(x)[1])
  train = x[index[1:floor(dim(x)[1]*p)],]
  test = x[index[((ceiling(dim(x)[1]*p))+1):dim(x)[1]],]
  return(list(train = train, test = test))
}

wine_split <- split_data(wine)

wine_test <- wine_split$test

wine_train <- wine_split$train
```

Decision Tree Model

Build classification tree model using rpart.

```
wine_rp <- rpart(quality~., data = wine_train)
```

Interpret Model

First, during the process of decision induction, we have to consider a statistic evaluation to partition the data into different partitions in accordance with the assessment result. Then, as we have determined by the child node, we can repeatedly perform the splitting until the stop criteria is satisfied.

Lets see how the model looks by using the printcp function to view the metrics of the complexity parameter.

In the placing wine_rp directly into the console gives us the below metrics for each node detail:
* n indicates sample size

- loss indicates the misclassification cost
- yval stands for the clasified wine value
- yprob stands for the probabilities of the classes

```
wine_rp
```

```
## n= 1119
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1119 654 5 (0.0063 0.035 0.42 0.4 0.13 0.011)
##     2) alcohol< 10.01667 526 193 5 (0.0076 0.034 0.63 0.29 0.034 0.0038)
##       4) sulphates< 0.585 260  63 5 (0.012 0.046 0.76 0.18 0.0038 0) *
##       5) sulphates>=0.585 266 130 5 (0.0038 0.023 0.51 0.39 0.064 0.0075)
##        10) total.sulfur.dioxide>=87.5 37   3 5 (0 0.027 0.92 0.054 0 0) *
##        11) total.sulfur.dioxide< 87.5 229 127 5 (0.0044 0.022 0.45 0.45 0.074 0.0087)
##          22) density< 0.99719 84  29 5 (0 0.024 0.65 0.27 0.036 0.012)
##            44) volatile.acidity>=0.385 71  17 5 (0 0.028 0.76 0.2 0.014 0) *
##            45) volatile.acidity< 0.385 13   4 6 (0 0 0.077 0.69 0.15 0.077) *
##          23) density>=0.99719 145  66 6 (0.0069 0.021 0.32 0.54 0.097 0.0069)
##            46) free.sulfur.dioxide>=22.5 18   8 5 (0 0.17 0.56 0.17 0.11 0) *
##            47) free.sulfur.dioxide< 22.5 127  51 6 (0.0079 0 0.29 0.6 0.094 0.0079) *
##     3) alcohol>=10.01667 593 296 6 (0.0051 0.035 0.22 0.5 0.22 0.017)
##       6) alcohol< 11.55 422 201 6 (0.0071 0.047 0.29 0.52 0.13 0.0071)
##        12) volatile.acidity>=0.375 311 142 6 (0.0096 0.061 0.32 0.54 0.058 0.0032)
##          24) total.sulfur.dioxide>=76 31  10 5 (0 0.032 0.68 0.29 0 0) *
##          25) total.sulfur.dioxide< 76 280 120 6 (0.011 0.064 0.29 0.57 0.064 0.0036)
##            50) sulphates< 0.595 102  59 5 (0.029 0.13 0.42 0.4 0.02 0)
##             100) total.sulfur.dioxide< 25.5 48  23 5 (0.062 0.17 0.52 0.21 0.042 0) *
##             101) total.sulfur.dioxide>=25.5 54  23 6 (0 0.093 0.33 0.57 0 0) *
##            51) sulphates>=0.595 178  59 6 (0 0.028 0.21 0.67 0.09 0.0056) *
##        13) volatile.acidity< 0.375 111  59 6 (0 0.009 0.18 0.47 0.32 0.018)
##          26) pH>=3.265 65  27 6 (0 0.015 0.22 0.58 0.18 0) *
##          27) pH< 3.265 46  22 7 (0 0 0.13 0.3 0.52 0.043) *
##       7) alcohol>=11.55 171  95 6 (0 0.0058 0.064 0.44 0.44 0.041)
##        14) sulphates< 0.695 94  42 6 (0 0.011 0.11 0.55 0.31 0.021)
##          28) total.sulfur.dioxide>=15.5 69  25 6 (0 0.014 0.13 0.64 0.19 0.029) *
##          29) total.sulfur.dioxide< 15.5 25   9 7 (0 0 0.04 0.32 0.64 0) *
##        15) sulphates>=0.695 77  30 7 (0 0 0.013 0.31 0.61 0.065)
##          30) free.sulfur.dioxide>=18.5 27  11 6 (0 0 0.037 0.59 0.3 0.074) *
##          31) free.sulfur.dioxide< 18.5 50  11 7 (0 0 0 0.16 0.78 0.06) *
```

Lets see how the model looks by using the printcp function to view the metrics of the complexity parameter. The CP serves as a penalty to control the size of the tree. In short, the greater the CP value, the fewer the number of splits there are.

- The output value represents the average deviance of the current tree divided by the average deviance of the null tree.
- A xerror value represents the relative error estimated by a 10-fold classification.
- xstd stands for the standard error of the relative error.

We see in the output below that as number of splits increase, the standard and relative error decrease, though their relationship to each other stays the same throughout. Also note the variable importance output and the magnitude of that importance, alcohol matters highly and residual sugar matters very little for example. We can see the tree output here, but its honestly best that we visualize this in a tree for better understanding.

```
printcp(wine_rp)
```

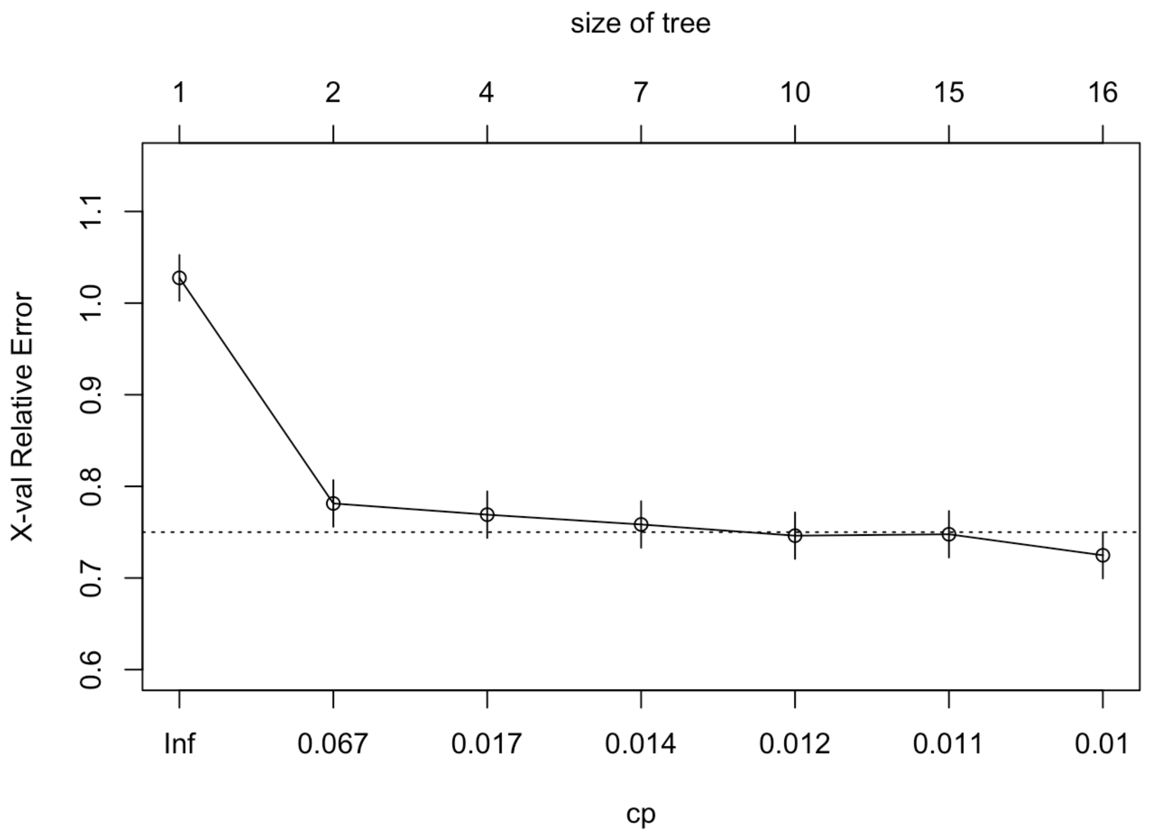
```
## 
## Classification tree:
## rpart(formula = quality ~ ., data = wine_train)
## 
## Variables actually used in tree construction:
## [1] alcohol              density              free.sulfur.dioxide 
## [4] pH                   sulphates            total.sulfur.dioxide
## [7] volatile.acidity    
## 
## Root node error: 654/1119 = 0.58445
## 
## n= 1119 
## 
##         CP nsplit rel error  xerror     xstd
## 1 0.252294      0   1.00000 1.02752 0.025052
## 2 0.017584      1   0.74771 0.78135 0.025478
## 3 0.016310      3   0.71254 0.76911 0.025444
## 4 0.012232      6   0.66361 0.75841 0.025409
## 5 0.011213      9   0.62691 0.74618 0.025365
## 6 0.010703     14   0.57034 0.74771 0.025371
## 7 0.010000     15   0.55963 0.72477 0.025274
```

Lets now graphically view the complexity parameters. plotcp generates an information graphic of the CP table.

- x axis illistrates the cp value
- y axis illistrates the relative error
- upper x axis displays the size of the tree
- dotted line indicates the upper limit of the standard deviation

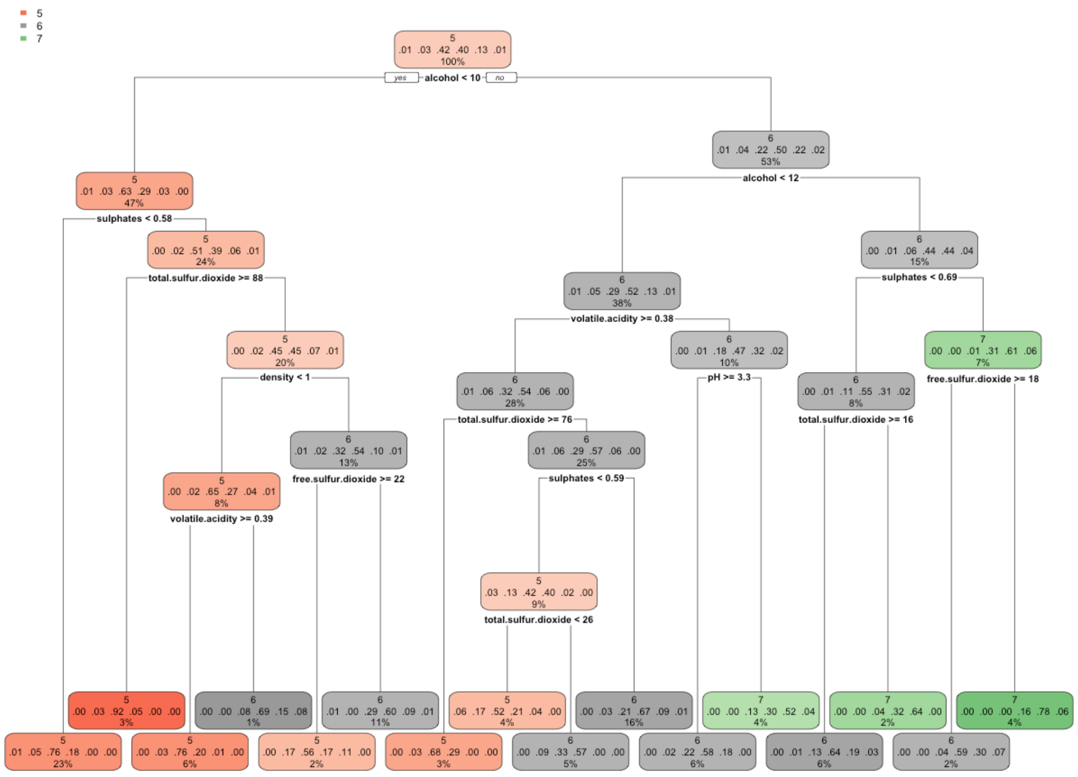
Therefore we can determine that minimum cross validation error occurs when the tree is at a size of 16

```
plotcp(wine_rp)
```



Plotting our initial decision tree model to get a better view of splits and the metrics associated left us with some interesting output. Due to the much larger sample size of the 5,7,8 qualities, we really only see those classifications represented in our decision tree. This means that at best our model can be at best 94 percent accurate given the lower volume negated by our model.

```
rpart.plot(wine_rp)
```


5
6
7
yes
alcohol < 10
no
alcohol < 12
sulphates < 0.58
total.sulfur.dioxide >= 88
sulphates < 0.69
volatile.acidity >= 0.38
density < 1
pH >= 3.3
free.sulfur.dioxide >= 18
total.sulfur.dioxide >= 76
total.sulfur.dioxide >= 16
free.sulfur.dioxide >= 22
sulphates < 0.59
volatile.acidity >= 0.39
total.sulfur.dioxide < 26

Predict Model and Conclusion

Use predict funtion to generate a classification table for the testing dataset.

```
predictions <- predict(wine_rp, wine_test, type = "class")

table(wine_test$quality, predictions)
```

```
##    predictions
##       3   4   5   6   7   8
##   3   0   0   3   0   0   0
##   4   0   0   9   4   1   0
##   5   0   0 135  78   3   0
##   6   0   0  46 125  18   0
##   7   0   0   5  24  22   0
##   8   0   0   0   2   4   0
```

We can see that the model looks like its doing okay, but not great at predicting the quality of a wine, nevertheless lets visualize it a little more easily via a caret’s confusion matrix function. Looking at the confusion matrix we continue to see the limitations of our decision tree model and its inability to pick up on the classes of small sample size. The model is only about 58% accurate, but has pretty solid sensitivity and specificity numbers within the 3 classes we are actually set out to predict.

```
confusionMatrix(table(predictions, wine_test$quality))
```

```
## 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   3   9 135  46   5   0
##           6   0   4  78 125  24   2
##           7   0   1   3  18  22   4
##           8   0   0   0   0   0   0
## 
## Overall Statistics
##                                           
##                Accuracy : 0.5887          
##                  95% CI : (0.5432, 0.6332)
##     No Information Rate : 0.4509          
##     P-Value [Acc > NIR] : 9.946e-10       
##                                           
##                   Kappa : 0.3269          
##  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   0.6250   0.6614  0.43137  0.00000
## Specificity          1.000000  1.00000   0.7605   0.6276  0.93925  1.00000
## Pos Pred Value            NaN      NaN   0.6818   0.5365  0.45833      NaN
## Neg Pred Value       0.993737  0.97077   0.7117   0.7398  0.93271  0.98747
## Prevalence           0.006263  0.02923   0.4509   0.3946  0.10647  0.01253
## Detection Rate       0.000000  0.00000   0.2818   0.2610  0.04593  0.00000
## Detection Prevalence 0.000000  0.00000   0.4134   0.4864  0.10021  0.00000
## Balanced Accuracy    0.500000  0.50000   0.6927   0.6445  0.68531  0.50000
```

Overall I find that the decision tree model, though non-optimal, would not benefit from pruning too much given the problems with sample size in the given dataset.

Random Forest Model

The next method we will use to classify the quality of wine will be the random forest method, which is essentially an ensemble model of decision trees combining the base principles of bagging with random feature selection to add additional diversity to the decision tree models.

After the ensemble of trees is generated, the model uses a vote to combine the trees' predictions. The ensemble uses only a small, random portion of the full feature set, random forests can handle extremely large datasets, and its error rates for most learning tasks are sufficient comparatively.

```
wine_rf <- randomForest(quality~., data = wine_train)
```

Interpret Model

Lets have a look to see what the model we build looks like. We see the default 500 tree split tied to 3 variables at each split. Also note the out of bag error rate (unbiased estimate of the test set error), a number of 32.26% is not wonderful, but if it holds true, is a better predictor than our decision tree model. Once again, in the confusion matrix, note the non-recognition of the 3 small sample size wine quality numbers…in fact, note the larger the sample size of a certain class the lower the error.

```
wine_rf
```

```
##
## Call:
##  randomForest(formula = quality ~ ., data = wine_train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 32.26%
## Confusion matrix:
##   3 4   5   6  7 8 class.error
## 3 0 0   6   1  0 0   1.0000000
## 4 0 0  22  17  0 0   1.0000000
## 5 0 0 359 103  3 0   0.2279570
## 6 0 2  94 325 27 0   0.2745536
## 7 0 0   5  68 74 1   0.5000000
## 8 0 0   0   5  7 0   1.0000000
```

Predict Model and Conclusion

Now lets see how well we predict the test dataset, not bad! Looks like we have properly predicted around 67% of the wine quality numbers, with a decent p value and sensitivity and specificity the expands past the 3 larger classes. This is a large improvement in at the very least robustness in the model without overfitting.

```
predictions <- predict(wine_rf, wine_test, type = "response")

confusionMatrix(table(predictions, wine_test$quality))
```

```
## Confusion Matrix and Statistics
##
##
## predictions   3   4   5   6   7   8
##           3   0   0   0   0   0   0
##           4   1   0   1   0   0   0
##           5   2   9 169  44   5   0
##           6   0   4  45 133  22   1
##           7   0   1   1  12  24   4
##           8   0   0   0   0   0   1
##
## Overall Statistics
##
##                Accuracy : 0.6827
##                  95% CI : (0.6389, 0.7242)
##     No Information Rate : 0.4509
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4764
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.000000   0.7824   0.7037  0.47059 0.166667
## Specificity          1.000000 0.995699   0.7719   0.7517  0.95794 1.000000
## Pos Pred Value            NaN 0.000000   0.7380   0.6488  0.57143 1.000000
## Neg Pred Value       0.993737 0.970650   0.8120   0.7956  0.93822 0.989540
## Prevalence           0.006263 0.029228   0.4509   0.3946  0.10647 0.012526
## Detection Rate       0.000000 0.000000   0.3528   0.2777  0.05010 0.002088
## Detection Prevalence 0.000000 0.004175   0.4781   0.4280  0.08768 0.002088
## Balanced Accuracy    0.500000 0.497849   0.7771   0.7277  0.71427 0.583333
```

To better understand, lets set up a tuning grid for both our random forest. The tuning grid defines defines how many features are randomly selected at each split. We will then supply the resulting grid to the train function with the ctrl object as follows to better determine predictive power is negligible, making me feel more confident in our initial random forest model.

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)

grid_rf <- expand.grid(.mtry = c(2,4,8))

m_rf <- train(quality~., data = wine_train, method = "rf", metric = "Kappa", trControl = ctrl, tun
eGrid = grid_rf)

m_rf
```

```
## Random Forest
##
## 1119 samples
##   11 predictor
##    6 classes: '3', '4', '5', '6', '7', '8'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 1006, 1007, 1008, 1006, 1005, 1009, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.6701360  0.4667519
##   4     0.6698425  0.4680153
##   8     0.6646476  0.4620515
##
## Kappa was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 4.
```

In the end I find that the random forest model is much more predictive in that it has greater breadth in predictive power, while also avoiding the problem of overfitting and predicting much more successfully than the decision tree. If it is however a wine maker interested in what makes a wine quality what it is via variable weight, it could be best to use a decision tree for decision making visability.