Statistical Method Illustration - Classification Methods

Daniel Shang

```
# Load the necessary packages
library(tidylog)
```

```
##
## Attaching package: 'tidylog'

## The following object is masked from 'package:stats':
##
##     filter
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.3
```

```
library(ggplot2)
library(party)
```

```
## Warning: package 'party' was built under R version 4.0.3

## Loading required package: grid

## Loading required package: mvtnorm

## Warning: package 'mvtnorm' was built under R version 4.0.3

## Loading required package: modeltools

## Warning: package 'modeltools' was built under R version 4.0.3

## Loading required package: stats4

## Loading required package: strucchange

## Warning: package 'strucchange' was built under R version 4.0.3

## Loading required package: zoo

##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
## Warning: package 'sandwich' was built under R version 4.0.3
```

```
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.0.3
```

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
# Load the data, remove missing values (if any), and convert columns to proper formats
## based on the documentation of the dataset
data = read.csv('heart.csv', )
data_clean = na.omit(mutate_all(data,
                      ~ifelse(. %in% c("N/A", "null", "", NULL),  NA, .)))
```

```
## mutate_all: no changes
```

```
colnames(data_clean)[1] = 'age'
data_clean$sex = as.factor(data_clean$sex)
data_clean$cp = as.factor(data_clean$cp)
data_clean$fbs = as.factor(data_clean$fbs)
data_clean$restecg = as.factor(data_clean$restecg)
data_clean$exang = as.factor(data_clean$exang)
data_clean$slope = as.factor(data_clean$slope)
data_clean$ca = as.factor(data_clean$ca)
data_clean$thal = as.factor(data_clean$thal)
data_clean$target = as.factor(data_clean$target)
```

```r
# Build a support vector machine (SVM) to predict the 'target' and summarize the model
svm_model = svm(target ~ ., data = data_clean, kernel = 'linear')
summary(svm_model)
```

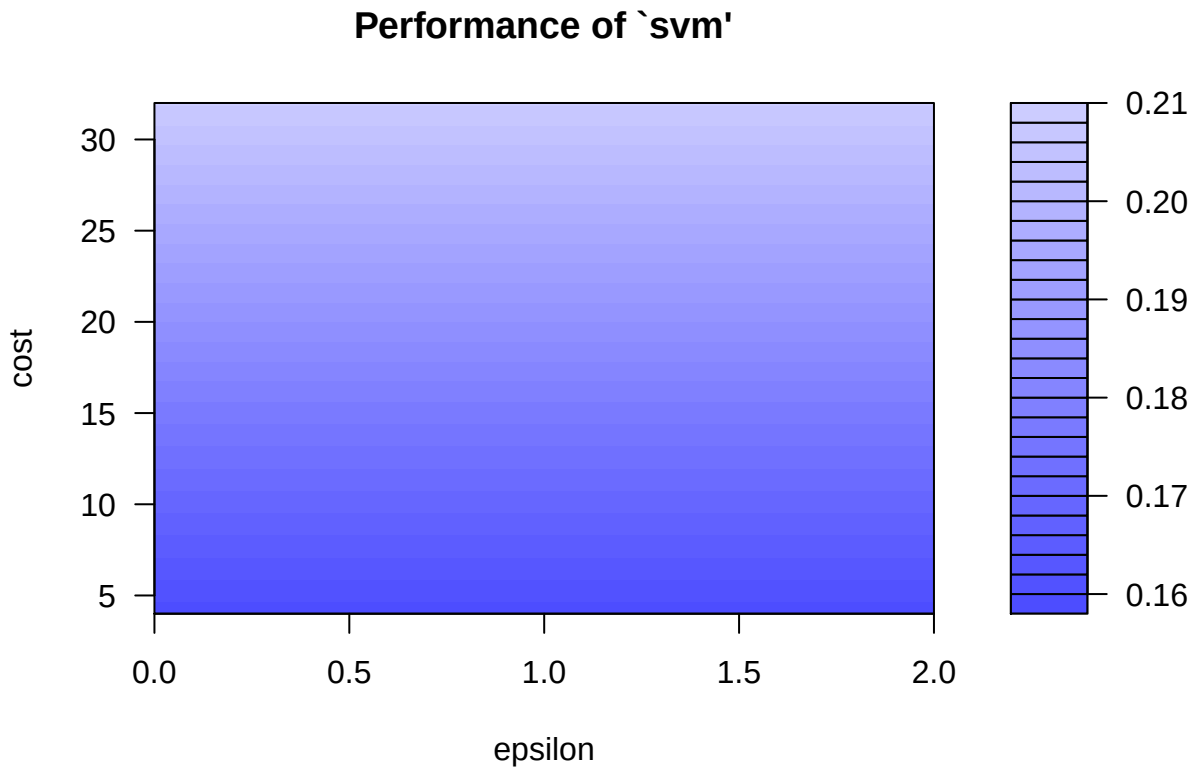
```
##
## Call:
## svm(formula = target ~ ., data = data_clean, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  114
##
##  ( 60 54 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
# Make predictions using the model and compare the outcome with the 'target' values
## in the dataset. Summarize the prediction accuracy and related statistics using
## a confusion matrix
set.seed(123)
prediction_svm1 = predict(svm_model, data_clean)
confusionMatrix(prediction_svm1, data_clean$target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 112  11
##          1  26 154
##
##                Accuracy : 0.8779
##                  95% CI : (0.8356, 0.9125)
##     No Information Rate : 0.5446
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7516
##
##  Mcnemar's Test P-Value : 0.02136
##
##             Sensitivity : 0.8116
##             Specificity : 0.9333
##          Pos Pred Value : 0.9106
##          Neg Pred Value : 0.8556
##              Prevalence : 0.4554
```

```
##          Detection Rate : 0.3696
##    Detection Prevalence : 0.4059
##       Balanced Accuracy : 0.8725
##
##        'Positive' Class : 0
##
```

```
# Use the 'tune' formula to figure out the best parameters for the SVM model to
## boost the model performance. The darker the color is, the better the model will
## perform, as indicated by the 'cost' y-axis label.
set.seed(123)
tune_svm = tune(svm, target ~ ., data = data_clean, range = list(epsilon = seq(0, 2, 0.1), cost = 2^(2:5
plot(tune_svm)
```




```
# Summarize the tuned model
summary(tune_svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  epsilon cost
##        0    4
```

```
##
## - best performance: 0.1589247
##
## - Detailed performance results:
##    epsilon cost     error dispersion
## 1      0.0    4 0.1589247 0.06874025
## 2      0.1    4 0.1589247 0.06874025
## 3      0.2    4 0.1589247 0.06874025
## 4      0.3    4 0.1589247 0.06874025
## 5      0.4    4 0.1589247 0.06874025
## 6      0.5    4 0.1589247 0.06874025
## 7      0.6    4 0.1589247 0.06874025
## 8      0.7    4 0.1589247 0.06874025
## 9      0.8    4 0.1589247 0.06874025
## 10     0.9    4 0.1589247 0.06874025
## 11     1.0    4 0.1589247 0.06874025
## 12     1.1    4 0.1589247 0.06874025
## 13     1.2    4 0.1589247 0.06874025
## 14     1.3    4 0.1589247 0.06874025
## 15     1.4    4 0.1589247 0.06874025
## 16     1.5    4 0.1589247 0.06874025
## 17     1.6    4 0.1589247 0.06874025
## 18     1.7    4 0.1589247 0.06874025
## 19     1.8    4 0.1589247 0.06874025
## 20     1.9    4 0.1589247 0.06874025
## 21     2.0    4 0.1589247 0.06874025
## 22     0.0    8 0.1654839 0.07909940
## 23     0.1    8 0.1654839 0.07909940
## 24     0.2    8 0.1654839 0.07909940
## 25     0.3    8 0.1654839 0.07909940
## 26     0.4    8 0.1654839 0.07909940
## 27     0.5    8 0.1654839 0.07909940
## 28     0.6    8 0.1654839 0.07909940
## 29     0.7    8 0.1654839 0.07909940
## 30     0.8    8 0.1654839 0.07909940
## 31     0.9    8 0.1654839 0.07909940
## 32     1.0    8 0.1654839 0.07909940
## 33     1.1    8 0.1654839 0.07909940
## 34     1.2    8 0.1654839 0.07909940
## 35     1.3    8 0.1654839 0.07909940
## 36     1.4    8 0.1654839 0.07909940
## 37     1.5    8 0.1654839 0.07909940
## 38     1.6    8 0.1654839 0.07909940
## 39     1.7    8 0.1654839 0.07909940
## 40     1.8    8 0.1654839 0.07909940
## 41     1.9    8 0.1654839 0.07909940
## 42     2.0    8 0.1654839 0.07909940
## 43     0.0   16 0.1786022 0.06553252
## 44     0.1   16 0.1786022 0.06553252
## 45     0.2   16 0.1786022 0.06553252
## 46     0.3   16 0.1786022 0.06553252
## 47     0.4   16 0.1786022 0.06553252
## 48     0.5   16 0.1786022 0.06553252
## 49     0.6   16 0.1786022 0.06553252
```

```
## 50     0.7   16 0.1786022 0.06553252
## 51     0.8   16 0.1786022 0.06553252
## 52     0.9   16 0.1786022 0.06553252
## 53     1.0   16 0.1786022 0.06553252
## 54     1.1   16 0.1786022 0.06553252
## 55     1.2   16 0.1786022 0.06553252
## 56     1.3   16 0.1786022 0.06553252
## 57     1.4   16 0.1786022 0.06553252
## 58     1.5   16 0.1786022 0.06553252
## 59     1.6   16 0.1786022 0.06553252
## 60     1.7   16 0.1786022 0.06553252
## 61     1.8   16 0.1786022 0.06553252
## 62     1.9   16 0.1786022 0.06553252
## 63     2.0   16 0.1786022 0.06553252
## 64     0.0   32 0.2082796 0.07285258
## 65     0.1   32 0.2082796 0.07285258
## 66     0.2   32 0.2082796 0.07285258
## 67     0.3   32 0.2082796 0.07285258
## 68     0.4   32 0.2082796 0.07285258
## 69     0.5   32 0.2082796 0.07285258
## 70     0.6   32 0.2082796 0.07285258
## 71     0.7   32 0.2082796 0.07285258
## 72     0.8   32 0.2082796 0.07285258
## 73     0.9   32 0.2082796 0.07285258
## 74     1.0   32 0.2082796 0.07285258
## 75     1.1   32 0.2082796 0.07285258
## 76     1.2   32 0.2082796 0.07285258
## 77     1.3   32 0.2082796 0.07285258
## 78     1.4   32 0.2082796 0.07285258
## 79     1.5   32 0.2082796 0.07285258
## 80     1.6   32 0.2082796 0.07285258
## 81     1.7   32 0.2082796 0.07285258
## 82     1.8   32 0.2082796 0.07285258
## 83     1.9   32 0.2082796 0.07285258
## 84     2.0   32 0.2082796 0.07285258
```

```r
# Use the best model tuned by the function and set it as our final model
set.seed(123)
final_svm = tune_svm$best.model
summary(final_svm)
```

```
##
## Call:
## best.tune(method = svm, train.x = target ~ ., data = data_clean,
##     ranges = list(epsilon = seq(0, 2, 0.1), cost = 2^(2:5)))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  4
##
## Number of Support Vectors:  145
##
```

```
##  ( 71 74 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
# Use the tuned model to make prediction and compare the accuracy with the previous
## model. Since the prediction accuracy increased from 0.8779 to 0.9241, we can
## conclude that the 'tune' function did a great job identifying the best model
prediction_svm2 = predict(final_svm, data_clean)
confusionMatrix(prediction_svm2, data_clean$target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 123   8
##          1  15 157
##
##                Accuracy : 0.9241
##                  95% CI : (0.8883, 0.9513)
##     No Information Rate : 0.5446
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8463
##
##  Mcnemar's Test P-Value : 0.2109
##
##             Sensitivity : 0.8913
##             Specificity : 0.9515
##          Pos Pred Value : 0.9389
##          Neg Pred Value : 0.9128
##              Prevalence : 0.4554
##          Detection Rate : 0.4059
##    Detection Prevalence : 0.4323
##       Balanced Accuracy : 0.9214
##
##        'Positive' Class : 0
##
```

Classification Tree

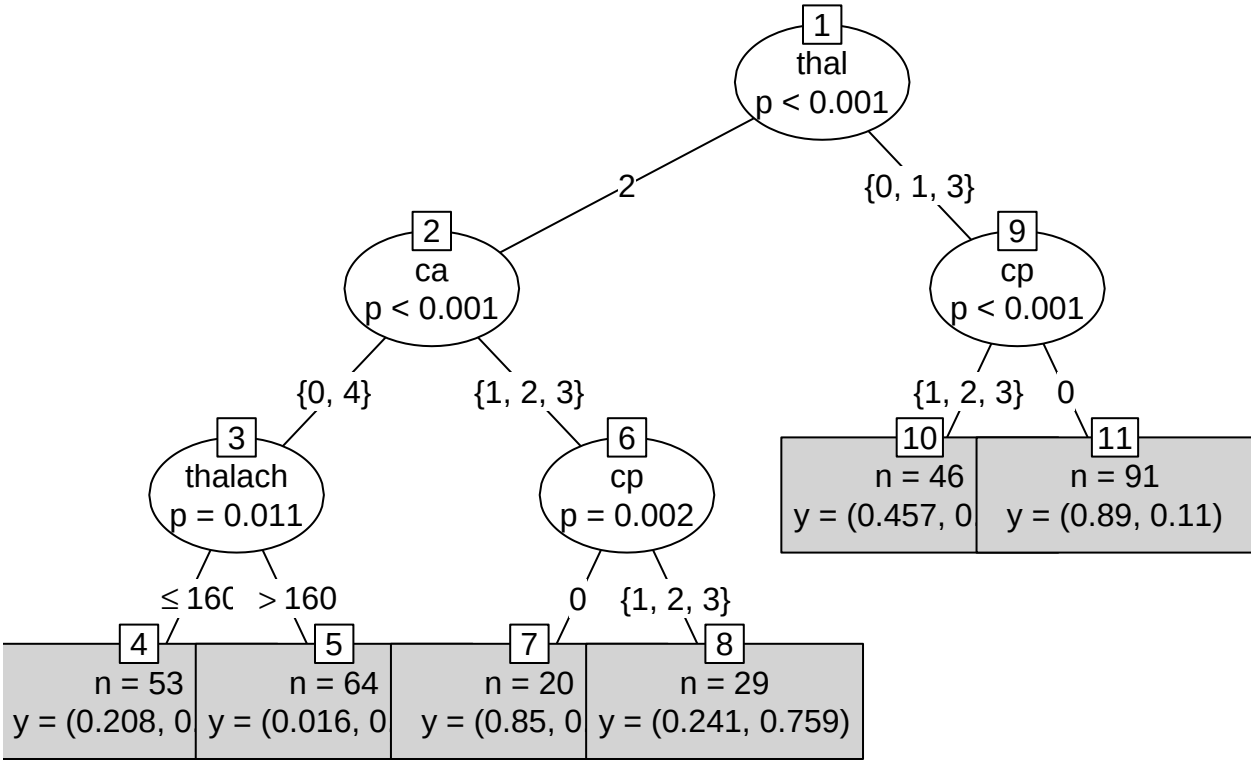
```
# Build a classification tree model to predict the target. I used a tree control
## parameter 'mincriterion.' The value of this parameter will be considered as
## 1 - p-value that must be exceeded in order to implement a node split.
tree_model1 = ctree(target~., data = data_clean, controls = ctree_control(mincriterion = 0.95))
summary(tree_model1)
```

```
##     Length      Class       Mode
##          1 BinaryTree         S4
```

```
tree_model1
```

```
##
##   Conditional inference tree with 6 terminal nodes
##
## Response:  target
## Inputs:  age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal
## Number of observations:  303
##
## 1) thal == {2}; criterion = 1, statistic = 85.022
##   2) ca == {0, 4}; criterion = 1, statistic = 35.631
##     3) thalach <= 160; criterion = 0.989, statistic = 11.213
##       4)*  weights = 53
##     3) thalach > 160
##       5)*  weights = 64
##   2) ca == {1, 2, 3}
##     6) cp == {0}; criterion = 0.998, statistic = 19.826
##       7)*  weights = 20
##     6) cp == {1, 2, 3}
##       8)*  weights = 29
## 1) thal == {0, 1, 3}
##   9) cp == {1, 2, 3}; criterion = 1, statistic = 31.784
##     10)*  weights = 46
##   9) cp == {0}
##     11)*  weights = 91
```

```
# Plot the tree model built
plot(tree_model1, type = 'simple')
```



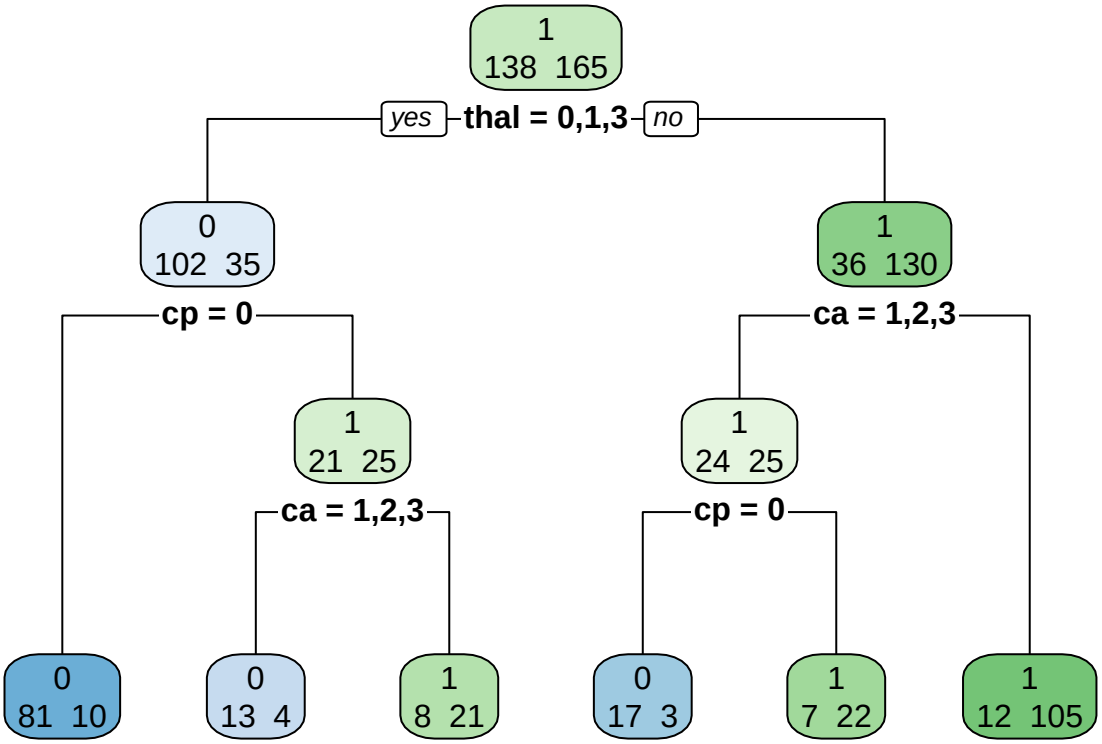

```
# Make prediction using the tree model and build a confusion matrix to evaluate
## its prediction accuracy
prediction_tree1 = predict(tree_model1, data = data_clean)
confusionMatrix(prediction_tree1, data_clean$target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  98  13
##          1  40 152
##
##                Accuracy : 0.8251
##                  95% CI : (0.7775, 0.8661)
##     No Information Rate : 0.5446
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6416
##
##  Mcnemar's Test P-Value : 0.0003551
##
##             Sensitivity : 0.7101
##             Specificity : 0.9212
##          Pos Pred Value : 0.8829
##          Neg Pred Value : 0.7917
##              Prevalence : 0.4554
```

```
##          Detection Rate : 0.3234
##    Detection Prevalence : 0.3663
##       Balanced Accuracy : 0.8157
##
##        'Positive' Class : 0
##
```

```r
# Build another tree model using a different package
tree_model2 = rpart(target ~ ., data = data_clean)
```

```r
# Plot the tree at a certain level of detail
rpart.plot(tree_model2, extra = 1)
```




```r
# Make prediction using the second tree model and build a confusion matrix to evaluate
## the prediction accuracy
prediction_tree2 = predict(tree_model2, data_clean, type = 'class')
confusionMatrix(prediction_tree2, data_clean$target)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction   0   1
##          0 111  17
##          1  27 148
##
```

```
##                Accuracy : 0.8548
##                  95% CI : (0.81, 0.8925)
##     No Information Rate : 0.5446
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7055
##
##  Mcnemar's Test P-Value : 0.1748
##
##             Sensitivity : 0.8043
##             Specificity : 0.8970
##          Pos Pred Value : 0.8672
##          Neg Pred Value : 0.8457
##              Prevalence : 0.4554
##          Detection Rate : 0.3663
##    Detection Prevalence : 0.4224
##       Balanced Accuracy : 0.8507
##
##        'Positive' Class : 0
##
```

Random Forest

```
# Build a random forest model to predict the 'target' variable in the dataset. I
## started with a huge number of trees (ntree) so that, based on the plot later,
## we can easily identify the number of trees that leads to least prediction error
set.seed(123)
rf_model1 = randomForest(target ~ ., data = data_clean, ntree = 2000)
print(rf_model1)
```

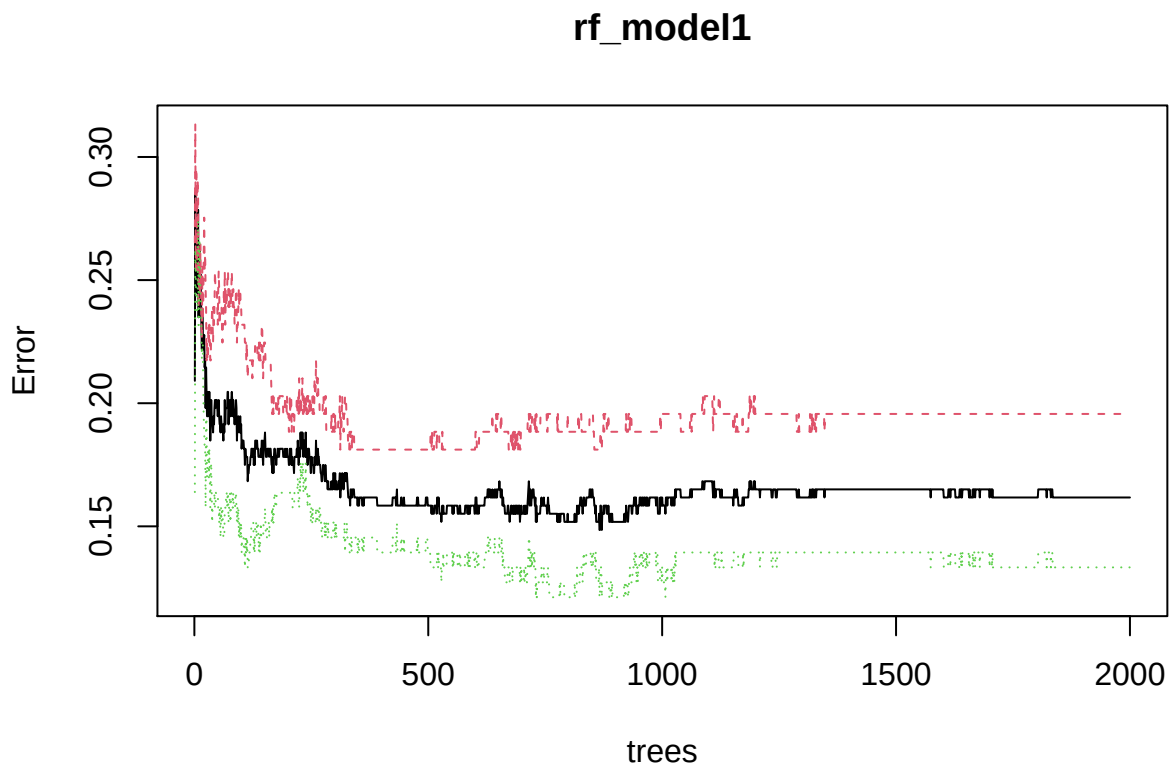
```
##
## Call:
##  randomForest(formula = target ~ ., data = data_clean, ntree = 2000)
##                Type of random forest: classification
##                      Number of trees: 2000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 16.17%
## Confusion matrix:
##     0   1 class.error
## 0 111  27   0.1956522
## 1  22 143   0.1333333
```

```
# Use the random forest model to make prediction and build a confusion matrix to
## evaluate the prediction accuracy
prediction_rf1 = predict(rf_model1, data = data_clean)
confusionMatrix(prediction_rf1, data_clean$target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 111  22
```

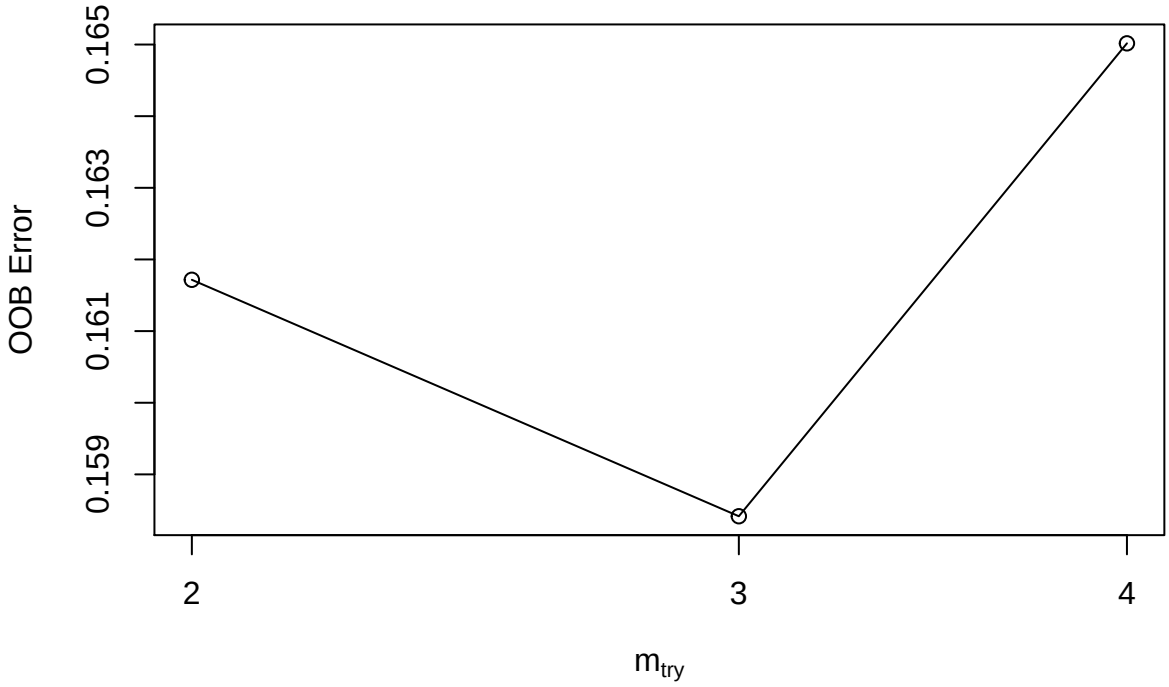
```
##          1  27 143
##
##                Accuracy : 0.8383
##                  95% CI : (0.7919, 0.8779)
##     No Information Rate : 0.5446
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.673
##
##  Mcnemar's Test P-Value : 0.5677
##
##             Sensitivity : 0.8043
##             Specificity : 0.8667
##          Pos Pred Value : 0.8346
##          Neg Pred Value : 0.8412
##              Prevalence : 0.4554
##          Detection Rate : 0.3663
##    Detection Prevalence : 0.4389
##       Balanced Accuracy : 0.8355
##
##        'Positive' Class : 0
##
```

```
# Plot the relationship between the number of trees and the prediction error. We
## can see that the error reaches the lowest point when the number of trees is
## around 750. Therefore, I will use this number to build a new model later to see
## if it does a great job predicting
plot(rf_model1)
```




```
# Use the 'tuneRF' function to figure out the 'mtry' parameter that leads to least
## prediction error. 'mtry' is the number of variables randomly sampled as candidates
## at each split of node. According to the plot, an 'mtry' of three leads to the
## random forest model the predicts most accurately
set.seed(123)
tune_rf1 = tuneRF(data_clean[, -14], data_clean[, 14], stepFactor = 1.5, plot = TRUE, ntreeTry = 750, t
```

```
## mtry = 3  OOB error = 15.84%
## Searching left ...
## mtry = 2     OOB error = 16.17%
## -0.02083333 0.01
## Searching right ...
## mtry = 4     OOB error = 16.5%
## -0.04166667 0.01
```




```
# Build a new model using the parameters we just figured out. We can see that the
## Out Of Bag (OBB) estimate of error rate decreases from 16.17% to 15.84%, meaning
## that the functions did a great job identifying the best parameters
set.seed(123)
rf_model2 = randomForest(target ~ ., data = data_clean, ntree = 750, mtry = 3, importance = TRUE, proxi
print(rf_model2)
```

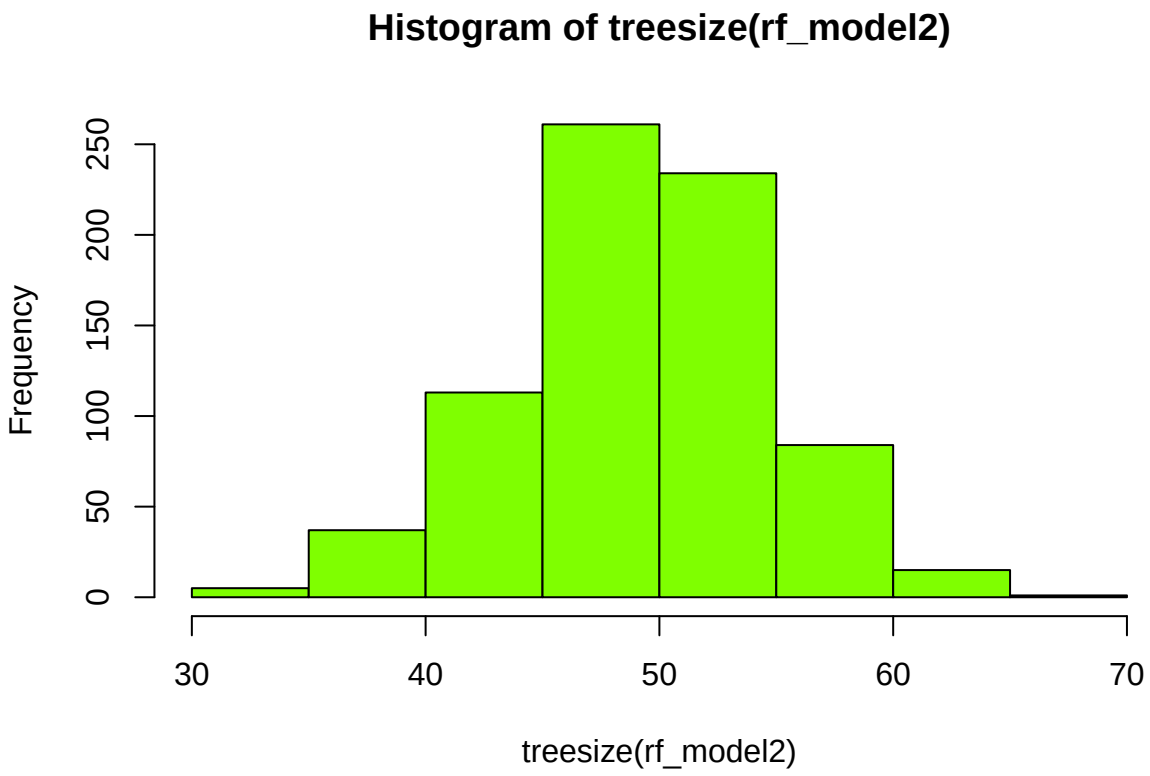
```
##
## Call:
##  randomForest(formula = target ~ ., data = data_clean, ntree = 750,      mtry = 3, importance = TRUE
##                Type of random forest: classification
##                      Number of trees: 750
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 15.84%
## Confusion matrix:
##     0   1 class.error
## 0 112  26   0.1884058
## 1  22 143   0.1333333
```

```
# Build a confusion matrix for more detailed statistics about the model performance
prediction_rf2 = predict(rf_model2, data = data_clean)
confusionMatrix(prediction_rf2, data_clean$target)
```

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

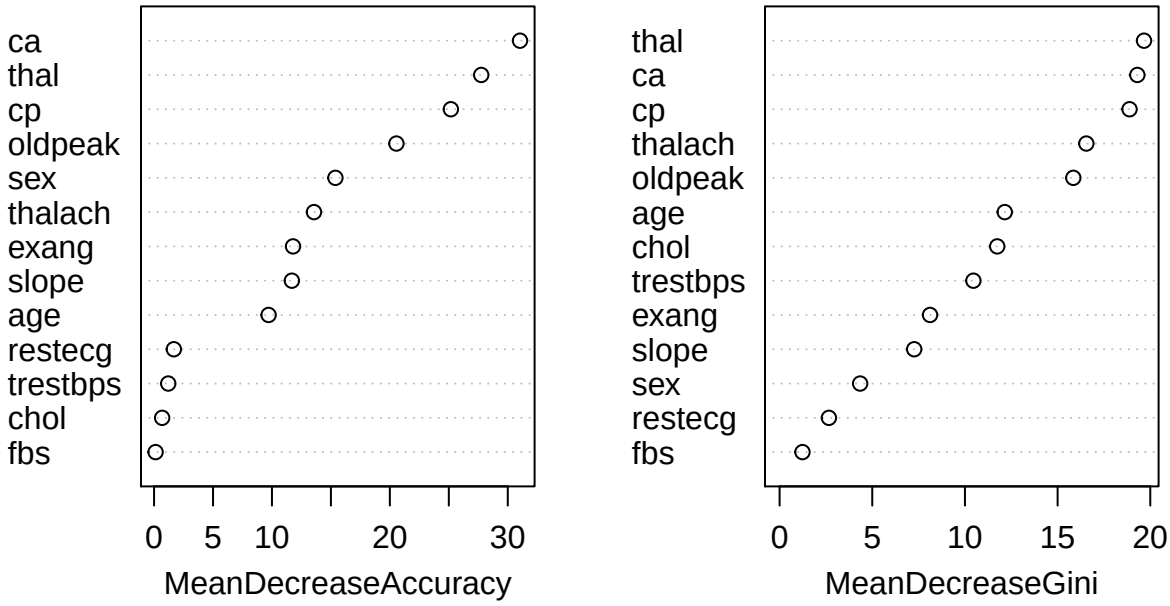
```
##
##           Reference
## Prediction   0   1
##          0 112  22
##          1  26 143
##
##                Accuracy : 0.8416
##                  95% CI : (0.7955, 0.8808)
##     No Information Rate : 0.5446
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6799
##
##  Mcnemar's Test P-Value : 0.665
##
##             Sensitivity : 0.8116
##             Specificity : 0.8667
##          Pos Pred Value : 0.8358
##          Neg Pred Value : 0.8462
##              Prevalence : 0.4554
##          Detection Rate : 0.3696
##    Detection Prevalence : 0.4422
##       Balanced Accuracy : 0.8391
##
##        'Positive' Class : 0
##
```

```r
# Plot the distribution of tree size to better understand the model
hist(treesize(rf_model2), col = 'chartreuse1')
```




```
# Plot all the variables in the dataset and sort them based on their relative
## importance when making the prediction. The first plot gives information about
## how much prediction accuracy will decrease if we remove the variable. For example,
## if we remove 'ca,' the prediction accuracy will decrease by 30%. The second plot
## shows how pure the nodes are at the end of the tree, if the variable is removed.
varImpPlot(rf_model2, main = 'Variable importance (high to low)')
```

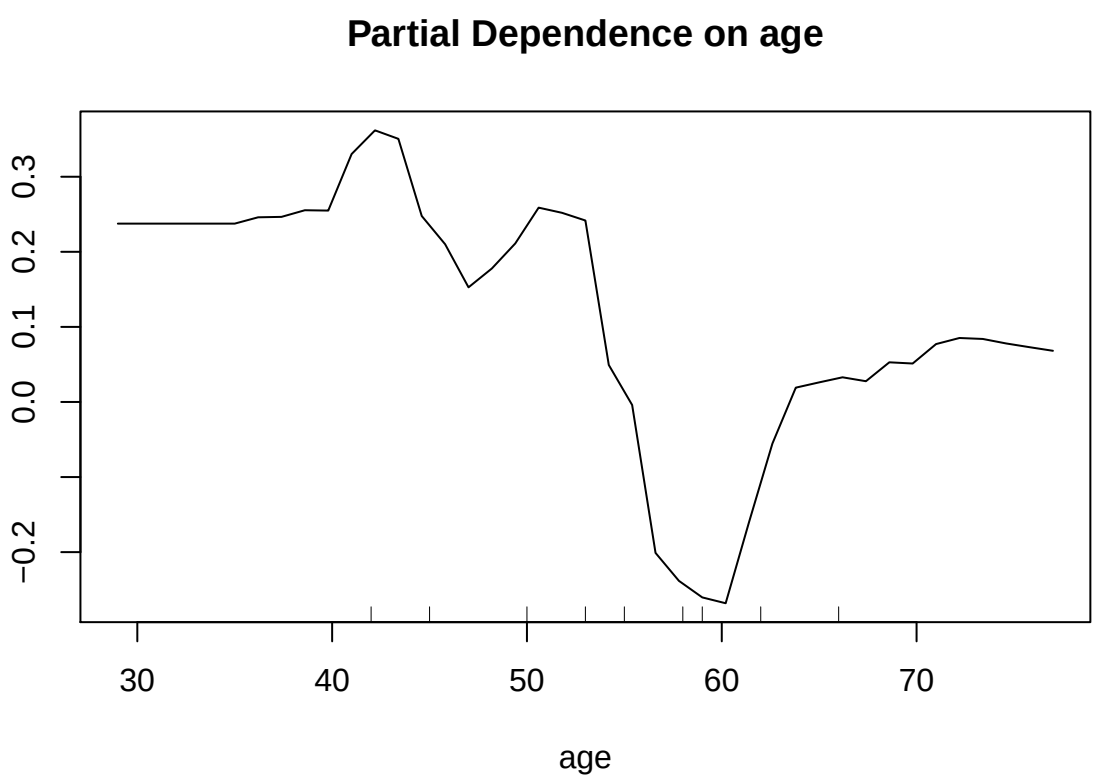
Variable importance (high to low)




```
# To know how many times each column is used in the entire random forest, we can
## use the 'varUsed' function
varUsed(rf_model2)
```

```
##  [1] 4527 1431 2714 4321 4677  749 1462 4830 1309 4057 1795 2670 1962
```

```
# To understand the marginal effect of a variable on the final prediction result,
## we can use the partial dependence plot. For example, this plot shows that, when
## age is greater than 53, the random forest is much less likely to predict
## 1 as the target for that record.
partialPlot(rf_model2, data_clean, age, '1')
```


Partial Dependence on age
0.3
0.2
0.1
0.0
−0.2
30
40
50
60
70
age