HarvardX - PH125.9x CYO Project

Brad Hummel

6/23/2020

Introduction

The objective of the PH125.9x CYO Project is to present two algorithms that predict the presence of heart disease in a patience. The objecive can be broken down into three goals. The first goal is to make a prediction using the classiication algorithm Decision Tree. The second goal is to make a prediction using the ensemble learning Random Forest algorithm with default paramters. The third goal is to make a prediction using Random Forest with optimized tuning parameters.

The database originated from Cleveland and is available via UCI. The database initally contained 76 attributes and a subset of 14 were made available. It consists of 13 predictor variables and 1 predicted vairable called num. The num is a factor variable with 4 values and it was simplified to a binary variable where numbers greater than one are set to two (factor value 1).

Random Forest creates multiple grroups, hundreds or thousands, of Decision Trees that don't use all the predictor variables in any one tree. That's one big difference between the two models, the Decision Trees use all the predictor variables. Random Forest uses hundreds of these parital Decision Trees, each tree randomly choosing a partial variable set, hence the name Random Forest.

Decision trees work well with training data but when new data is introduced it is less accurate. This is due to overfitting, a condition where using training data affects the usage of new data, the test data, ends up producing poor results. Since Random Forest uses many trees, a term called bagging, it ends up producing better results.

Creating a Random Forest consists of five steps. Step 1, a bootstrap set of the data set is created that consists of random selections of the data set by resampling. Step 2, random subsets of the bootstrap data are used to create Decision Trees. Step 3, go back to step 1 and repeat. This is done hudrends of times to create a random forest of trees. Step 4, with the forest created, evaluate each tree to predict a new data point. Step 5, the test set is used to evaluate the training set.

In this project there are a few tuning variables avaialbe for tuning the Random forest to find an optimal solution: mtry, maxnodes, maxtrees.

Methods/Analysis

The data set is loaded from "`https : //archive.ics.uci.edu/ml/machine − learning − databases/heart − disease/processed.cleveland.data`".

First, the data is cleaned. Column headers are added that correspond to the data set website link. The predicted attribute, `num`, is a factor variable that has four values, 1-4. To simlpify the result, values 2-4 are converted to 1. Each of the 14 attributes are converted to an appropriate type. There are ? in the data. These ? are converted to `NA`. The attibutes are identified and rows with `NA` are coerced to factor types and results in these rows being removed.

The data set is partitioned into `train` set and `test` set. The `training` set is used for training and making adjustments to the Random Forest tuning variables. The `test` set is completely ignored during this model development.

The first training was done with just defaults. Nothing was optimized. The next training was done to find the best tunable parameter `mtry`. The next training after that was done to find the best `maxnodes` with the optimized `mtry`. The final training was done to find the best
tmaxtrees with the optimized
tmtryandmaxnodes. Finally, these optimized paramters are used to find the optimal solution.

R version 3.6.3 is provides default packages for basic data analysis. The library caret is used for creating partitions. The library dplyr is available for any additional data wrangling. The library ggplot2 is used for viusalization. The library randomForest is used for the Random Forest algorithm. A few more libraries were added in case any additional functionality is needed.

Detailed Method/Analysis

Install packages.

```r
options(warn = -1)

if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret

## Loading required package: lattice

## Loading required package: ggplot2
```

```r
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
if(!require(tidyr)) install.packages("tidyr", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyr
```

```r
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(rattle)) install.packages("rattle", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rattle

## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```r
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: randomForest

## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':
##
##     importance

## The following object is masked from 'package:dplyr':
##
##     combine

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

```
if(!require(RColorBrewer)) install.packages("RColorBrewer", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: RColorBrewer
```

Load libraries.

```
library(caret)
library(dplyr)
library(tidyr)
library(ggplot2)
library(grid)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:randomForest':
##
##     combine

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(lattice)
library(rpart)
library(rpart.plot)
library(rattle)
library(randomForest)
library(RColorBrewer)
```

Load the Cleveland data set.

```
data <- read.csv(
  "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data",
  header=FALSE
)
```

`Cleanthedataset.`

Add column headings corresponding to the website's headings. Verify the coluumns are correct

```
names(data) <- c("age", "sex", "cp", "trestbps", "choi", "fbs", "restecg", "thalach", "exang", "oldpeak
head(data)
```

```
##   age sex cp trestbps choi fbs restecg thalach exang oldpeak slope  ca thai num
## 1  63   1  1      145  233   1       2     150     0     2.3     3 0.0  6.0   0
## 2  67   1  4      160  286   0       2     108     1     1.5     2 3.0  3.0   2
## 3  67   1  4      120  229   0       2     129     1     2.6     2 2.0  7.0   1
## 4  37   1  3      130  250   0       0     187     0     3.5     3 0.0  3.0   0
## 5  41   0  2      130  204   0       2     172     0     1.4     1 0.0  3.0   0
## 6  56   1  2      120  236   0       0     178     0     0.8     1 0.0  3.0   0
```

Change the
tnum column's range of 1-4 to 1 and verify.

```
data$num[data$num > 1] <- 1
head(data)
```

```
##   age sex cp trestbps choi fbs restecg thalach exang oldpeak slope  ca thai num
## 1  63   1  1      145  233   1       2     150     0     2.3     3 0.0  6.0   0
## 2  67   1  4      160  286   0       2     108     1     1.5     2 3.0  3.0   1
## 3  67   1  4      120  229   0       2     129     1     2.6     2 2.0  7.0   1
## 4  37   1  3      130  250   0       0     187     0     3.5     3 0.0  3.0   0
## 5  41   0  2      130  204   0       2     172     0     1.4     1 0.0  3.0   0
## 6  56   1  2      120  236   0       0     178     0     0.8     1 0.0  3.0   0
```

```
sapply(data, class)
```

```
##       age       sex        cp  trestbps      choi       fbs   restecg   thalach
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
##     exang   oldpeak     slope        ca      thai       num
## "numeric" "numeric" "numeric"  "factor"  "factor" "numeric"
```

Coerce data columns into the appropriate types and verify.

```
data <- transform(
  data,
  age=as.integer(age),
  sex=as.factor(sex),
  cp=as.factor(cp),
  trestbps=as.integer(trestbps),
  choi=as.integer(choi),
  fbs=as.factor(fbs),
  restecg=as.factor(restecg),
  thalach=as.integer(thalach),
  exang=as.factor(exang),
  oldpeak=as.numeric(oldpeak),
  slope=as.factor(slope),
  ca=as.factor(ca),
  thai=as.factor(thai),
  num=as.factor(num)
)

sapply(data, class)
```

```
##       age       sex        cp  trestbps      choi       fbs   restecg   thalach
## "integer"  "factor"  "factor" "integer" "integer"  "factor"  "factor" "integer"
##     exang   oldpeak     slope        ca      thai       num
##  "factor" "numeric"  "factor"  "factor"  "factor"  "factor"
```

Convert ? to `NA`.

```r
data[ data == "?"] <- NA
colSums(is.na(data))
```

```
##      age      sex       cp trestbps     choi      fbs  restecg  thalach
##        0        0        0        0        0        0        0        0
##    exang  oldpeak    slope       ca     thai      num
##        0        0        0        4        2        0
```

```r
summary(data)
```

```
##       age         sex     cp         trestbps          choi       fbs
##  Min.   :29.00   0: 97   1: 23   Min.   : 94.0   Min.   :126.0   0:258
##  1st Qu.:48.00   1:206   2: 50   1st Qu.:120.0   1st Qu.:211.0   1: 45
##  Median :56.00           3: 86   Median :130.0   Median :241.0
##  Mean   :54.44           4:144   Mean   :131.7   Mean   :246.7
##  3rd Qu.:61.00                   3rd Qu.:140.0   3rd Qu.:275.0
##  Max.   :77.00                   Max.   :200.0   Max.   :564.0
##  restecg    thalach      exang      oldpeak     slope      ca         thai
##  0:151   Min.   : 71.0   0:204   Min.   :0.00   1:142   ?   :  0   ?   :  0
##  1:  4   1st Qu.:133.5   1: 99   1st Qu.:0.00   2:140   0.0 :176   3.0 :166
##  2:148   Median :153.0           Median :0.80   3: 21   1.0 : 65   6.0 : 18
##          Mean   :149.6           Mean   :1.04           2.0 : 38   7.0 :117
##          3rd Qu.:166.0           3rd Qu.:1.60           3.0 : 20   NA's:  2
##          Max.   :202.0           Max.   :6.20           NA's:  4
##  num
##  0:164
##  1:139
##
##
##
##
```

```r
data$thai[which(is.na(data$thai))] <- as.factor("3.0")
data <- data[!(data$ca %in% c(NA)),]
colSums(is.na(data))
```

```
##      age      sex       cp trestbps     choi      fbs  restecg  thalach
##        0        0        0        0        0        0        0        0
##    exang  oldpeak    slope       ca     thai      num
##        0        0        0        0        0        0
```

```r
summary(data)
```

```
##       age         sex     cp         trestbps          choi       fbs
##  Min.   :29.00   0: 97   1: 23   Min.   : 94.0   Min.   :126.0   0:255
##  1st Qu.:48.00   1:202   2: 49   1st Qu.:120.0   1st Qu.:211.0   1: 44
##  Median :56.00           3: 84   Median :130.0   Median :242.0
##  Mean   :54.53           4:143   Mean   :131.7   Mean   :247.1
##  3rd Qu.:61.00                   3rd Qu.:140.0   3rd Qu.:275.5
##  Max.   :77.00                   Max.   :200.0   Max.   :564.0
##  restecg    thalach      exang      oldpeak      slope     ca        thai
##  0:148   Min.   : 71.0   0:201   Min.   :0.000   1:140   ?  :  0   ?  :  0
##  1:  4   1st Qu.:133.0   1: 98   1st Qu.:0.000   2:138   0.0:176   3.0:166
##  2:147   Median :153.0           Median :0.800   3: 21   1.0: 65   6.0: 18
##          Mean   :149.5           Mean   :1.052           2.0: 38   7.0:115
##          3rd Qu.:165.5           3rd Qu.:1.600           3.0: 20
##          Max.   :202.0          Max.   :6.200
##  num
##  0:161
##  1:138
##
##
##
##
```

Remove NA rows by coercing to factor.

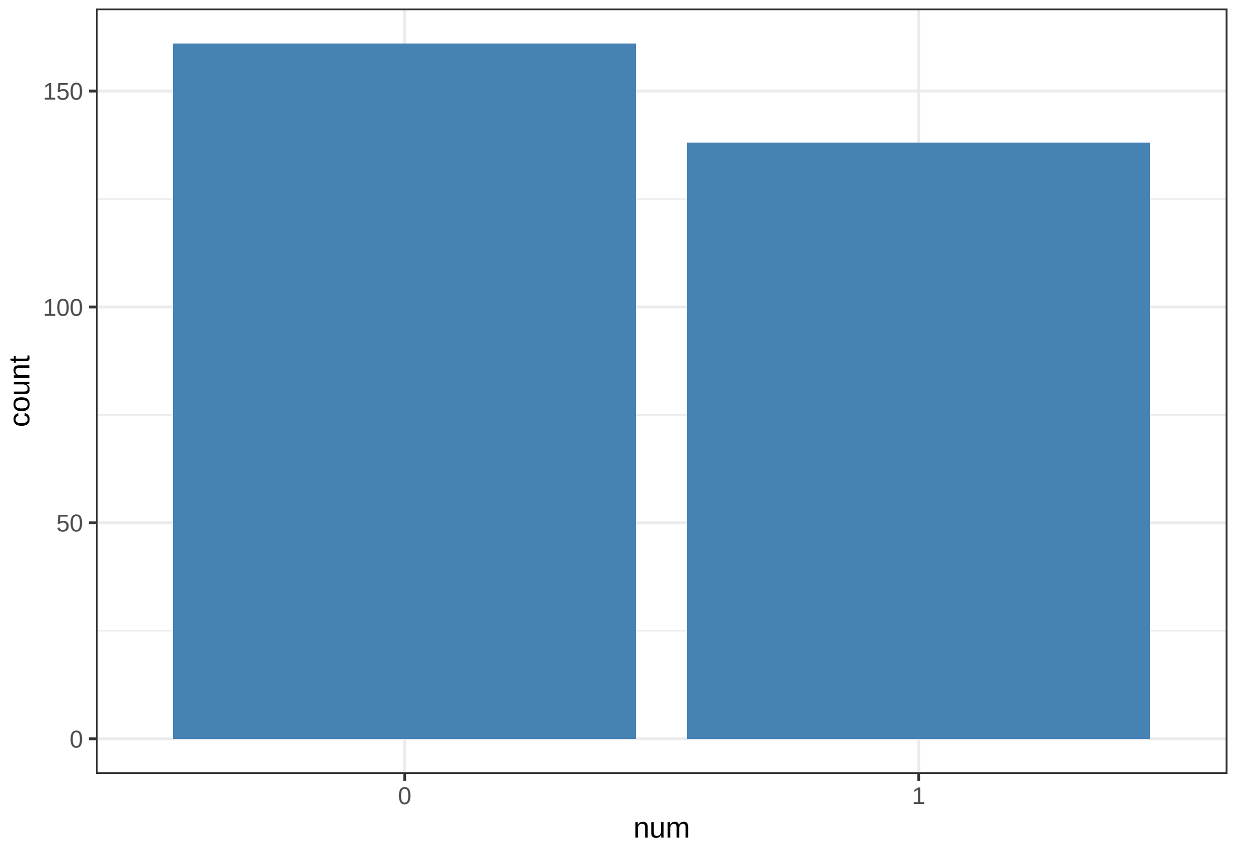
```
data$ca <- factor(data$ca)
data$thai <- factor(data$thai)
summary(data)
```

```
##       age         sex      cp         trestbps          choi       fbs
##  Min.   :29.00   0: 97   1: 23   Min.   : 94.0   Min.   :126.0   0:255
##  1st Qu.:48.00   1:202   2: 49   1st Qu.:120.0   1st Qu.:211.0   1: 44
##  Median :56.00           3: 84   Median :130.0   Median :242.0
##  Mean   :54.53           4:143   Mean   :131.7   Mean   :247.1
##  3rd Qu.:61.00                   3rd Qu.:140.0   3rd Qu.:275.5
##  Max.   :77.00                   Max.   :200.0   Max.   :564.0
##  restecg    thalach       exang     oldpeak       slope     ca        thai
##  0:148   Min.   : 71.0   0:201   Min.   :0.000   1:140   0.0:176   3.0:166
##  1:  4   1st Qu.:133.0   1: 98   1st Qu.:0.000   2:138   1.0: 65   6.0: 18
##  2:147   Median :153.0           Median :0.800   3: 21   2.0: 38   7.0:115
##          Mean   :149.5           Mean   :1.052           3.0: 20
##          3rd Qu.:165.5           3rd Qu.:1.600
##          Max.   :202.0           Max.   :6.200
##  num
##  0:161
##  1:138
##
##
##
##
```

`$Data Exploration $`

`Plot proportion of disease present vs disease not present`

```
data %>%
  ggplot(aes(x = num)) +
  geom_histogram(stat = 'count', fill = "steelblue") +
  theme_bw()
```



Determine number of disease vs not disease and portions

```
data_nrows <- nrow(data)
data_nrows
```

```
## [1] 299
```

```
data_present <- sum(as.numeric(data$num) == 2)
data_present
```

```
## [1] 138
```

```
data_not_present <- data_nrows - data_present
data_not_present
```

```
## [1] 161
```

```
data_present_percent <- data_present / data_nrows
data_present_percent
```

```
## [1] 0.4615385
```

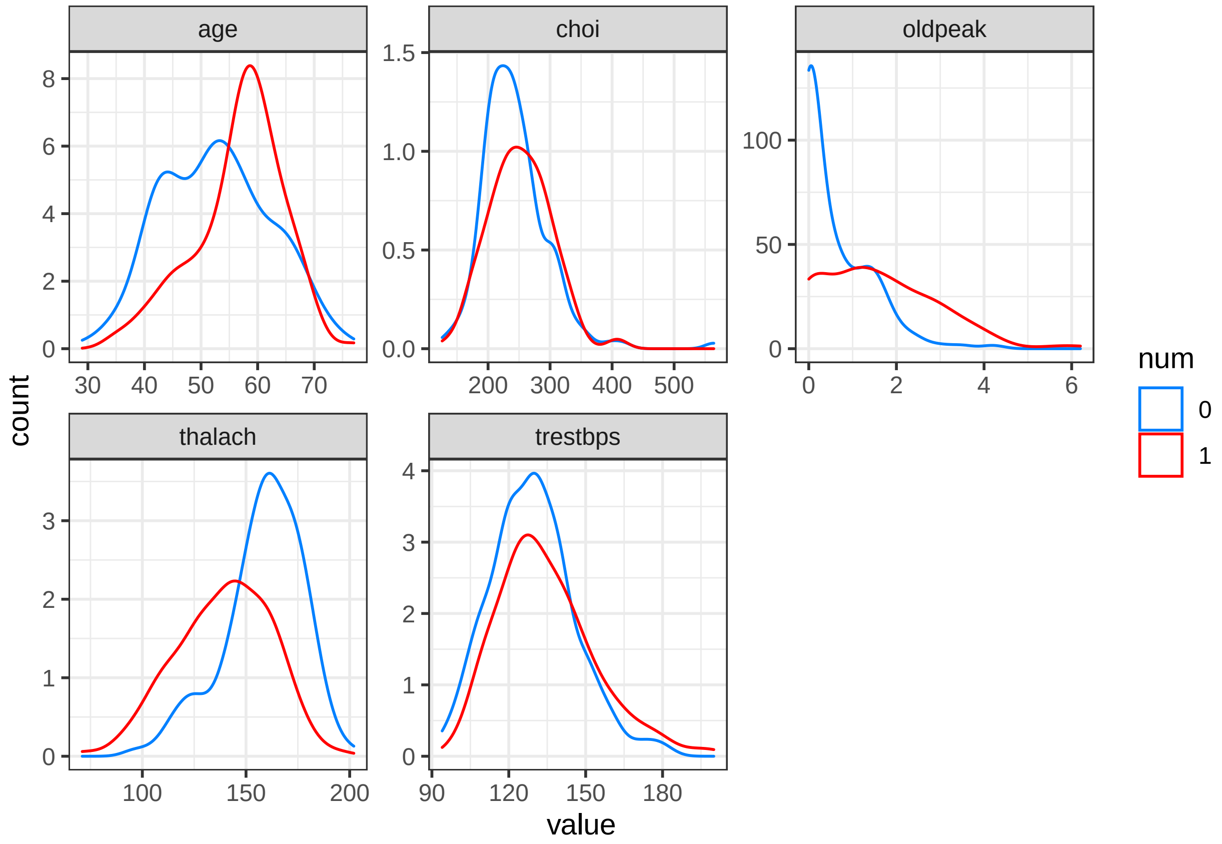
```
data_not_present_percent <- 1 - data_present_percent
data_not_present_percent
```

```
## [1] 0.5384615
```

The dimenion of the database is 299 X 14. The number of patients with heart disease is 138, 46.15% of the database. The number of patients without heart disease is 161, 53.85% of the database.

Plot a few predictors.

```r
data %>%
  gather(-sex, -cp, -fbs, -restecg, -exang, -slope, -ca, -thai, -num, key = "var", value = "value") %>%
  ggplot(aes(x = value, y = ..count.. , colour = num)) +
  scale_color_manual(values=c("#0080FF", "#FF0000"))+
  geom_density() +
  facet_wrap(~var, scales = "free",  nrow = 2) +
  theme_bw()
```



Analysis shows that as patients approach age 55 the presence of heart disease starts to exceed no presence. Cholesterol (chol) doesn't appear to have as much as an effect as advertised. The number of test subjects with heart disease is either lower or equal to test subjects without heart disease.

Create train set and test set by partitioning the data set. The train_set and test_set are used by the Random Forest algorithm. The dt_train_set and dt_test_set are used by the Decision Tree alogrithm.

```r
set.seed(123, sample.kind="Rounding")

test_index <- createDataPartition(y = data$num, times = 1, p = 0.25, list = FALSE)
train_set <- data[-test_index,]
test_set <- data[test_index,]

dt_train_set <- data[-test_index,]
dt_test_set <- data[test_index,]
```

$Decision Tree $ First the Descision Tree algorithm is evaluated.

Create a full Design Tree.

```r
dt_full <- rpart(num ~ . , data = dt_train_set, method = "class", cp = 0)
```

The complexity table is used to control the size of the tree and find the optimal tree size. Creating a full tree with a cp setting of zero means there are no restrictions and ends up creating a complex tree.

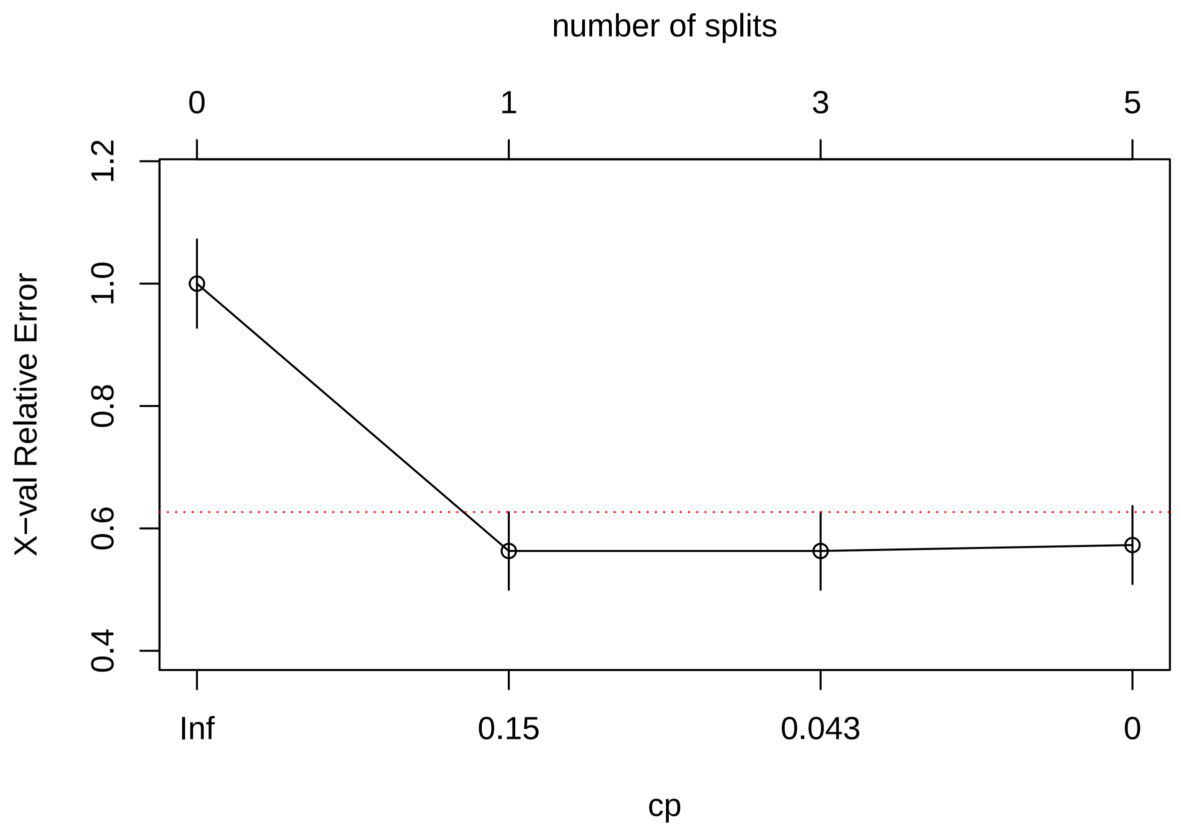
Print the complexity parameter (cp) table to help select the decision tree that minimizes misclassification error.

```r
printcp(dt_full)
```

```
##
## Classification tree:
## rpart(formula = num ~ ., data = dt_train_set, method = "class",
##     cp = 0)
##
## Variables actually used in tree construction:
## [1] ca      cp      thai    thalach
##
## Root node error: 103/223 = 0.46188
##
## n= 223
##
##         CP nsplit rel error  xerror     xstd
## 1 0.485437      0   1.00000 1.00000 0.072280
## 2 0.048544      1   0.51456 0.56311 0.063601
## 3 0.038835      3   0.41748 0.56311 0.063601
## 4 0.000000      5   0.33981 0.57282 0.063953
```

Plot the cp.

```r
plotcp(dt_full, lty = 3, col = 2, upper = "splits")
```



Find the best cp based on the lowest xerror.

```
bestcp <- dt_full$cptable[which.min(dt_full$cptable[,"xerror"]),"CP"]
bestcp
```

```
## [1] 0.04854369
```

Prune the bestcp.

```
dt_pruned <- prune(dt_full, cp = bestcp)
summary(dt_pruned)
```

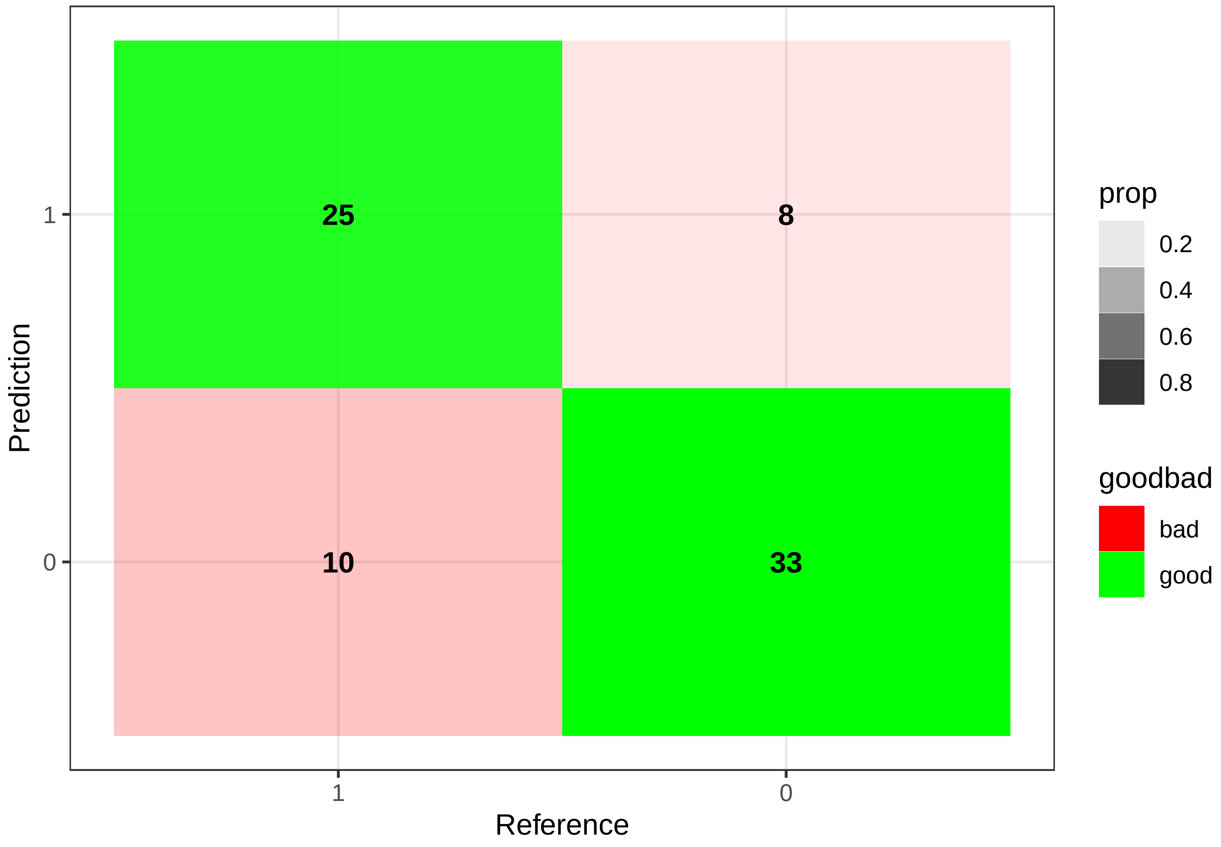
```
## Call:
## rpart(formula = num ~ ., data = dt_train_set, method = "class",
##     cp = 0)
##   n= 223
##
##           CP nsplit rel error    xerror       xstd
## 1 0.48543689      0 1.0000000 1.0000000 0.07228024
## 2 0.04854369      1 0.5145631 0.5631068 0.06360135
##
## Variable importance
##    thai thalach oldpeak      cp     sex   slope
##      45      13      12      11      10      10
##
## Node number 1: 223 observations,    complexity param=0.4854369
##   predicted class=0  expected loss=0.4618834  P(node) =1
##     class counts:   120   103
##    probabilities: 0.538 0.462
##   left son=2 (123 obs) right son=3 (100 obs)
##   Primary splits:
##       thai    splits as  LRR,       improve=30.09999, (0 missing)
##       cp      splits as  LLLR,      improve=28.27550, (0 missing)
##       ca      splits as  LRRR,      improve=24.09630, (0 missing)
##       thalach < 146.5 to the right, improve=21.38058, (0 missing)
##       oldpeak < 1.7   to the left,  improve=19.84396, (0 missing)
##   Surrogate splits:
##       thalach < 150.5 to the right, agree=0.677, adj=0.28, (0 split)
##       oldpeak < 0.75  to the left,  agree=0.673, adj=0.27, (0 split)
##       cp      splits as  LLLR,      agree=0.659, adj=0.24, (0 split)
##       sex     splits as  LR,        agree=0.650, adj=0.22, (0 split)
##       slope   splits as  LRR,       agree=0.650, adj=0.22, (0 split)
##
## Node number 2: 123 observations
##   predicted class=0  expected loss=0.2276423  P(node) =0.5515695
##     class counts:    95    28
##    probabilities: 0.772 0.228
##
## Node number 3: 100 observations
##   predicted class=1  expected loss=0.25  P(node) =0.4484305
##     class counts:    25    75
##    probabilities: 0.250 0.750
```

Predict based on the pruned decision tree.

```r
dt_predict <- predict(dt_pruned, dt_test_set, type = "class")
```

Calculate confusion matrix and plot it.

```r
# Calculate confusion matrix and plot it
table <- data.frame(confusionMatrix(dt_predict, test_set$num)$table)

plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  xlim(rev(levels(table$Reference)))
```



Calculate the classification error, subtract from 1 to get accuracy.

```
dt_accuracy <- 1 - round((table$Freq[2] + table$Freq[3]) / (table$Freq[1] + table$Freq[2] + table$Freq[
dt_accuracy
```

```
## [1] 0.763
```

$Random Forest Algorithm $ Now let's see how the Random Forest algorithm performs.

K-fold cross validation is handled by the the trControl function. cv is the tyoe of method used for resampling the dataset. number is the number of folders. The type of search is a grid of variables to try.

```
set.seed(123, sample.kind="Rounding")

trControl <- trainControl(method = "cv",
                          number = 10,
                          search = "grid")
```

Train the default Random Forest model.

```
rf_default <- train(num ~ .,
                    data = train_set,
                    method = "rf",
                    metric = "Accuracy",
                    trControl = trControl)

rf_default
```

```
## Random Forest
```

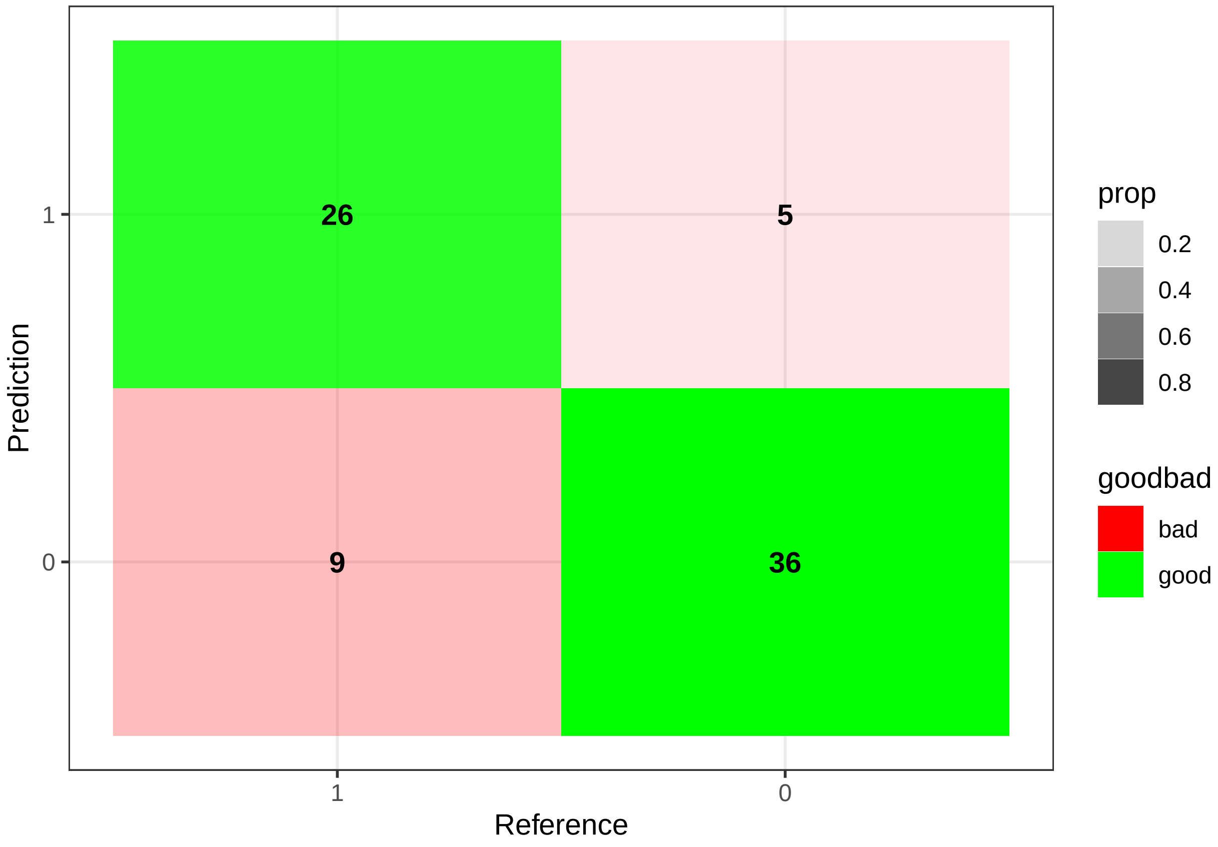
```
##
## 223 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 201, 201, 201, 201, 200, 201, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8213439  0.6368605
##   11    0.7719368  0.5369794
##   20    0.7766798  0.5461955
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Make prediction for the Random Forest with default parameters.

```
pred_default <-predict(rf_default, test_set)
```

Calculate the Confusion Matrix and plot it.

```
table <- data.frame(confusionMatrix(pred_default, test_set$num)$table)

plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  xlim(rev(levels(table$Reference)))
```



Calculate classification error to get accuracy.

```
rf_accuracy <- 1 - round((table$Freq[2] + table$Freq[3]) / (table$Freq[1] + table$Freq[2] + table$Freq[
rf_accuracy
```

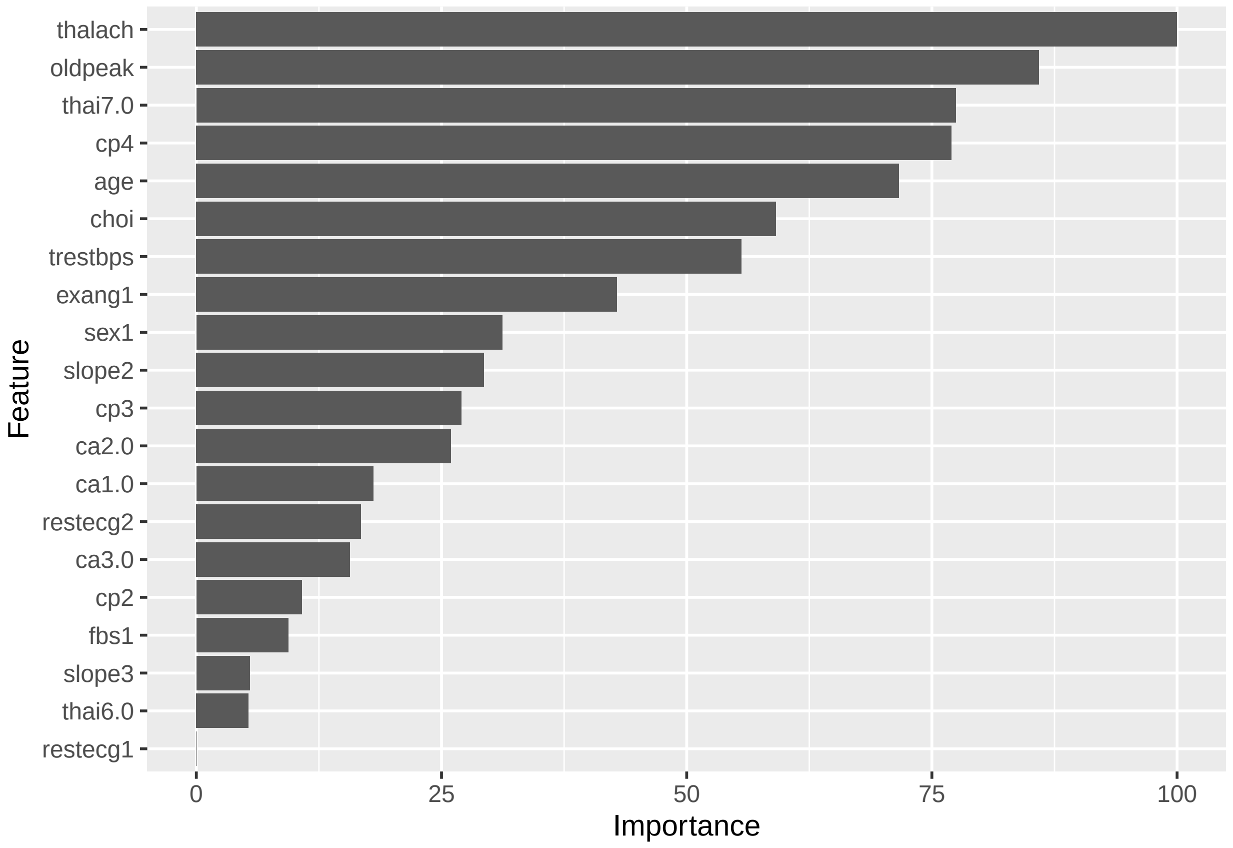
```
## [1] 0.816
```

Calculate the variable importance and plot it.

```
varImp(rf_default)
```

```
## rf variable importance
##
##          Overall
## thalach  100.000
## oldpeak   85.928
## thai7.0   77.437
## cp4       76.983
## age       71.645
## choi      59.108
## trestbps  55.577
## exang1    42.905
## sex1      31.194
## slope2    29.331
## cp3       27.043
## ca2.0     25.960
## ca1.0     18.044
## restecg2  16.784
## ca3.0     15.654
## cp2         10.747
## fbs1         9.363
## slope3       5.454
## thai6.0      5.317
## restecg1     0.000
```

```
ggplot2::ggplot(varImp(rf_default))
```



Find the best mtry.

```
tuneMtry <- expand.grid(.mtry = c(1: 10))
rf_mtry <- train(num ~ .,
                 data = train_set,
                 method = "rf",
                 metric = "Accuracy",
                 tuneGrid = tuneMtry,
                 trControl = trControl,
                 importance = TRUE,
                 nodesize = 14,
                 ntree = 300)

rf_mtry
```

```
## Random Forest
##
## 223 samples
##  13 predictor
##   2 classes: '0', '1'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 201, 200, 201, 201, 201, 200, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.8158103  0.6226722
##    2    0.8154150  0.6242924
##    3    0.7978261  0.5895336
##    4    0.7974308  0.5883713
##    5    0.7837945  0.5598265
##    6    0.7928854  0.5793295
##    7    0.7794466  0.5504426
##    8    0.8017787  0.5970459
##    9    0.7796443  0.5518354
##   10    0.7750988  0.5433145
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 1.
```

```
best_mtry <- rf_mtry$bestTune$mtry
best_mtry
```

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

Find best maxnodes.

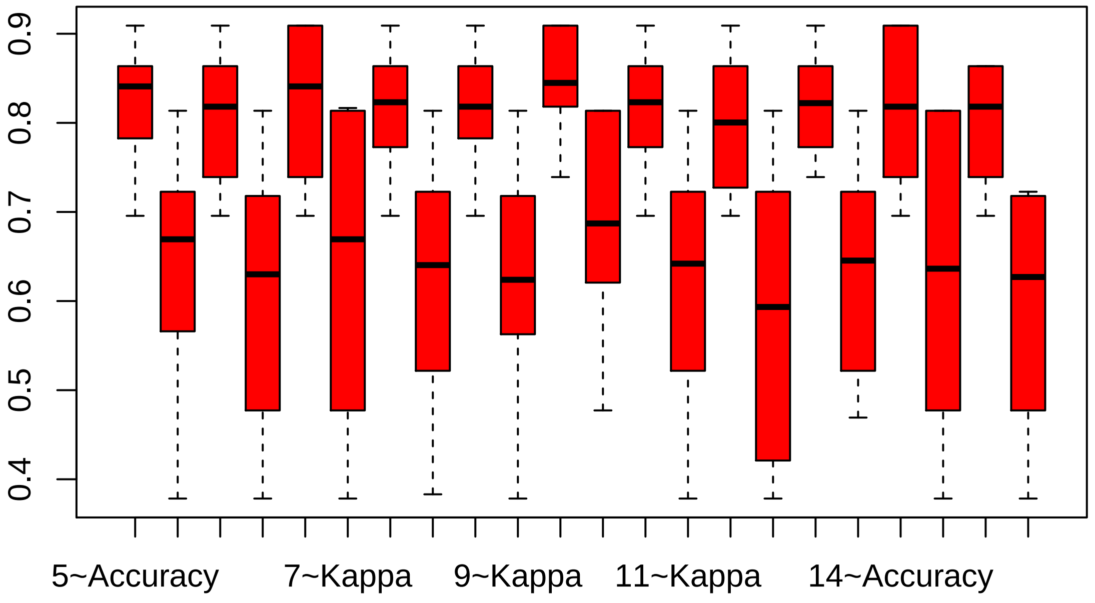
```
maxNodeList <- list()
tuneMtry <- expand.grid(.mtry = best_mtry)

for (mn in c(5: 15)) {
  set.seed(123, sample.kind="Rounding")
  rf_maxnode <- train(num ~ .,
                      data = train_set,
                      method = "rf",
                      metric = "Accuracy",
                      tuneGrid = tuneMtry,
                      trControl = trControl,
                      importance = TRUE,
                      nodesize = 14,
                      maxnodes = mn,
                      ntree = 300)
  current_iteration <- toString(mn)
  maxNodeList[[current_iteration]] <- rf_maxnode
}
results_maxnodes <- resamples(maxNodeList)
summary(results_maxnodes)
```

```
##
## Call:
## summary.resamples(object = results_maxnodes)
##
## Models: 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
## Number of resamples: 10
##
```

```
## Accuracy
##         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 5  0.6956522 0.7915020 0.8409091 0.8217391 0.8636364 0.9090909    0
## 6  0.6956522 0.7475296 0.8181818 0.8083004 0.8636364 0.9090909    0
## 7  0.6956522 0.7475296 0.8409091 0.8264822 0.9090909 0.9090909    0
## 8  0.6956522 0.7727273 0.8231225 0.8126482 0.8636364 0.9090909    0
## 9  0.6956522 0.7826087 0.8181818 0.8169960 0.8636364 0.9090909    0
## 10 0.7391304 0.8181818 0.8448617 0.8438735 0.8977273 0.9090909    0
## 11 0.6956522 0.7727273 0.8231225 0.8083004 0.8636364 0.9090909    0
## 12 0.6956522 0.7386364 0.8003953 0.8037549 0.8636364 0.9090909    0
## 13 0.7391304 0.7840909 0.8221344 0.8213439 0.8636364 0.9090909    0
## 14 0.6956522 0.7475296 0.8181818 0.8175889 0.9090909 0.9090909    0
## 15 0.6956522 0.7475296 0.8181818 0.8037549 0.8636364 0.8636364    0
##
## Kappa
##         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 5  0.3783784 0.5797007 0.6693192 0.6358496 0.7215040 0.8135593    0
## 6  0.3783784 0.4883893 0.6300170 0.6078112 0.7179487 0.8135593    0
## 7  0.3783784 0.4883893 0.6693192 0.6450646 0.8135593 0.8166667    0
## 8  0.3831418 0.5276680 0.6403432 0.6186463 0.7226891 0.8135593    0
## 9  0.3783784 0.5635627 0.6239041 0.6267822 0.7179487 0.8135593    0
## 10 0.4772727 0.6253533 0.6871021 0.6834265 0.7908418 0.8135593    0
## 11 0.3783784 0.5295157 0.6419932 0.6101668 0.7215040 0.8135593    0
## 12 0.3783784 0.4540009 0.5933637 0.5989902 0.7215040 0.8135593    0
## 13 0.4692308 0.5496377 0.6454297 0.6365331 0.7215040 0.8135593    0
## 14 0.3783784 0.4883893 0.6363388 0.6292138 0.8135593 0.8135593    0
## 15 0.3783784 0.4961659 0.6270115 0.6017078 0.7179487 0.7226891    0
```

Plot best maxnodes.

```
boxplot(results_maxnodes$values[1:10,2:23], col = "red")
```



Find the best maxtrees.

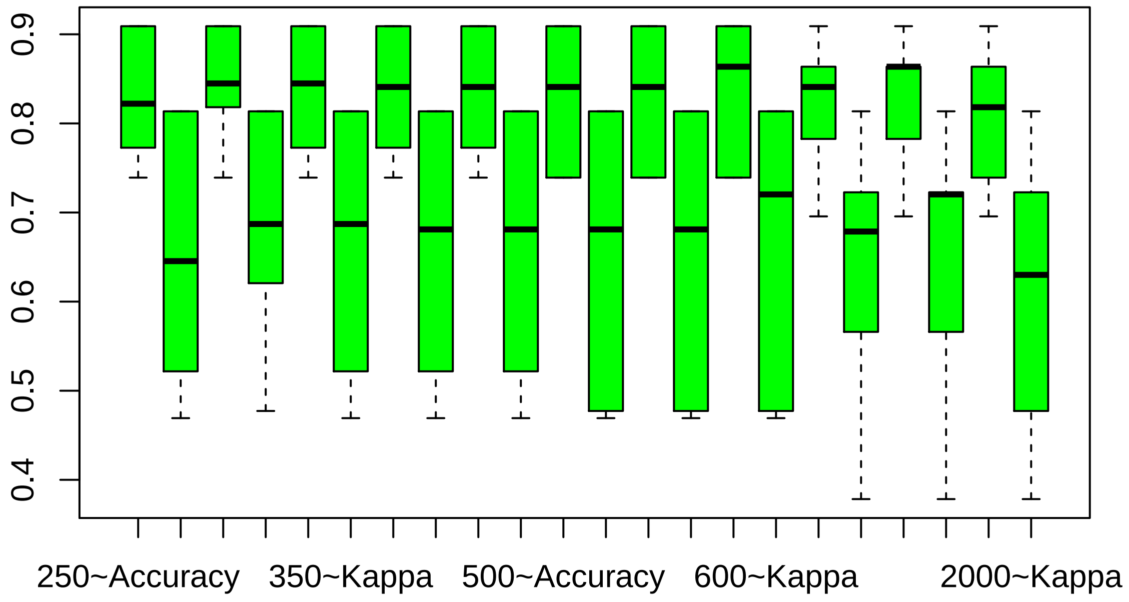
```
maxTreeList <- list()
tuneMtry <- expand.grid(.mtry = best_mtry)
```

```r
for (nt in c(250, 300, 350, 400, 450, 500, 550, 600, 800, 1000, 2000)) {
  set.seed(123)
  rf_maxtrees <- train(num ~ .,
                       data = train_set,
                       method = "rf",
                       metric = "Accuracy",
                       tuneGrid = tuneMtry,
                       trControl = trControl,
                       importance = TRUE,
                       nodesize = 14,
                       maxnodes = 10,
                       ntree = nt)
  key <- toString(nt)
  maxTreeList[[key]] <- rf_maxtrees
}
results_tree <- resamples(maxTreeList)
summary(results_tree)
```

```
##
## Call:
## summary.resamples(object = results_tree)
##
## Models: 250, 300, 350, 400, 450, 500, 550, 600, 800, 1000, 2000
## Number of resamples: 10
##
## Accuracy
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 250  0.7391304 0.7840909 0.8221344 0.8304348 0.8977273 0.9090909    0
## 300  0.7391304 0.8181818 0.8448617 0.8438735 0.8977273 0.9090909    0
## 350  0.7391304 0.7840909 0.8448617 0.8349802 0.8977273 0.9090909    0
## 400  0.7391304 0.7751976 0.8409091 0.8306324 0.8977273 0.9090909    0
## 450  0.7391304 0.7751976 0.8409091 0.8306324 0.8977273 0.9090909    0
## 500  0.7391304 0.7475296 0.8409091 0.8262846 0.8977273 0.9090909    0
## 550  0.7391304 0.7475296 0.8409091 0.8262846 0.8977273 0.9090909    0
## 600  0.7391304 0.7588933 0.8636364 0.8353755 0.8977273 0.9090909    0
## 800  0.6956522 0.7915020 0.8409091 0.8306324 0.8636364 0.9090909    0
## 1000 0.6956522 0.7915020 0.8636364 0.8351779 0.8636364 0.9090909    0
## 2000 0.6956522 0.7475296 0.8181818 0.8128458 0.8636364 0.9090909    0
##
## Kappa
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 250  0.4692308 0.5496377 0.6454297 0.6555802 0.7908418 0.8135593    0
## 300  0.4772727 0.6253533 0.6871021 0.6834265 0.7908418 0.8135593    0
## 350  0.4692308 0.5511404 0.6871021 0.6645158 0.7908418 0.8135593    0
## 400  0.4692308 0.5328138 0.6810167 0.6559681 0.7908418 0.8135593    0
## 450  0.4692308 0.5328138 0.6810167 0.6559681 0.7908418 0.8135593    0
## 500  0.4692308 0.4883893 0.6810167 0.6470916 0.7908418 0.8135593    0
## 550  0.4692308 0.4883893 0.6810167 0.6470916 0.7908418 0.8135593    0
## 600  0.4692308 0.5177906 0.7203189 0.6667125 0.7908418 0.8135593    0
## 800  0.3783784 0.5828616 0.6786465 0.6570277 0.7226891 0.8135593    0
## 1000 0.3783784 0.5843644 0.7203189 0.6659633 0.7226891 0.8135593    0
## 2000 0.3783784 0.4943182 0.6300170 0.6206919 0.7226891 0.8135593    0
```

Plot best maxtrees.

```
boxplot(results_tree$values[1:10,2:23], col = "green")
```



Now that we have the optimized tuning parameters, generate final fit.

```
set.seed(123)

tuneMtry <- expand.grid(.mtry = best_mtry)

rf_fit <-train(
              num ~ .,
              data = train_set,
              method = "rf",
              metric = "Accuracy",
              trControl = trControl,
              tuneGrid = tuneMtry,
              importance = TRUE,
              nodesize = 14,
              maxnodes = 10,
              ntree = 800)

rf_fit
```

```
## Random Forest
##
## 223 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 201, 201, 201, 201, 200, 201, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8306324  0.6570277
```

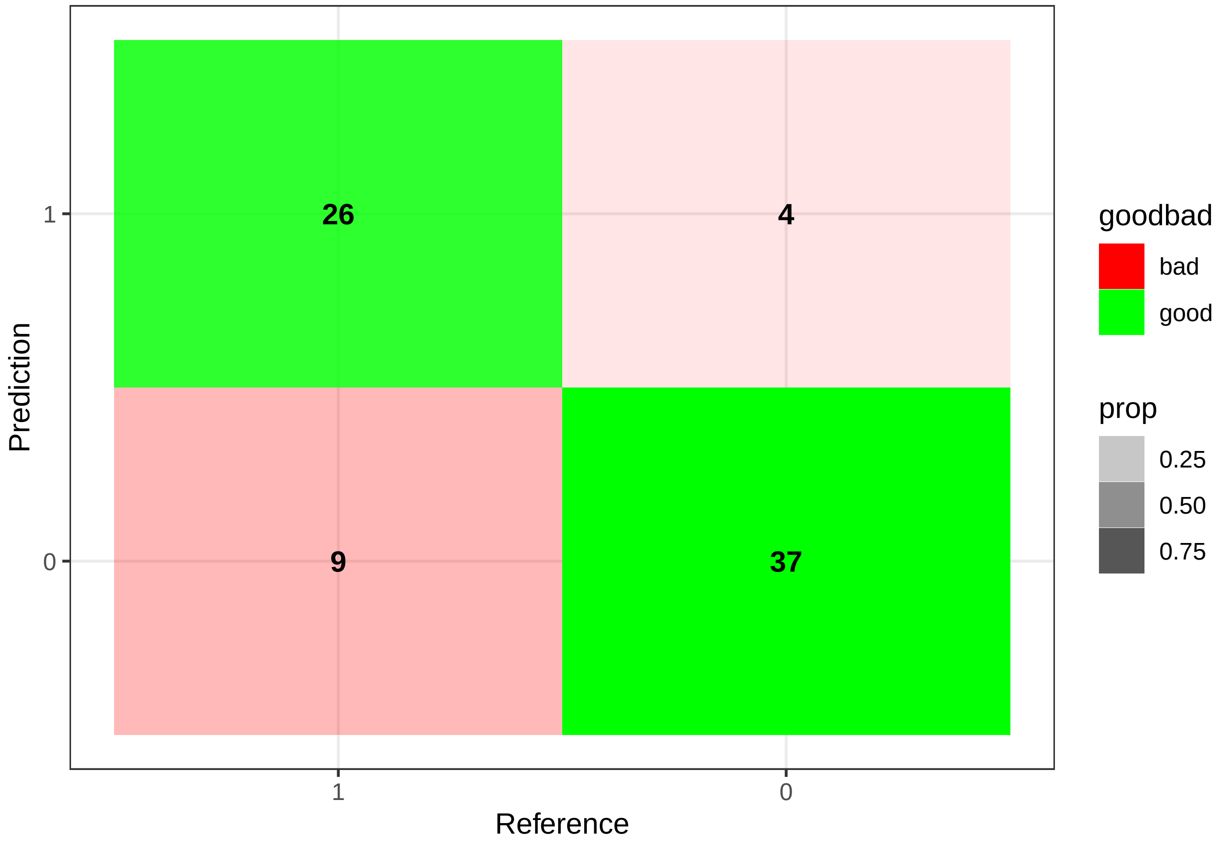
```
##
## Tuning parameter 'mtry' was held constant at a value of 1
```

Make a prediction with the final fit.

```
prediction <-predict(rf_fit, test_set)
```

Plot Confusion Matrix.

```
table <- data.frame(confusionMatrix(prediction, test_set$num)$table)

plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good", "bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "green", bad = "red")) +
  theme_bw() +
  xlim(rev(levels(table$Reference)))
```



Calculate the classification error, subtract from 1 to get accuracy. Sum up the false positives and false negatives and divide by the totalof the confusion matrix.

```
rf_accuracy <- 1 - round((table$Freq[2] + table$Freq[3]) / (table$Freq[1] + table$Freq[2] + table$Freq[
rf_accuracy
```

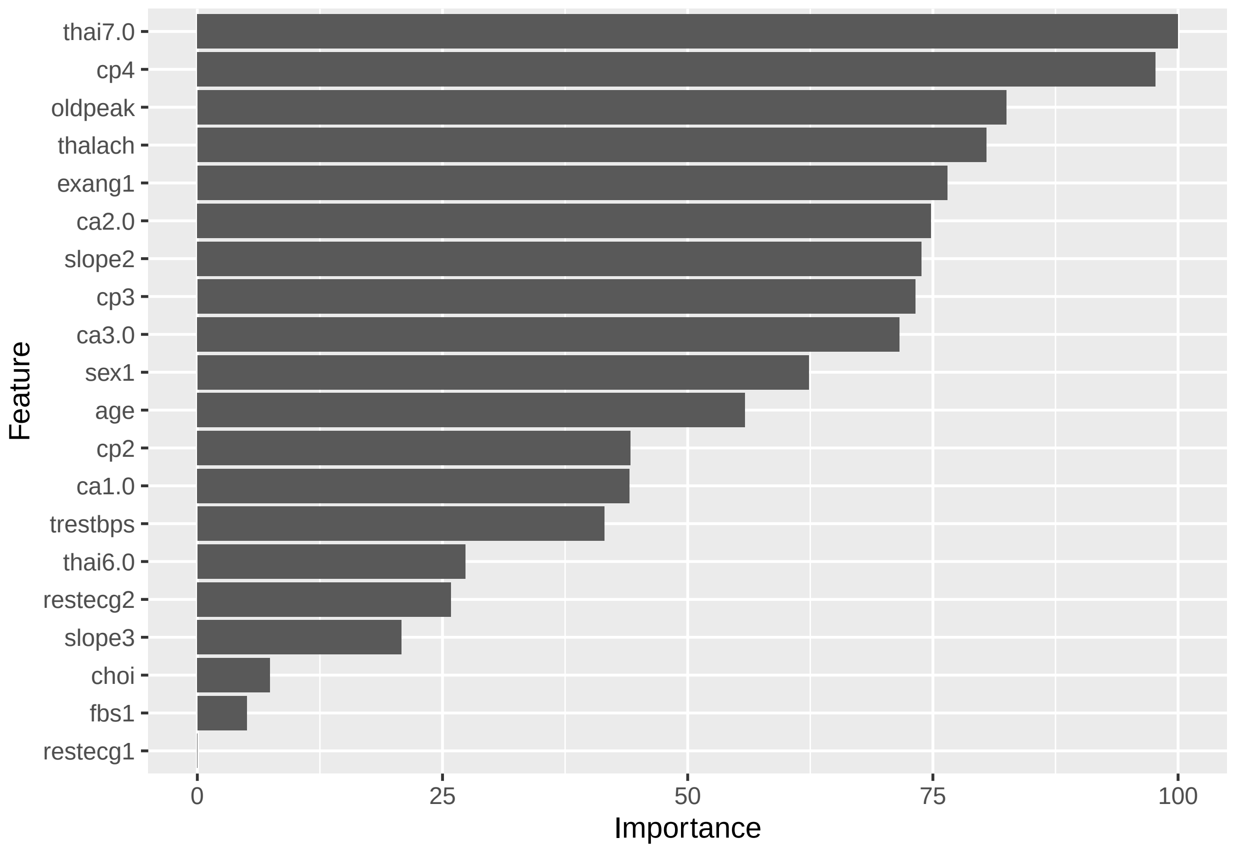
```
## [1] 0.829
```

Determine variable importance and plot.

```
varImp(rf_fit)
```

```
## rf variable importance
##
##          Importance
## thai7.0     100.000
## cp4          97.708
## oldpeak      82.482
## thalach      80.444
## exang1       76.470
## ca2.0        74.818
## slope2       73.831
## cp3          73.198
## ca3.0        71.603
## sex1         62.334
## age          55.849
## cp2          44.157
## ca1.0        44.068
## trestbps     41.496
## thai6.0      27.324
## restecg2     25.872
## slope3       20.818
## choi          7.411
## fbs1          5.033
## restecg1      0.000
```

```
ggplot2::ggplot(varImp(rf_fit))
```



Results

The Random Forest algorithm produced better results than the Design Tree algorithm.

Acurracy: Data Tree model with default parameters 0.763; Random Forest model with default parameters 0.816; Random Forest model with optimized tuning parameters 0.829.

The complexity paramter (cp) table showed that the best cp to prune was the second one, with a cp of 0.048544 and the lowest xerror of 0.56311. Actually a third cp had the same xerror value but 2 was chosen first.

Three tuning paramters were used to generate an optimized fit for Random Forest: mtry, maxnodes and maxtrees. The optmized tuning parameters were: a) mtryof1; b) maxnodesof11; c) maxtreesof350.

The tuning parameter mtry is the number of randomly chosen variables at each split in the tree. The tuning parameter maxnodes is the number of terminal nodes. The tuning parameter maxtree is the number of trees to build.

The mtry tuning variable of 1 was chosen out the range 1 to 10.
The default variable is normally

$$\sqrt{number-of-variables} \tag{1}$$

Lower mtry values tend to lead to less correlated trees which increases stability. Moderate values have more of an effect when mtry values are low. However, lower mtry values can perform worse since less optimal values are used. Higher values were attempted but the accuracy was always worse.

The optimal maxtrees found was 350, a little lower than expected. The maxtrees don't appear as tunable but from the literature it is recommended they are high. Values from

500 to 2000 were attempted as well as low as 50 and didn't appear to have very much effect. Intuitively one would think that a higher number of trees would lead to more stability and lead to better results.

The optimal maxnodes found was 11. This is the maximum number of terminal nodes. Changes to this parameter did not seem to have much effect.

The nodesize that was used was 14, the number of variables. The typical default is 1 for classification but performance improved with 14.

The variable importance shows the importance of variables in the prediction rule. The 4 most important variables for the Random Forest default approach are thaalach thai7.0 (100%), oldpeak (86%), thai7.0 (77%), and cp4 (77%). The least important variables for the Random Forest default approach are restecgi (0%), thai6.0 (5%), slope (5%), and fbsi (9%).

The variable importance changed from the the Random Forest default approach to the optimized approach. The 4 most important variables for the Random Forest default approach are thai7.0 (100%), cp4 (98%), oldpeak (82%), and thaalach (80%). The least important variables for the Random Forest default approach are restecgi (0%), fbsi (5%), chol (7%), and slope3 (20%).

The higher number of trees the more stable the variable importance. Increasing mtry will increase the magnitude of variable of importance and affects splitting.

The database was partitioned into 75% training and 25% test. A few other partitions were tried: 0.2, 0.3, 0.4. The 0.25 partition index produced the most accurate results out of the partitions attempted.

Conclusion

Decision Tree is a weak learner and suffers from overfitting. Overfitting means that it can get overtrained with the training set and when finally evaluated with the test set, it doesn't perform very well.

The Random Forest uses many decision trees (hundreds or thousands) to create an ensemble of trees and doesn't overfit too often. This leads to a generalization of a goup of weak learning decision trees to create a stronger learning algorithm. This has been shown by the accuracy results above.

The train function was used for training Random Forests. The randomForest function based on Breiman's algorithm produces better results and would be interesting future work.

Ensemble learning algorithms like Gradient Boosting Model (GBM) and Extreme Gradient Boosting (XGB) model would also be good candidates for future evaluation. GBM caused R Studio to crash and would like to look into why it was occurring. XGB produced good results and more time is needed analyze it further.