P8106_HW4

Naomi Simon-Kumar ns3782

04/11/2025

Contents

Loading libraries

```
# Load libraries
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.2.0 --
## v broom        1.0.7     v rsample      1.2.1
## v dials        1.3.0     v tune         1.2.1
## v infer        1.0.7     v workflows    1.1.4
## v modeldata    1.4.0     v workflowsets 1.1.0
## v parsnip      1.2.1     v yardstick    1.3.2
## v recipes      1.1.0
```

```
## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Dig deeper into tidy modeling with R at https://www.tmwr.org
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following objects are masked from 'package:yardstick':
##
##     precision, recall, sensitivity, specificity
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(ggplot2)
library(rpart)
```

```
##
## Attaching package: 'rpart'
##
## The following object is masked from 'package:dials':
##
##     prune
```

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

```
## Warning: package 'ranger' was built under R version 4.4.3
```

Part 1: Tree Based Models using College Data

Partition into training and testing set

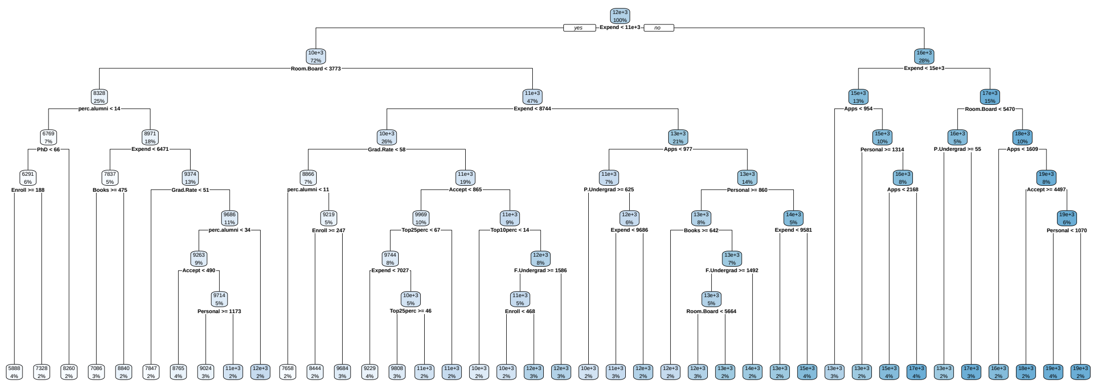
```
# Read in dataset
college <- read.csv("College.csv")

# Remove NAs
college <- na.omit(college)

# Set seed for reproducibility
```

```r
set.seed(299)

# Split data into training and testing data
data_split_college <- initial_split(college, prop = 0.8)

# Extract the training and test data, removing college ID column
training_data_college <- training(data_split_college) %>% select(-College)
testing_data_college <- testing(data_split_college) %>% select(-College)
```

a) Build a regression tree on training data

In order to implement the CART approach implementing recursive partitioning and pruning, I first fit a regression tree using cp=0 (complexity parameter). This parameter controls the complexity pruning in the CART algorithm, i.e., how splits are undertaken. Setting cp=0 was a safe choice to ensure that the tree was sufficiently large, allowing all potential splits are considered. I also produced a plot of this tree.

```r
# Set seed for reproducibility
set.seed(299)

# Fit initial tree:
initial.tree.fit <- rpart(Outstate ~ ., data = training_data_college,
                          control = rpart.control(cp = 0))

# Tree plot
rpart.plot(initial.tree.fit)
```

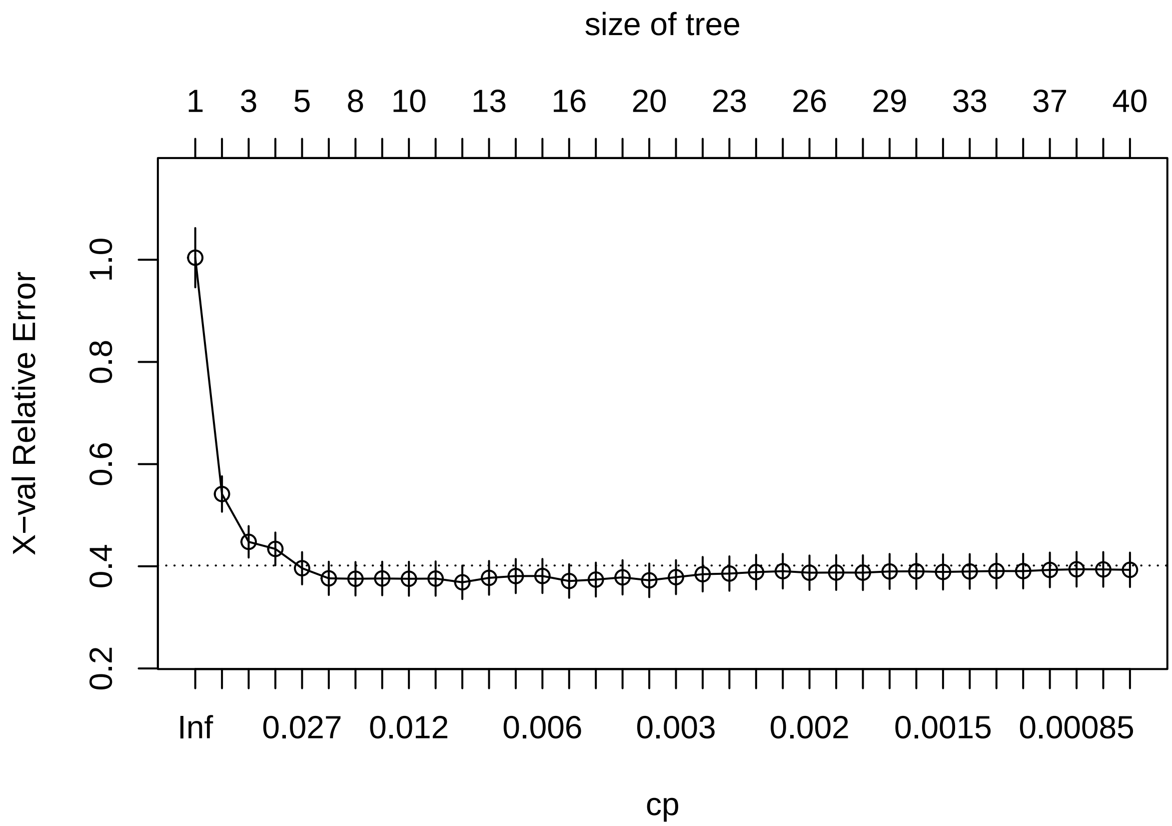


```r
# Print and plot the cp table
printcp(initial.tree.fit)
```

```
##
## Regression tree:
## rpart(formula = Outstate ~ ., data = training_data_college, control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
##  [1] Accept      Apps        Books       Enroll      Expend      F.Undergrad
##  [7] Grad.Rate   P.Undergrad perc.alumni Personal    PhD         Room.Board
## [13] Top10perc   Top25perc
##
## Root node error: 6222135701/452 = 13765787
##
## n= 452
##
##            CP nsplit rel error  xerror     xstd
## 1  0.49965968      0   1.00000 1.00404 0.058031
## 2  0.10266905      1   0.50034 0.54145 0.034753
## 3  0.05071124      2   0.39767 0.44785 0.030837
## 4  0.04021488      3   0.34696 0.43401 0.032119
## 5  0.01820952      4   0.30675 0.39616 0.031612
## 6  0.01319074      5   0.28854 0.37639 0.032586
## 7  0.01255106      7   0.26215 0.37556 0.032802
## 8  0.01194445      8   0.24960 0.37611 0.032884
```

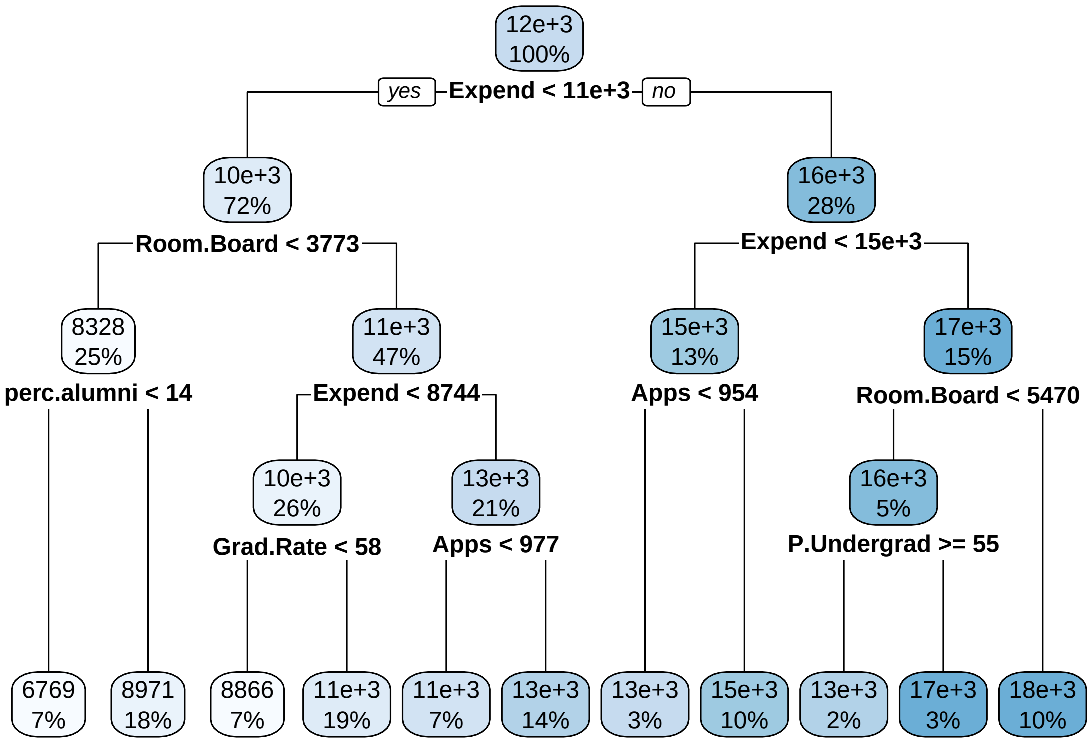
```
## 9  0.01165684      9   0.23766 0.37553 0.033331
## 10 0.00942611     10   0.22600 0.37590 0.033641
## 11 0.00770392     11   0.21658 0.36867 0.032816
## 12 0.00672841     12   0.20887 0.37730 0.033123
## 13 0.00619338     13   0.20214 0.38085 0.033378
## 14 0.00587852     14   0.19595 0.38098 0.033357
## 15 0.00538113     15   0.19007 0.37111 0.032927
## 16 0.00531018     17   0.17931 0.37395 0.033274
## 17 0.00377286     18   0.17400 0.37834 0.033610
## 18 0.00353985     19   0.17023 0.37250 0.032941
## 19 0.00254368     20   0.16669 0.37870 0.033291
## 20 0.00246805     21   0.16414 0.38447 0.033757
## 21 0.00212300     22   0.16167 0.38576 0.033736
## 22 0.00210237     23   0.15955 0.38858 0.033838
## 23 0.00207706     24   0.15745 0.39037 0.033877
## 24 0.00199280     25   0.15537 0.38740 0.033677
## 25 0.00188271     26   0.15338 0.38777 0.034094
## 26 0.00187160     27   0.15150 0.38754 0.034083
## 27 0.00179477     28   0.14963 0.38986 0.034280
## 28 0.00167899     30   0.14604 0.39016 0.034537
## 29 0.00139108     31   0.14436 0.38893 0.034479
## 30 0.00120513     32   0.14297 0.38988 0.033974
## 31 0.00119267     33   0.14176 0.39068 0.033937
## 32 0.00110541     34   0.14057 0.39051 0.033940
## 33 0.00099926     36   0.13836 0.39288 0.033968
## 34 0.00073074     37   0.13736 0.39417 0.033983
## 35 0.00031851     38   0.13663 0.39388 0.033991
## 36 0.00000000     39   0.13631 0.39305 0.033715
```

```r
plotcp(initial.tree.fit)
```



To explore the impact of adjusting the complexity parameter, I also fit a second model setting cp=0.01. This model had fewer splits by comparison. When plotted, this tree was noticeably smaller than the previous tree.

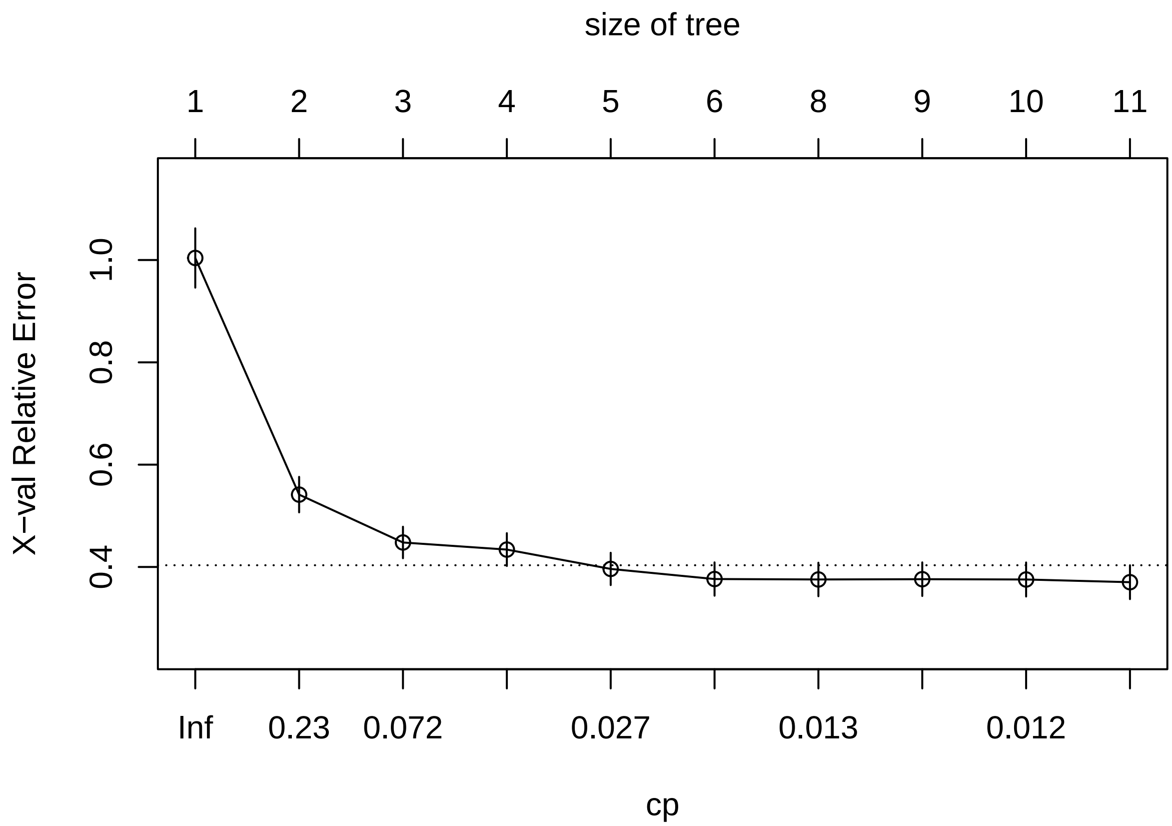
```r
# Set seed for reproducibility
set.seed(299)

# Fit another tree:
tree.fit.2 <- rpart(Outstate ~ ., data = training_data_college,
                    control = rpart.control(cp = 0.01))

# Tree plot
rpart.plot(tree.fit.2)
```




```r
# Print and plot the cp table
printcp(tree.fit.2)
```

```
##
## Regression tree:
## rpart(formula = Outstate ~ ., data = training_data_college, control = rpart.control(cp = 0.01))
##
## Variables actually used in tree construction:
## [1] Apps        Expend      Grad.Rate   P.Undergrad perc.alumni Room.Board
##
## Root node error: 6222135701/452 = 13765787
##
## n= 452
##
##          CP nsplit rel error  xerror     xstd
## 1  0.499660      0   1.00000 1.00404 0.058031
## 2  0.102669      1   0.50034 0.54145 0.034753
## 3  0.050711      2   0.39767 0.44785 0.030837
## 4  0.040215      3   0.34696 0.43401 0.032119
## 5  0.018210      4   0.30675 0.39616 0.031612
## 6  0.013191      5   0.28854 0.37639 0.032586
## 7  0.012551      7   0.26215 0.37556 0.032802
## 8  0.011944      8   0.24960 0.37611 0.032884
## 9  0.011657      9   0.23766 0.37553 0.033331
## 10 0.010000     10   0.22600 0.37024 0.033137
```

```r
plotcp(tree.fit.2)
```



Importantly, setting cp=0 is the preferred choice, as it allowed for a large enough tree to be grown for the cost complexity table. In the cp = 0.01 model, the smallest xerror (scaled cross-validation error) was 0.37369, whereas the cp = 0 model achieved a slightly lower minimum xerror of 0.36867. This shows us that a fully grown tree in this case is better suited for selecting an optimal complexity parameter based on cross-validation.

The optimal tree selected from the cp = 0 model has a **complexity parameter of 0.00770392**, with **11 splits** (i.e., 12 terminal nodes). This was the model that minimised scaled cross-validation error (xerror = 0.36867). Therefore, this was chosen as the final pruned tree.

b) Perform random forest on training data

I decided to explore two tuning grids to find the optimal random forest model using cross-validation RMSE. The mtry parameter controls the number of predictors randomly selected at each split in the forest. I decided to tune mtry over the full range of possible values, from 1 to 16 (i.e., the full number of predictors in the college dataset).

For the first grid (min.node.size = 1:7), the best model had mtry = 7 and min.node.size = 5.

```r
# Set seed for reproducibility
set.seed(299)

# Set cross-validation
ctrl <- trainControl(method = "cv")
```

```r
# Define grid for tuning mtry and min.node.size
rf.grid <- expand.grid(
  mtry = 1:16, # max no. of predictors
  splitrule = "variance",
  min.node.size = c(1:7)
)

# Fit random forest using ranger via caret
rf.fit <- train(
  Outstate ~ .,
  data = training_data_college,
  method = "ranger",
  tuneGrid = rf.grid,
  trControl = ctrl
)
```

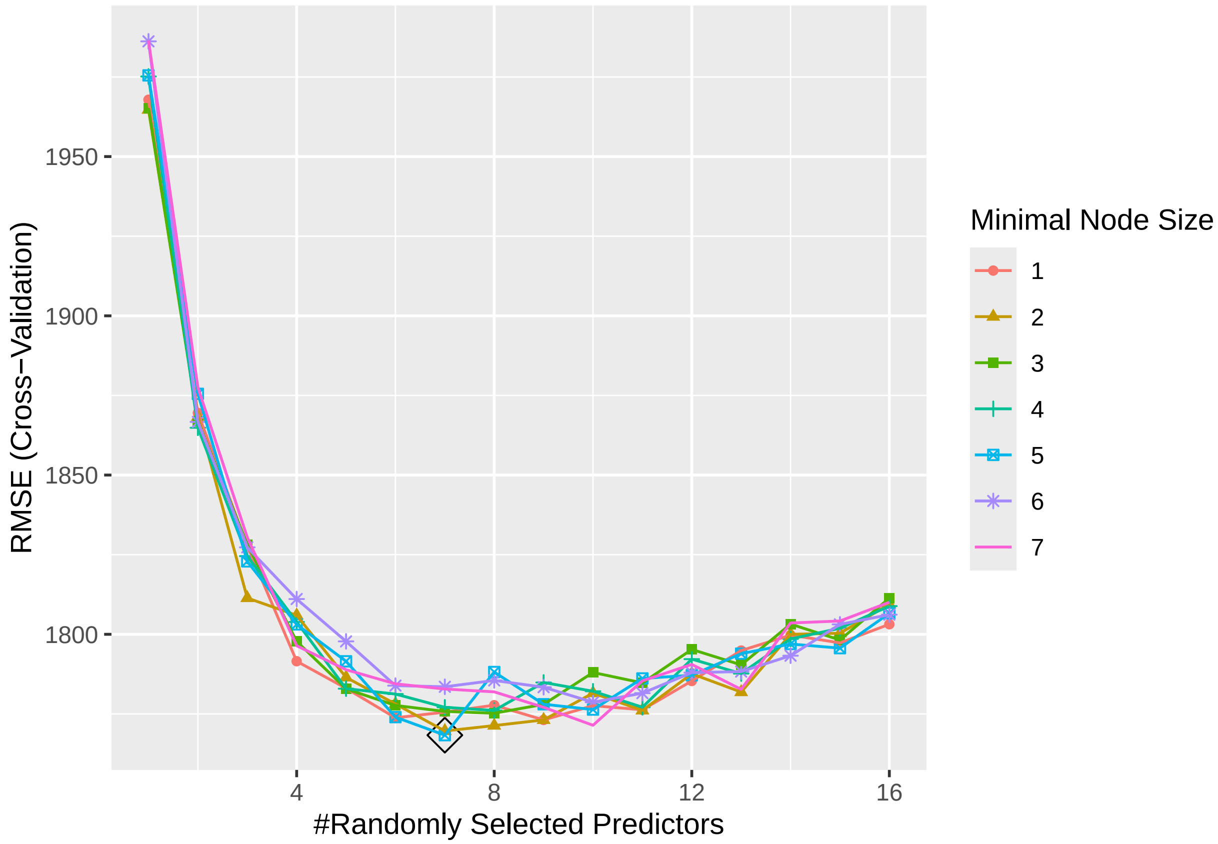
```
## Growing trees.. Progress: 54%. Estimated remaining time: 1 minute, 44 seconds.
```

```r
# Obtain optimal tuning parameters from cross-validation
rf.fit$bestTune # mtry = 7, min.node.size = 5
```

```
##    mtry splitrule min.node.size
## 47    7  variance             5
```

```r
ggplot(rf.fit, highlight = TRUE)
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because more
## than 6 becomes difficult to discriminate
## i you have requested 7 values. Consider specifying shapes manually if you need
##   that many have them.

## Warning: Removed 16 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



I then decided to try an expanded grid range for min.node.size, extending it to 1:10 to check for any better performing values beyond the original range.

```
# Set seed for reproducibility
set.seed(299)

# Define another grid for tuning mtry and min.node.size
rf.grid2 <- expand.grid(
  mtry = 1:16, # max no. of predictors
  splitrule = "variance",
  min.node.size = c(1:10)
)

# Fit random forest using ranger via caret
rf.fit2 <- train(
  Outstate ~ .,
  data = training_data_college,
  method = "ranger",
  tuneGrid = rf.grid2,
  trControl = ctrl
)
```

```
## Growing trees.. Progress: 48%. Estimated remaining time: 12 minutes, 19 seconds.
```

```r
# Optimal parameters
rf.fit2$bestTune  # mtry = 9, min.node.size = 3 (row 83)
```

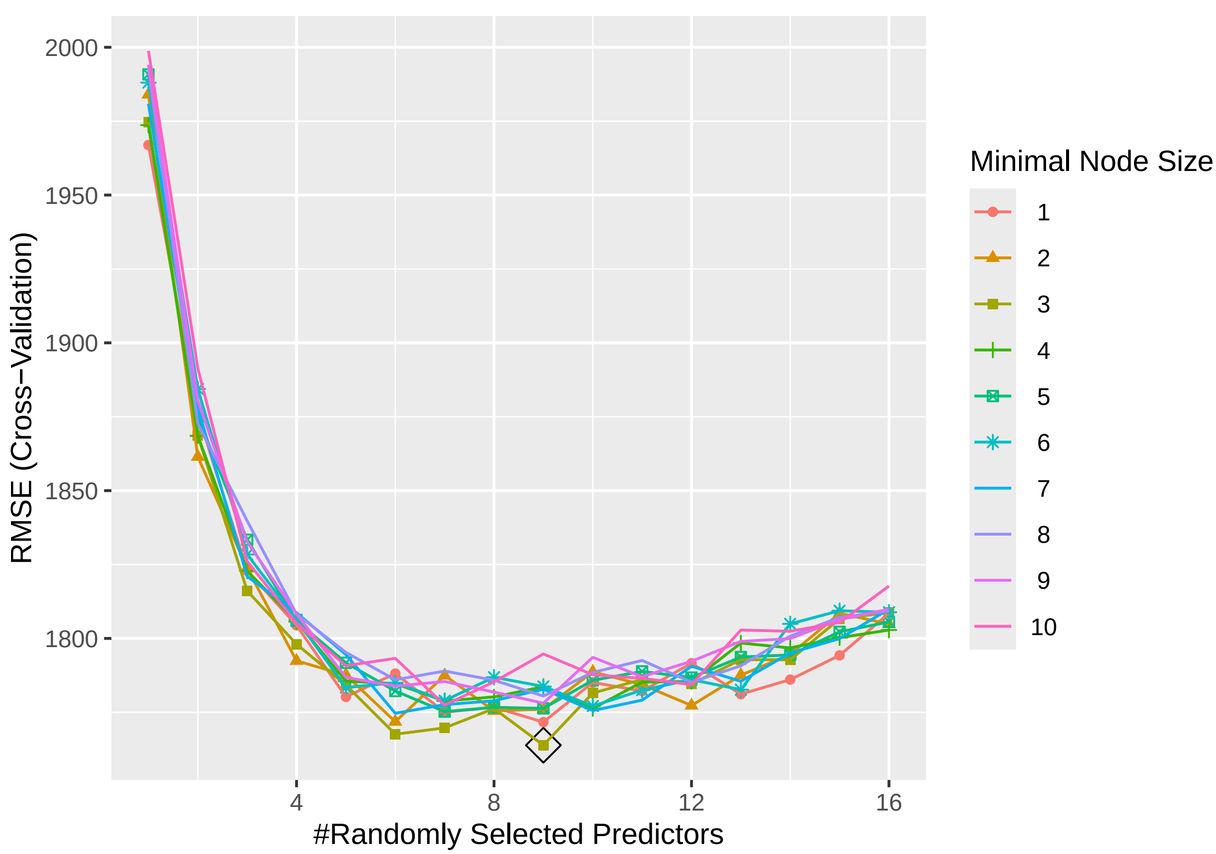
```
##    mtry splitrule min.node.size
## 83    9  variance             3
```

```r
rf.fit2$results[83, ] # pulling the lowest RMSE: 1763.865
```

```
##    mtry splitrule min.node.size     RMSE Rsquared     MAE   RMSESD RsquaredSD
## 83    9  variance             3 1763.865 0.774242 1332.22 307.1101 0.08056556
##       MAESD
## 83 176.1794
```

```r
# Plot performance for tuning grid values
ggplot(rf.fit2, highlight = TRUE)
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because more
## than 6 becomes difficult to discriminate
## i you have requested 10 values. Consider specifying shapes manually if you need
##   that many have them.

## Warning: Removed 64 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



From this grid, **the optimal tuning parameters were mtry = 9 and min.node.size = 3**, selected based on the lowest cross validation RMSE (1763.865). The corresponding plot shows the lowest RMSE.

Next, reporting the variable importance and the test error for this selected model (rf.fit2):

c) Perform boosting on training data