Data Science II Homework 3

Camille Okonkwo

Contents

```r
library(tidymodels)
library(caret)
library(rpart)
library(rpart.plot)
library(party)
library(partykit)
library(gbm)
library(ranger)
library(pROC)
set.seed(2)
```

Question 1

In this exercise, we will build tree-based models using the `College` data. The response variable is the out-of-state tuition (Outstate). Partition the data set into two parts: training data (80%) and test data (20%)

```
college = read_csv("data/College.csv") |>
  drop_na() |>
  select(-College)

set.seed(2)

# create a random split of 80% training and 20% test data
data_split = initial_split(data = college, prop = 0.8)

# partitioned datasets
training_data = training(data_split)
testing_data = testing(data_split)

head(training_data)
```

```
## # A tibble: 6 x 17
##    Apps Accept Enroll Top10perc Top25perc F.Undergrad P.Undergrad Outstate
##   <dbl>  <dbl>  <dbl>     <dbl>     <dbl>       <dbl>       <dbl>    <dbl>
## 1  1380    768    263        57        82        1000         105    19300
## 2   434    321    141        28        53         624         269    10950
## 3  2013   1053    212        33        61         912         158     5150
## 4  2324   1319    370        52        81        1686          35    16560
## 5  1709   1385    634        36        72        2281          50    14125
## 6   427    385    143        18        38         581         533    12700
## # i 9 more variables: Room.Board <dbl>, Books <dbl>, Personal <dbl>, PhD <dbl>,
## #   Terminal <dbl>, S.F.Ratio <dbl>, perc.alumni <dbl>, Expend <dbl>,
## #   Grad.Rate <dbl>
```

```
head(testing_data)
```

```
## # A tibble: 6 x 17
##    Apps Accept Enroll Top10perc Top25perc F.Undergrad P.Undergrad Outstate
##   <dbl>  <dbl>  <dbl>     <dbl>     <dbl>       <dbl>       <dbl>    <dbl>
## 1  2186   1924    512        16        29        2683        1227    12280
## 2  1428   1097    336        22        50        1036          99    11250
## 3   193    146     55        16        44         249         869     7560
## 4   582    498    172        21        44         799          78    10468
## 5  1732   1425    472        37        75        1830         110    16548
## 6   494    313    157        23        46        1317        1235     8352
## # i 9 more variables: Room.Board <dbl>, Books <dbl>, Personal <dbl>, PhD <dbl>,
## #   Terminal <dbl>, S.F.Ratio <dbl>, perc.alumni <dbl>, Expend <dbl>,
## #   Grad.Rate <dbl>
```

```
# training data
x = model.matrix(Outstate ~ ., training_data)[, -1] # matrix of predictors
head(x)
```

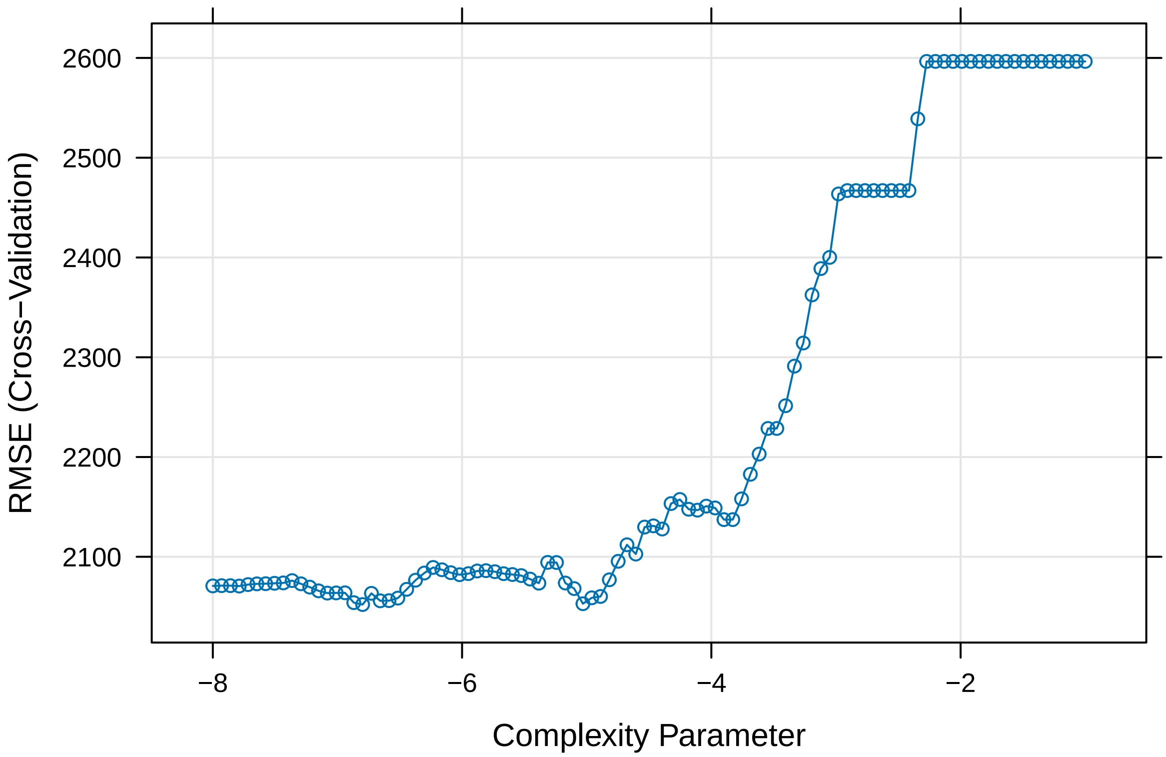
```
##   Apps Accept Enroll Top10perc Top25perc F.Undergrad P.Undergrad Room.Board
## 1 1380    768    263        57        82        1000         105       6694
## 2  434    321    141        28        53         624         269       4600
```

```
## 3 2013   1053    212         33         61         912          158        3036
## 4 2324   1319    370         52         81        1686           35        5140
## 5 1709   1385    634         36         72        2281           50        3600
## 6  427    385    143         18         38         581          533        5800
##   Books Personal PhD Terminal S.F.Ratio perc.alumni Expend Grad.Rate
## 1   600      700  89       93       6.1          18  14779        83
## 2   550      950  79       82      12.9          30   9264        81
## 3   500     1655  64       74      10.5          11   7547        59
## 4   558     1152  91       93      10.5          30  16196        79
## 5   400      700  79       89      12.5          58   9907        80
## 6   450      700  81       85      10.3          37  11758        84
```

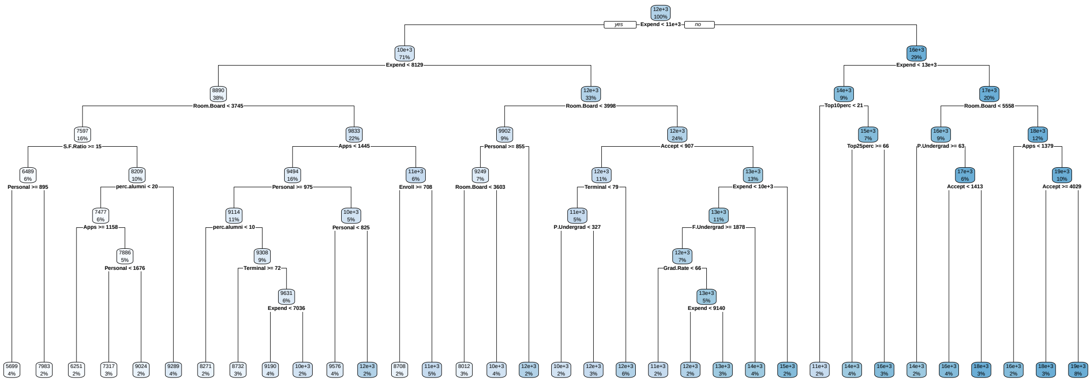
```
y = training_data$Outstate # vector of response

# testing data
x2 = model.matrix(Outstate ~ .,testing_data)[, -1] # matrix of predictors
y2 = testing_data$Outstate # vector of response
```

(a) Build a regression tree on the training data to predict the response. Create a plot of the tree.

```r
# setting a 10-fold cross-validation
ctrl = trainControl(method = "cv", number = 10)

set.seed(2)

rpart.fit = train(Outstate ~ . ,
                  training_data,
                  method = "rpart",
                  tuneGrid = data.frame(cp = exp(seq(-8,-1, length = 100))),
                  trControl = ctrl)

plot(rpart.fit, xTrans = log)
```

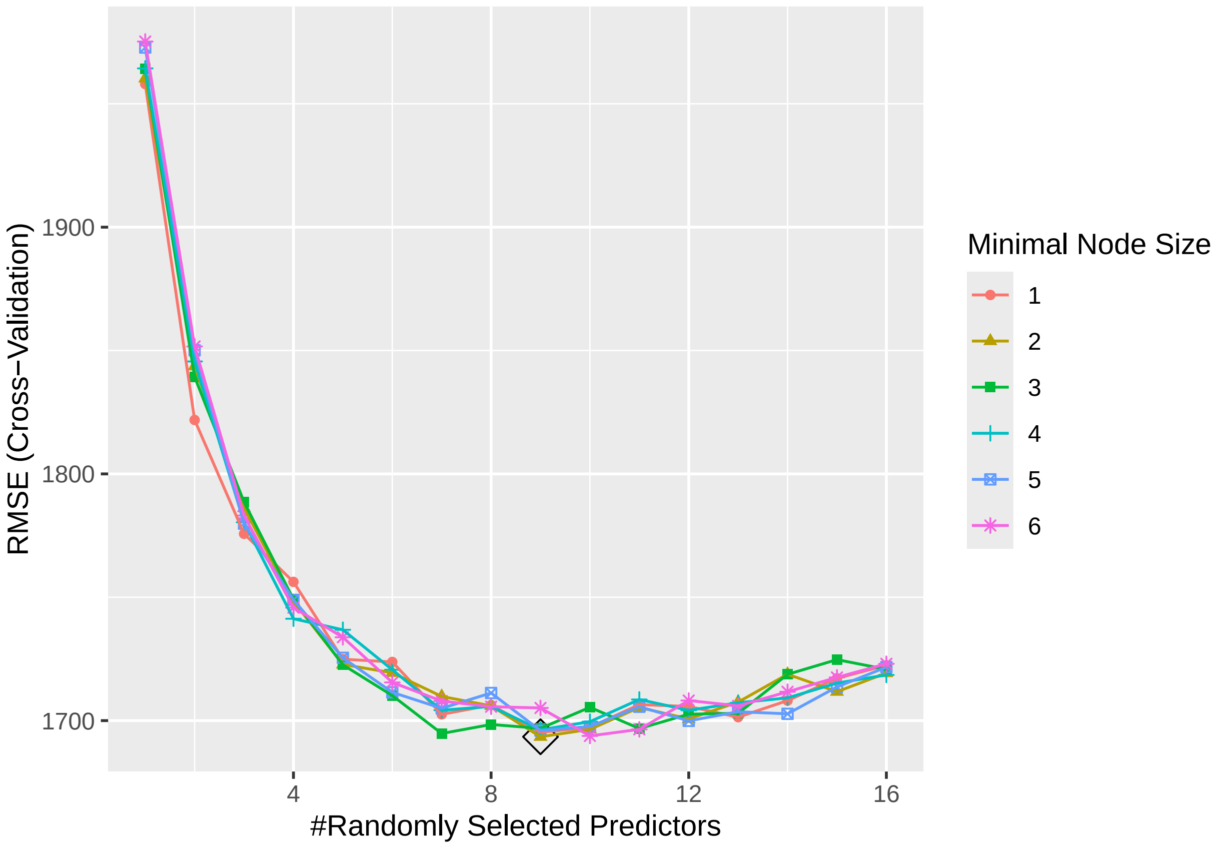



```r
# plot of the tree
rpart.plot(rpart.fit$finalModel)
```



(b) Perform random forest on the training data. Report the variable importance and the test error.

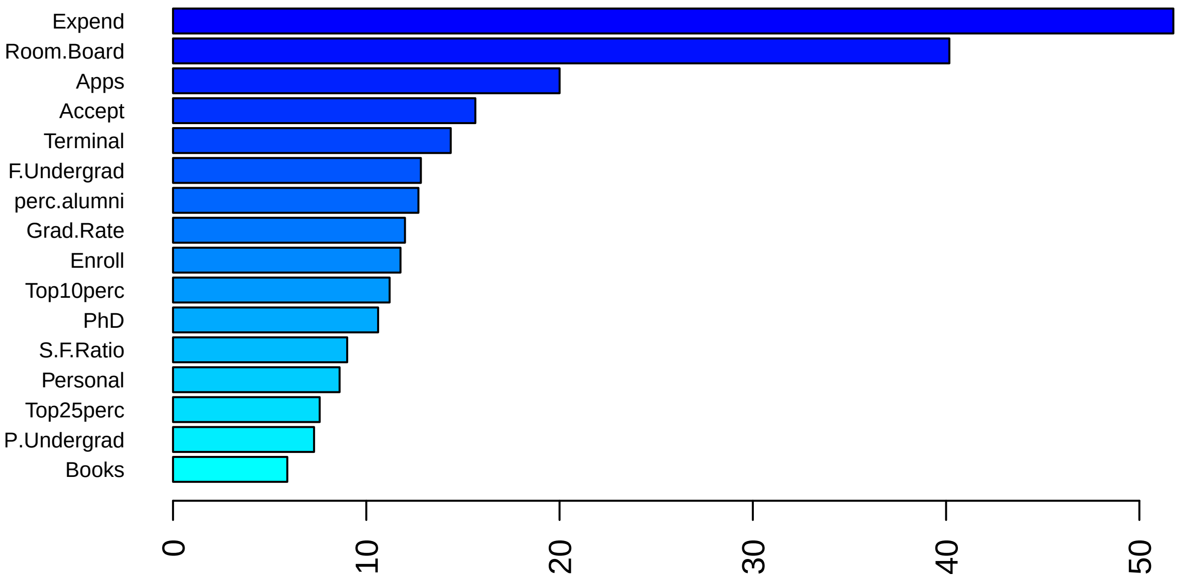
```r
# Try more if possible
rf.grid = expand.grid(mtry = 1:16,
                      splitrule = "variance",
                      min.node.size = 1:6)

set.seed(2)

# regression random forest
rf.fit = train(Outstate ~ . ,
               data = training_data,
               method = "ranger",
               tuneGrid = rf.grid,
               trControl = ctrl)

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




```
rf.fit$bestTune
```

```
##    mtry splitrule min.node.size
## 50    9  variance             2
```

```
set.seed(2)

# variable importance
rf2.final.per = ranger(Outstate ~.,
                       data = training_data,
                       mtry = rf.fit$bestTune[[1]],
                       splitrule = "variance",
                       min.node.size = rf.fit$bestTune[[3]],
                       importance = "permutation",
                       scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf2.final.per), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan", "blue"))(16))
```

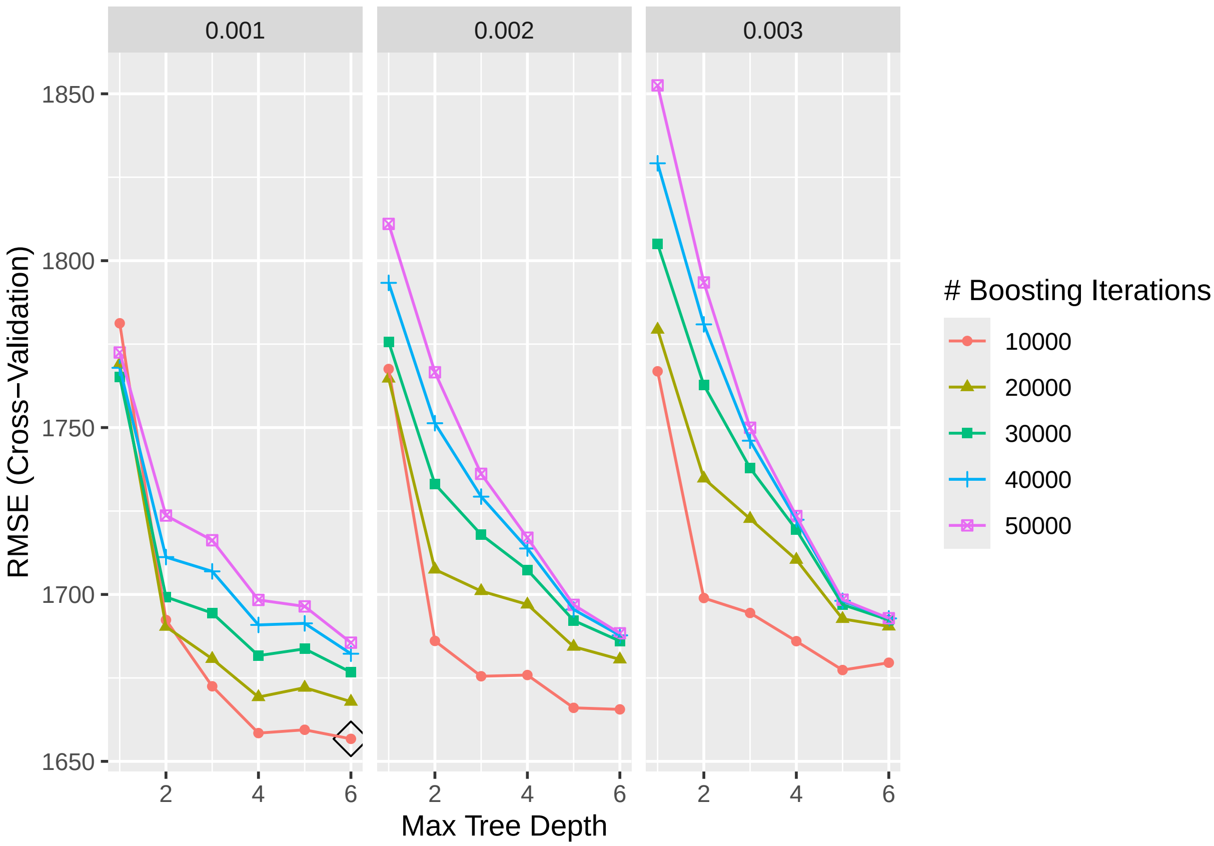



```r
# test error
pred.rf.fit = predict(rf.fit, newdata = testing_data)

test.error.rf = mean((pred.rf.fit - y2)^2)
```

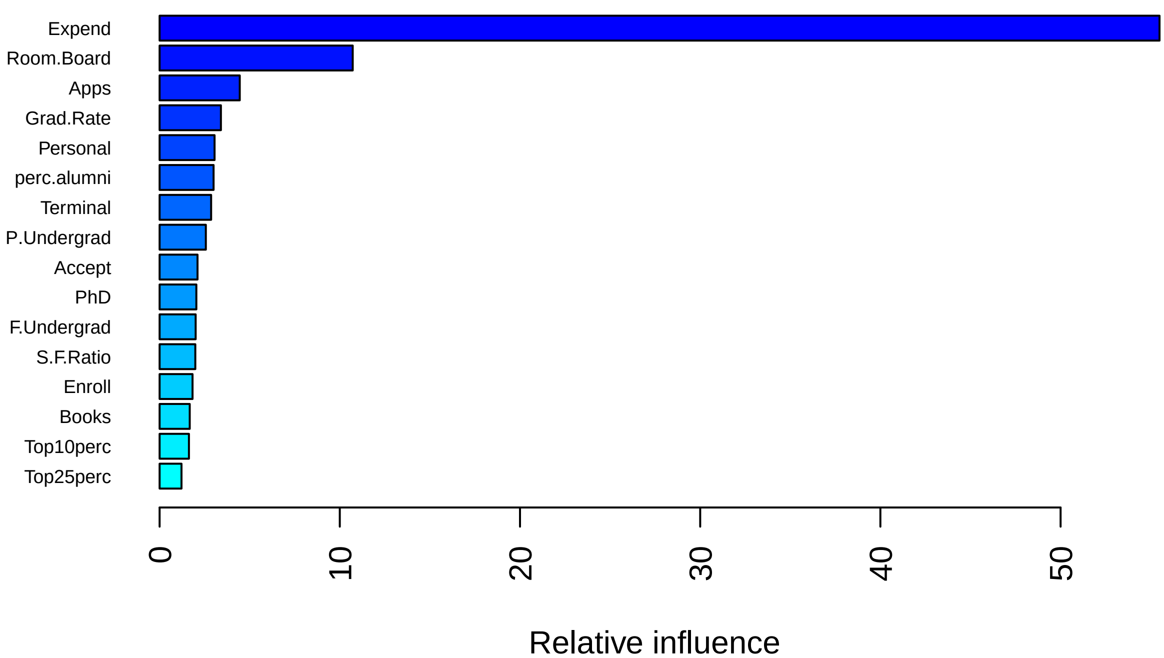
The test error for the random forest model is 3.3961713×10^6.

(c) Perform boosting on the training data. Report the variable importance and the test error.

```r
gbm.grid = expand.grid(n.trees = c(10000,20000,30000,40000,50000),
                       interaction.depth = 1:6,
                       shrinkage = c(0.001,0.002,0.003),
                       n.minobsinnode = c(1))
set.seed(2)

# boosting
gbm.fit = train(Outstate ~ . ,
                data = training_data,
                method = "gbm",
                tuneGrid = gbm.grid,
                trControl = ctrl,
                verbose = FALSE)

ggplot(gbm.fit, highlight = TRUE)
```




```
# variable importance
summary(gbm.fit$finalModel, las = 2, cBars = 16, cex.names = 0.6)
```




```
##                      var   rel.inf
## Expend            Expend 55.490745
## Room.Board    Room.Board 10.717518
```

```
## Apps               Apps  4.447935
## Grad.Rate     Grad.Rate  3.402716
## Personal       Personal  3.050114
## perc.alumni perc.alumni  2.994665
## Terminal       Terminal  2.859568
## P.Undergrad P.Undergrad  2.566812
## Accept           Accept  2.103675
## PhD                 PhD  2.039593
## F.Undergrad F.Undergrad  2.001172
## S.F.Ratio     S.F.Ratio  1.982115
## Enroll           Enroll  1.827980
## Books             Books  1.672953
## Top10perc     Top10perc  1.627772
## Top25perc     Top25perc  1.214668
```

```r
# test error
pred.gbm.fit <- predict(gbm.fit, newdata = testing_data)

test.error.gbm <- mean((pred.gbm.fit - y2)^2)
```

The test error for the gbm model is 3.3282959×10^6.

Question 2

This problem is based on the data `auto.csv` in Homework 3. The dataset contains 392 observations.

The response variable is `mpg_cat`, which indicates whether the miles per gallon of a car is high or low.

The predictors are:

- `cylinders`: Number of cylinders between 4 and 8
- `displacement`: Engine displacement (cu. inches)
- `horsepower`: Engine horsepower
- `weight`: Vehicle weight (lbs.)
- `acceleration`: Time to accelerate from 0 to 60 mph (sec.)
- `year`: Model year (modulo 100)
- `origin`: Origin of car (1. American, 2. European, 3. Japanese)

Split the dataset into two parts: training data (70%) and test data (30%).

```r
auto = read_csv("data/auto.csv") |>
  drop_na() |>
  mutate(
    mpg_cat = as.factor(mpg_cat),
    mpg_cat = forcats::fct_relevel(mpg_cat, c("low", "high")),
    cylinders = as.factor(cylinders),
    origin = as.factor(origin)
  )

set.seed(2)

# create a random split of 70% training and 30% test data
data_split2 = initial_split(data = auto, prop = 0.7)

# partitioned datasets
training_data2 = training(data_split2)
testing_data2 = testing(data_split2)

head(training_data2)
```

```
## # A tibble: 6 x 8
##   cylinders displacement horsepower weight acceleration  year origin mpg_cat
##   <fct>            <dbl>      <dbl>  <dbl>        <dbl> <dbl> <fct>  <fct>
## 1 4                   86         64   1875         16.4    81 1      high
## 2 6                  225        100   3651         17.7    76 1      low
## 3 6                  231        165   3445         13.4    78 1      low
## 4 5                  131        103   2830         15.9    78 2      low
## 5 4                   98         65   2380         20.7    81 1      high
## 6 4                   97         75   2155         16.4    76 3      high
```

```r
head(testing_data2)
```

```
## # A tibble: 6 x 8
##   cylinders displacement horsepower weight acceleration  year origin mpg_cat
##   <fct>            <dbl>      <dbl>  <dbl>        <dbl> <dbl> <fct>  <fct>
## 1 8                  302        140   3449         10.5    70 1      low
## 2 8                  390        190   3850          8.5    70 1      low
```

```
## 3 4                    113          95    2372          15       70 3       high
## 4 6                    200          85    2587          16       70 1       low
## 5 4                     97          88    2130          14.5     70 3       high
## 6 4                    107          90    2430          14.5     70 2       high
```

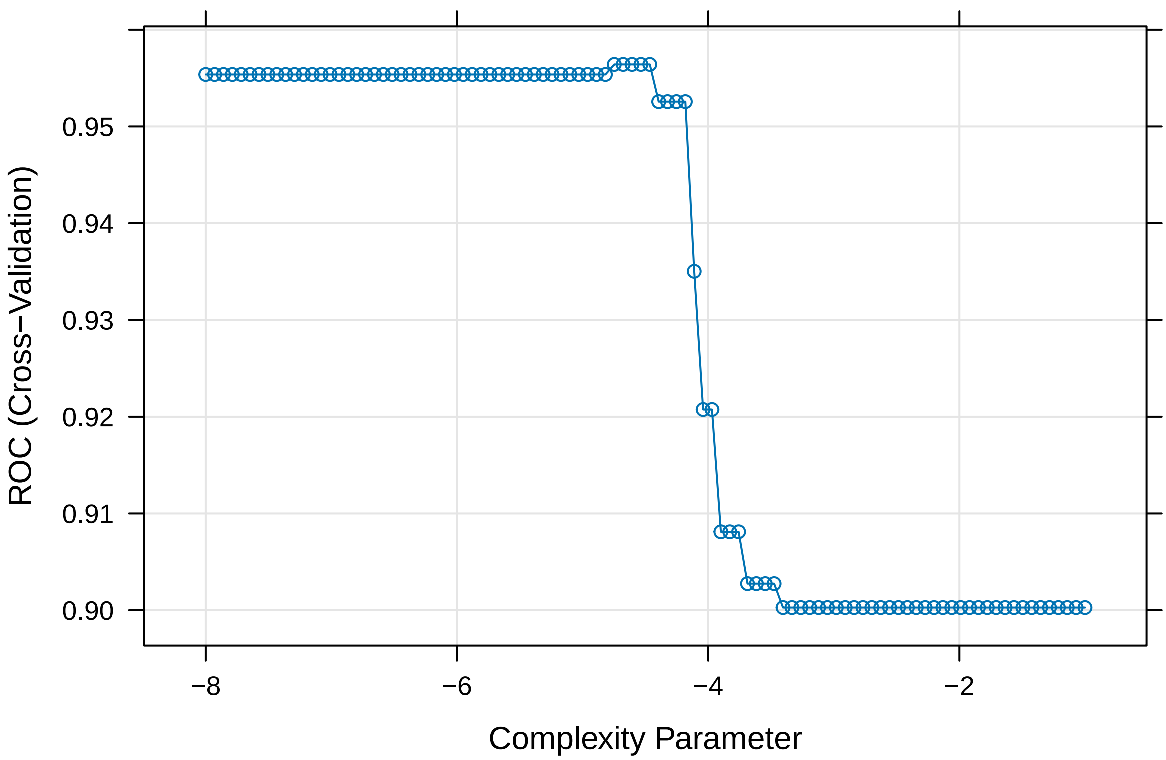
```r
# training data
x_1 = model.matrix(mpg_cat ~ ., training_data2)[, -1] # matrix of predictors
head(x_1)
```

```
##   cylinders4 cylinders5 cylinders6 cylinders8 displacement horsepower weight
## 1          1          0          0          0           86         64   1875
## 2          0          0          1          0          225        100   3651
## 3          0          0          1          0          231        165   3445
## 4          0          1          0          0          131        103   2830
## 5          1          0          0          0           98         65   2380
## 6          1          0          0          0           97         75   2155
##   acceleration year origin2 origin3
## 1         16.4   81       0       0
## 2         17.7   76       0       0
## 3         13.4   78       0       0
## 4         15.9   78       1       0
## 5         20.7   81       0       0
## 6         16.4   76       0       1
```

```r
y_1 = training_data2$mpg_cat # vector of response

# testing data
x_2 = model.matrix(mpg_cat ~ .,testing_data2)[, -1] # matrix of predictors
y_2 = testing_data2$mpg_cat # vector of response
```

(a) Build a classification tree using the training data, with `mpg_cat` as the response. Which tree size corresponds to the lowest cross-validation error? Is this the same as the tree size obtained using the 1SE rule?

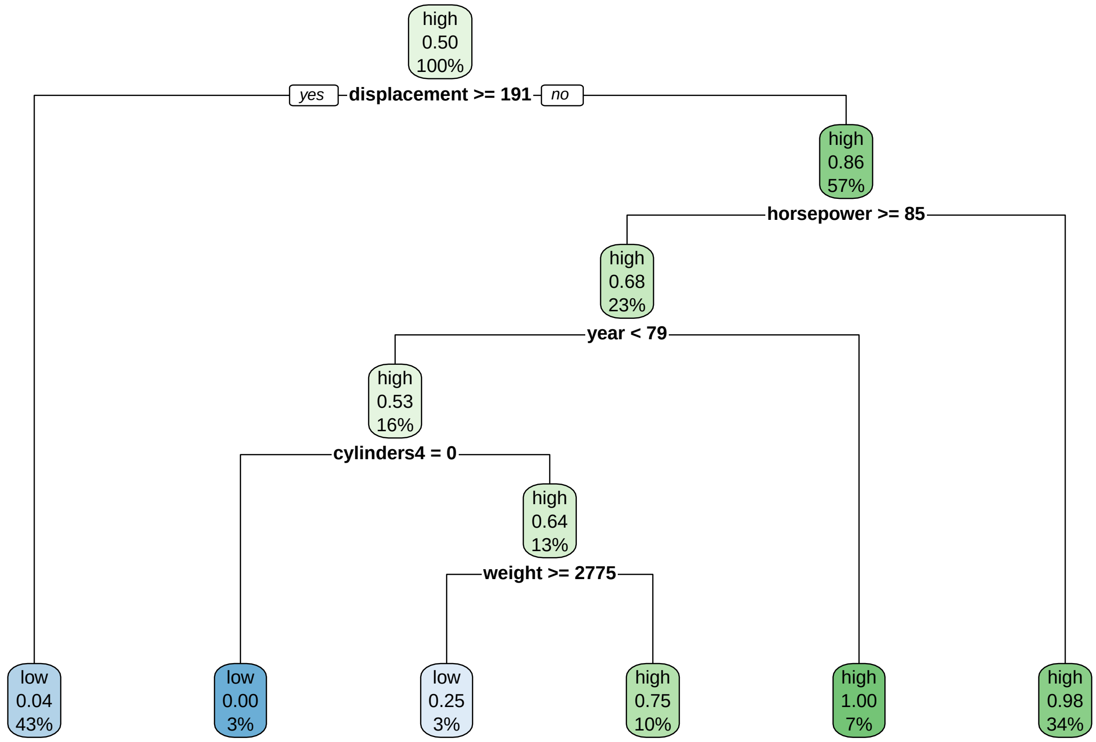
```
ctrl1 = trainControl(method = "cv",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)

set.seed(2)

# CART
cart.fit = train(mpg_cat ~ . ,
                 training_data2,
                 method = "rpart",
                 tuneGrid = data.frame(cp = exp(seq(-8,-1, len = 100))),
                 trControl = ctrl1,
                 metric = "ROC")

plot(cart.fit, xTrans = log)
```



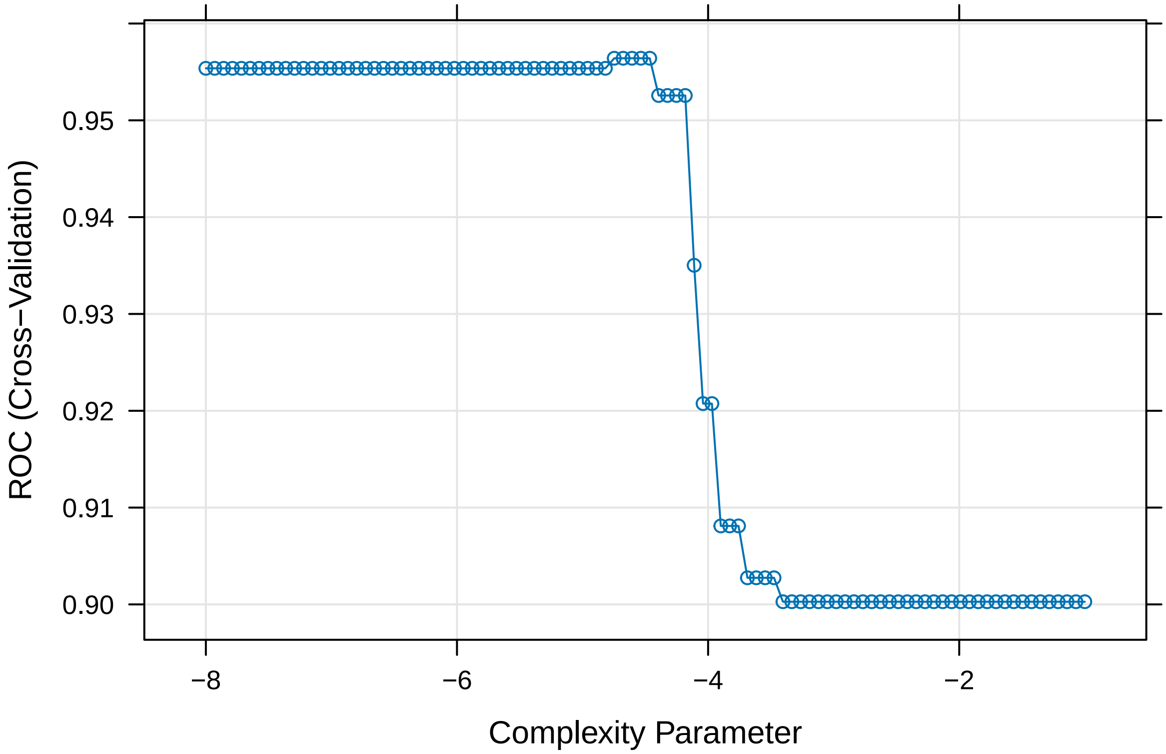

```
cart.fit$bestTune
```

```
##            cp
## 51 0.01150876
```

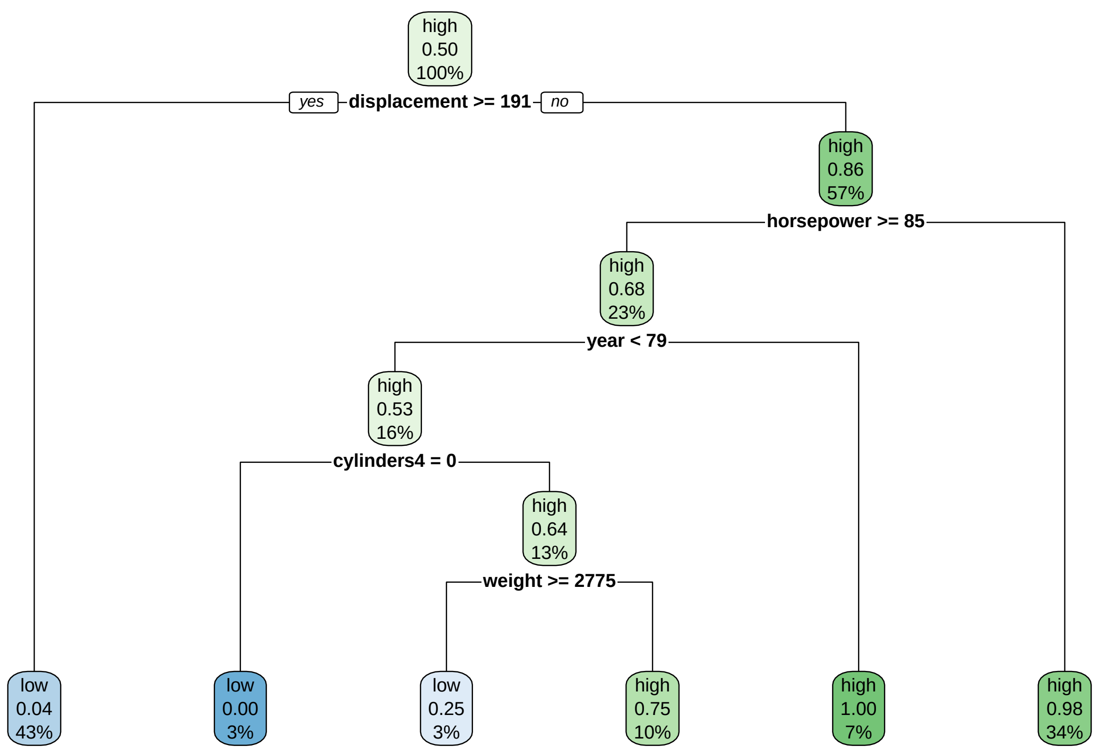
```
rpart.plot(cart.fit$finalModel)
```




```r
# Which tree size corresponds to the lowest cross-validation error? Is this the same as the tree size o

# using the 1SE rule:
ctrl2 = trainControl(method = "cv",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     selectionFunction = "oneSE")

set.seed(2)

cart.fit2 = train(mpg_cat ~ . ,
                  training_data2,
                  method = "rpart",
                  tuneGrid = data.frame(cp = exp(seq(-8,-1, len = 100))),
                  trControl = ctrl2,
                  metric = "ROC")
plot(cart.fit2, xTrans = log)
```




```r
rpart.plot(cart.fit2$finalModel)
```



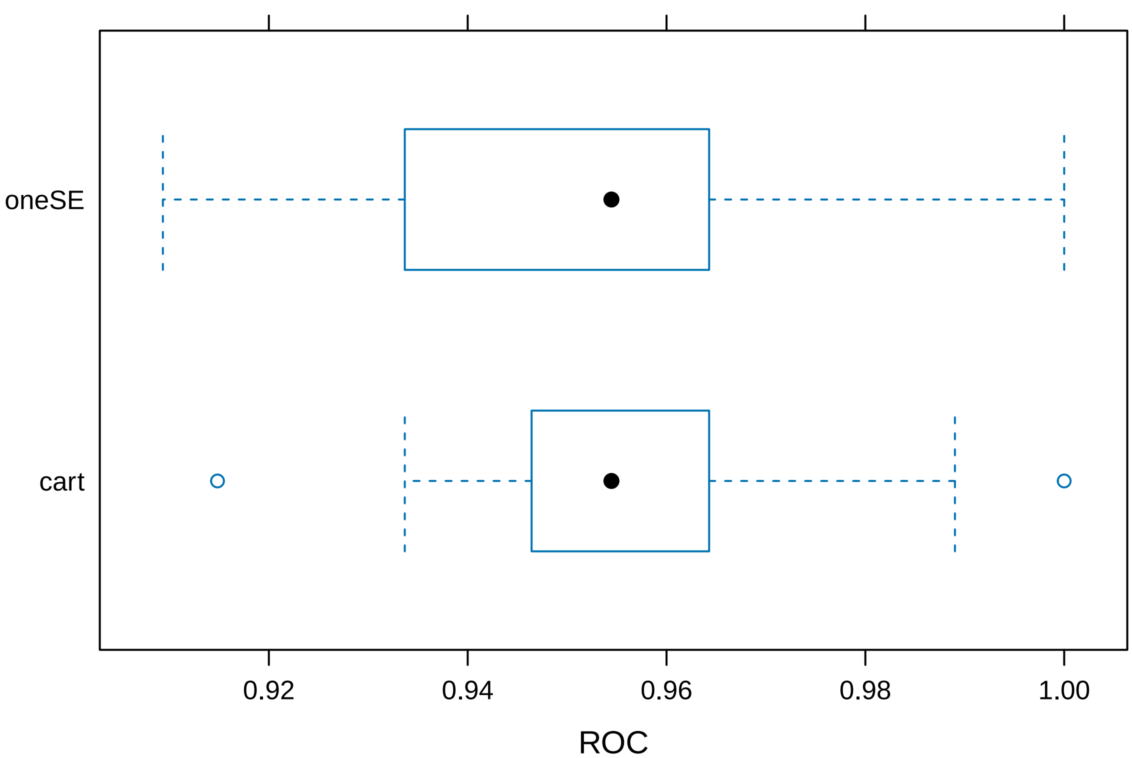

```r
cart.fit2$bestTune
```

```
##            cp
## 55 0.01527072
```

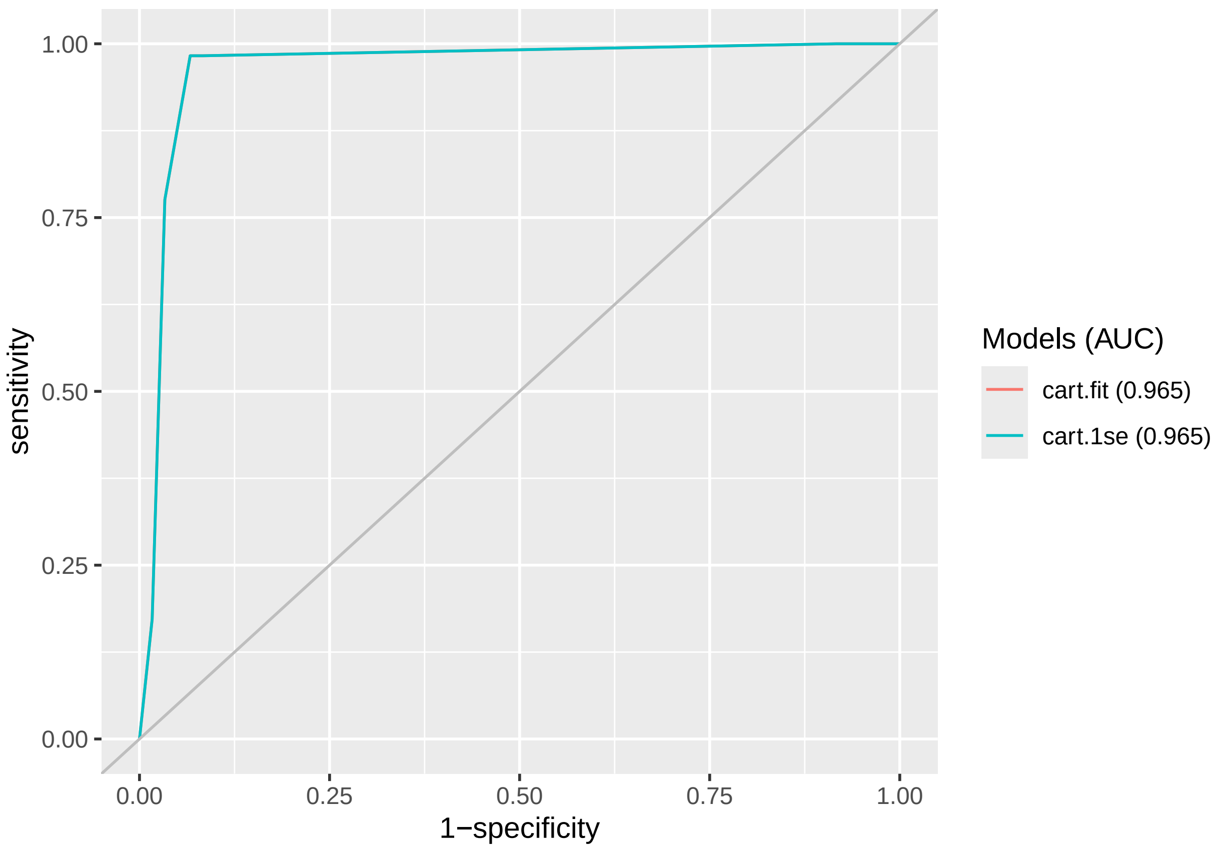
```r
# resampling comparison
rpart.plot(cart.fit2$finalModel)

resamp = resamples(list(cart = cart.fit, oneSE = cart.fit2))
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: cart, oneSE
## Number of resamples: 10
##
## ROC
##            Min.   1st Qu.   Median      Mean   3rd Qu. Max. NA's
## cart  0.9148352 0.9467720 0.954459 0.9564138 0.9630102    1    0
## oneSE 0.9093407 0.9368622 0.954459 0.9525676 0.9630102    1    0
##
## Sens
##            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## cart  0.7692308 0.8008242 0.9258242 0.8967033 0.9821429    1    0
## oneSE 0.7692308 0.8008242 0.9258242 0.8967033 0.9821429    1    0
##
## Spec
##            Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## cart  0.8461538 0.9285714 0.9285714 0.9274725 0.9285714    1    0
## oneSE 0.8461538 0.9285714 0.9285714 0.9346154 0.9821429    1    0
```

```r
bwplot(resamp, metric = "ROC")
```



```
# predicition
cart.fit.pred <- predict(cart.fit, newdata = testing_data2, type = "prob")[,2]
cart.fit2.pred <- predict(cart.fit2, newdata = testing_data2, type = "prob")[,2]

# ROC curves (for AUC)
roc.cart <- roc(testing_data2$mpg_cat, cart.fit.pred)
roc.cart2 <- roc(testing_data2$mpg_cat, cart.fit2.pred)

# AUC values
auc <- c(roc.cart$auc[1], roc.cart2$auc[1])
modelNames <- c("cart.fit", "cart.1se")

# combined ROC curves
ggroc(list(roc.cart, roc.cart2),
      legacy.axes = TRUE) +
  scale_color_discrete(labels = paste0(modelNames, " (", round(auc,3),")"),
                       name = "Models (AUC)") + geom_abline(intercept = 0, slope = 1, color = "grey")
```




```
# missclassification error
misclass = predict(cart.fit, newdata = testing_data2, type = "raw")
misclass2 = predict(cart.fit2, newdata = testing_data2, type = "raw")
# Convert character labels to binary
misclass_binary <- ifelse(misclass == "low", 0, 1)
misclass_binary2 <- ifelse(misclass2 == "low", 0, 1)
# take the mean of the logical vector
mean(misclass_binary)
```

```
## [1] 0.5169492
```

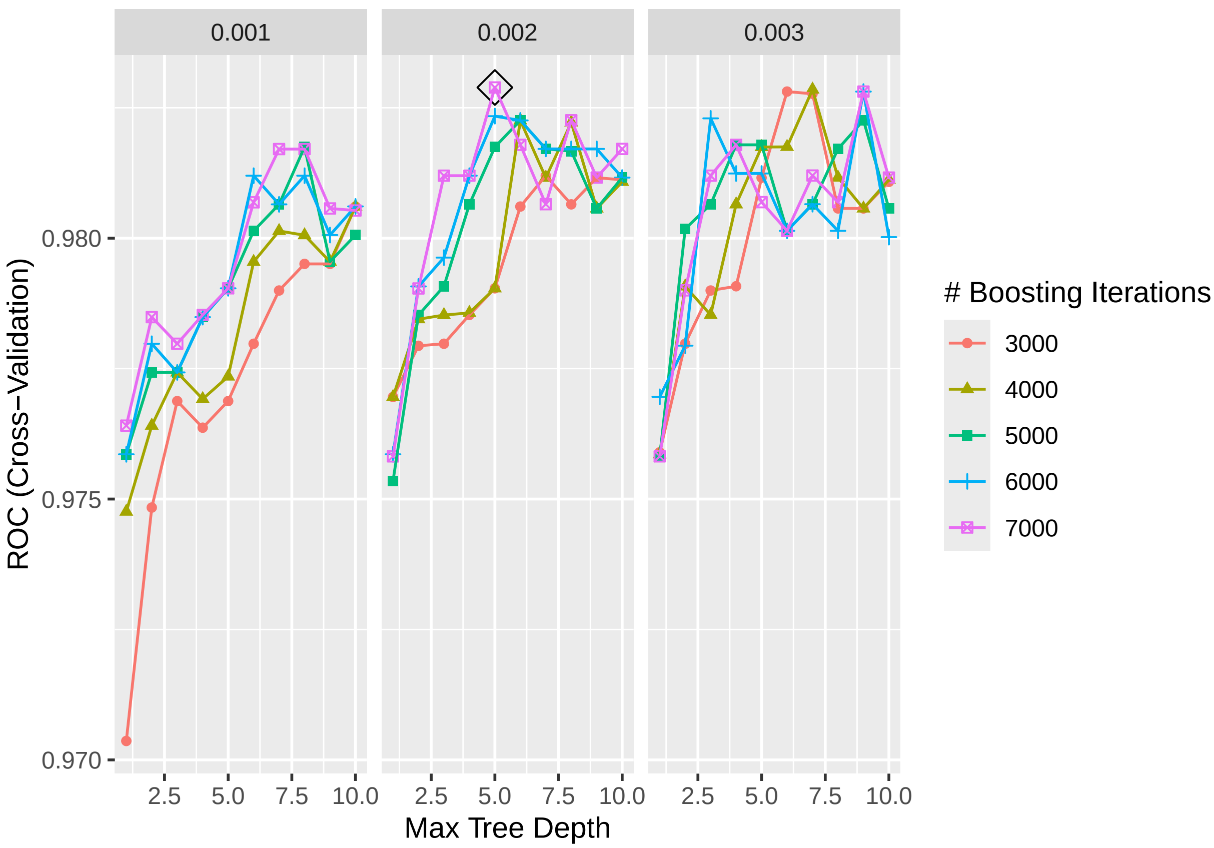
```r
mean(misclass_binary2)
```

```
## [1] 0.5169492
```

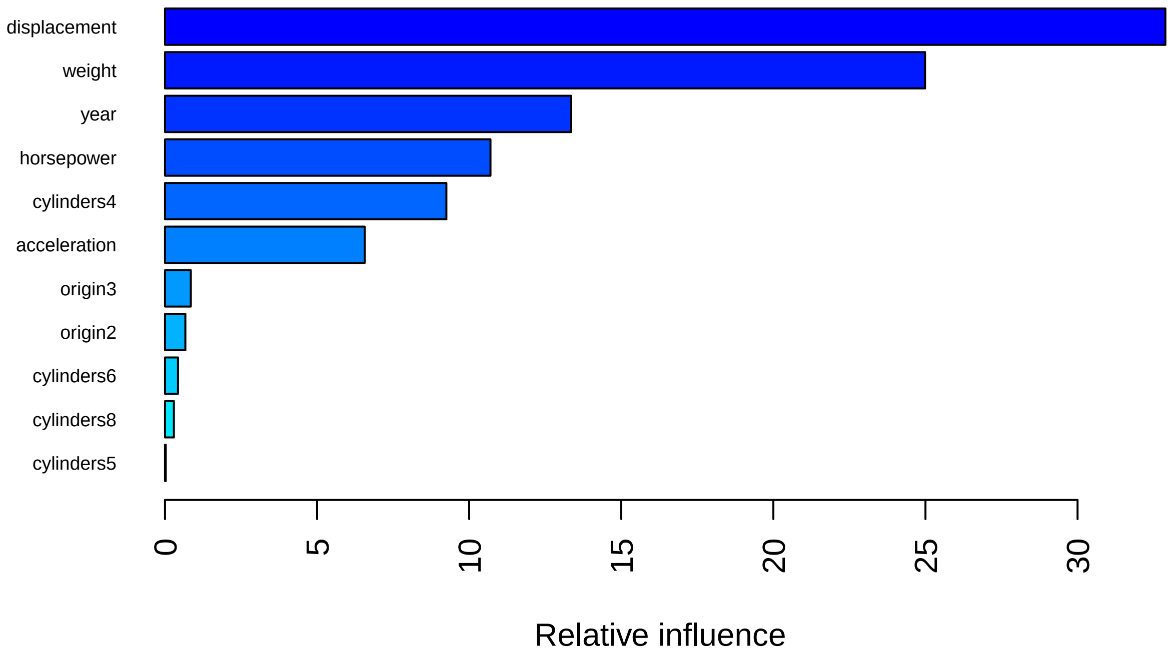
From the re-sampling summary, we can see the minSE model has the highest mean ROC, however the model using 1SE has the same median ROC. Both models however, have the same AUC and same misclassification error rate, indicating comparative predictive performance. Even when changing seeds, the tree sizes for both models are the same. The tree size for both trees is 6.

(b) Perform boosting on the training data and report the variable importance. Report the test data performance.

```
# set grid
gbmA.grid =
  expand.grid(n.trees = c(3000,4000,5000,6000,7000),
              interaction.depth = 1:10,
              shrinkage = c(0.001, 0.002, 0.003),
              n.minobsinnode = 1)

set.seed(2)

# boosting
gbmA.fit = train(mpg_cat ~ . ,
                 training_data2,
                 tuneGrid = gbmA.grid,
                 trControl = ctrl1,
                 method = "gbm",
                 distribution = "adaboost",
                 metric = "ROC",
                 verbose = FALSE)

ggplot(gbmA.fit, highlight = TRUE)
```




```
# variable importance
summary(gbmA.fit$finalModel, las = 2, cBars = 13, cex.names = 0.6)
```




```
##                       var    rel.inf
## displacement displacement 32.8875939
## weight             weight 24.9865605
## year                 year 13.3480629
## horsepower     horsepower 10.7001898
## cylinders4     cylinders4  9.2503115
## acceleration acceleration  6.5653178
## origin3           origin3  0.8485400
## origin2           origin2  0.6693138
## cylinders6     cylinders6  0.4275195
## cylinders8     cylinders8  0.2916469
## cylinders5     cylinders5  0.0249434
```

```r
# test data performance (ROC): missclassification error
misclass_gbmA = predict(gbmA.fit, newdata = testing_data2, type = "raw")

# Convert character labels to binary
misclass_binary_gbmA = ifelse(misclass_gbmA == "low", 0, 1)

# take the mean of the logical vector
mean(misclass_binary_gbmA)
```

```
## [1] 0.5
```

The misclassification error rate for the gbmA model is 0.5