# P8106 HW 4

Minjie Bao

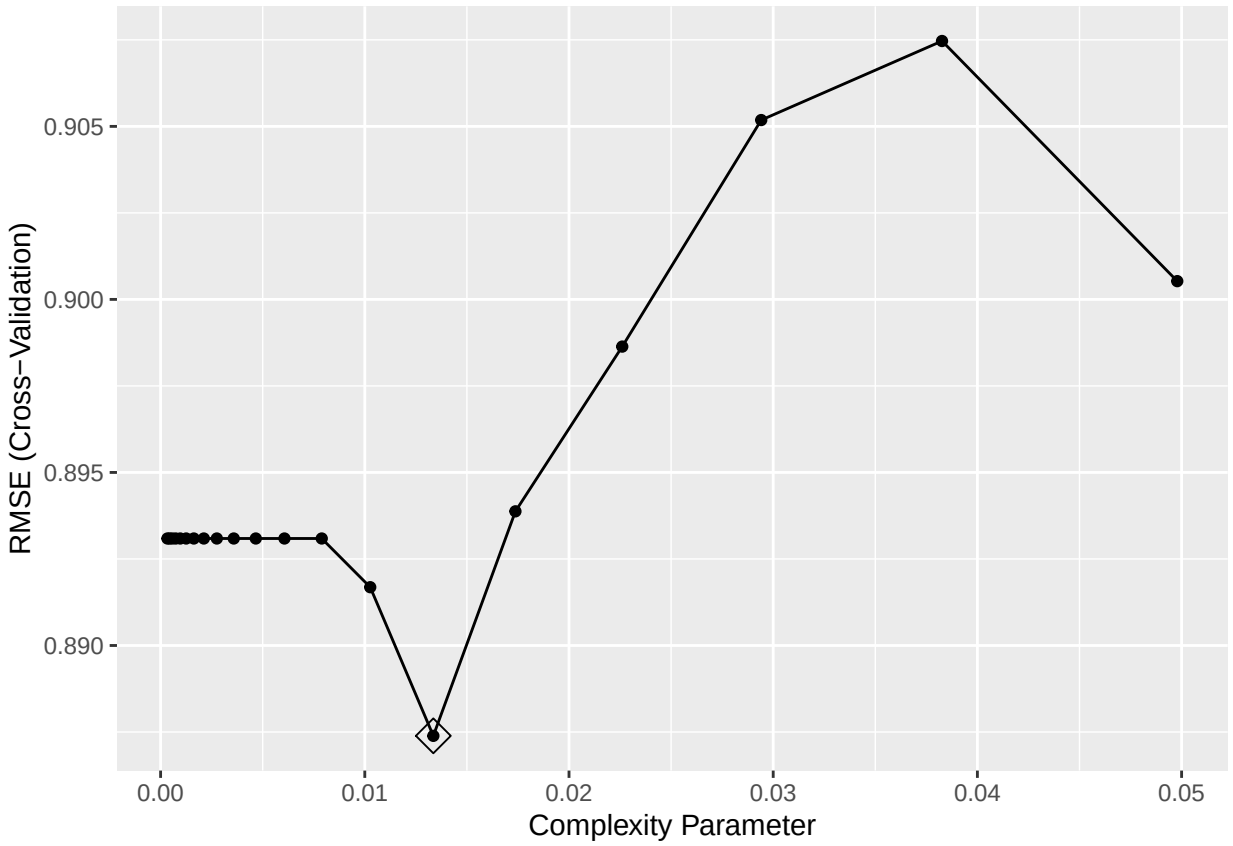
## Contents

```r
library(ISLR)
library(lasso2)
library(tidyverse)
library(caret)
library(mlbench)
library(rpart)
library(rpart.plot)
library(party)
library(partykit)
library(randomForest)
library(gbm)
library(plotmo)
library(pdp)
library(lime)
library(ranger)

set.seed(2021)
```

# Problem 1

**(a) Fit a regression tree with lpsa as the response and the other variables as predictors. Use cross-validation to determine the optimal tree size. Which tree size corresponds to the lowest cross-validation error? Is this the same as the tree size obtained using the 1 SE rule?**

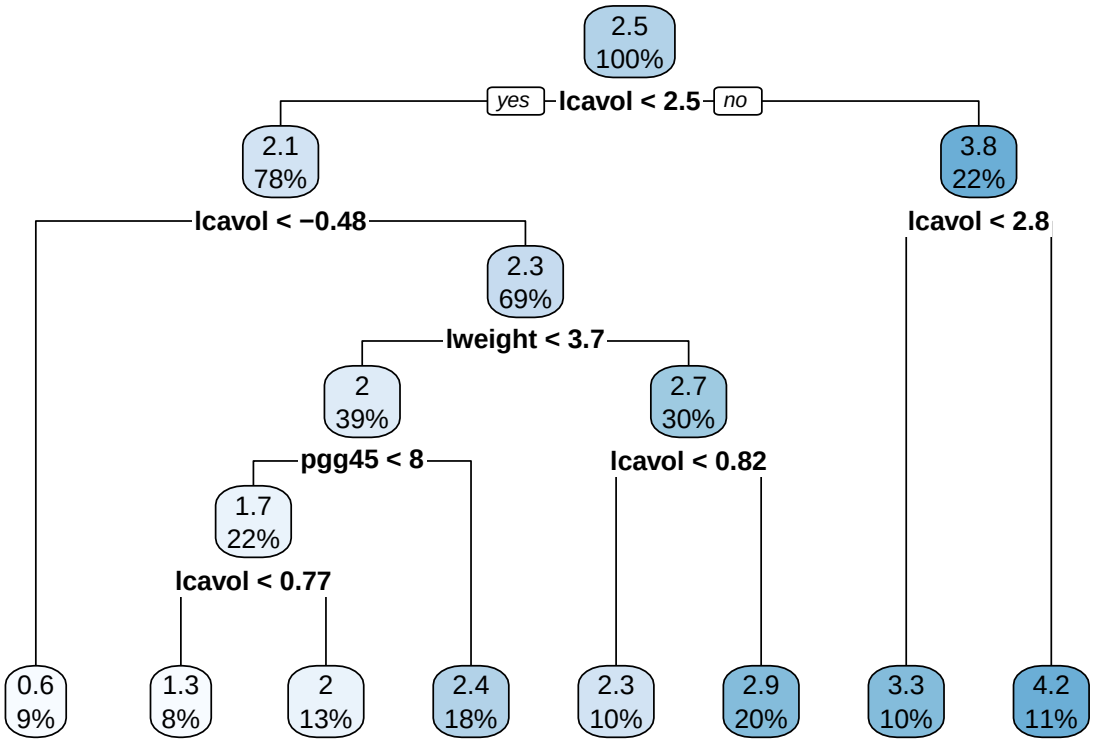
```r
data(Prostate)
#head(Prostate)

ctrl = trainControl(method = "cv")

# minimum cross-validation error
set.seed(2021)
tree.fit = train(lpsa~.,
             data = Prostate,
             method = "rpart",
             tuneGrid = data.frame(cp =exp(seq(-8, -3, length = 20))),
             trControl = ctrl)
ggplot(tree.fit, highlight = TRUE)
```



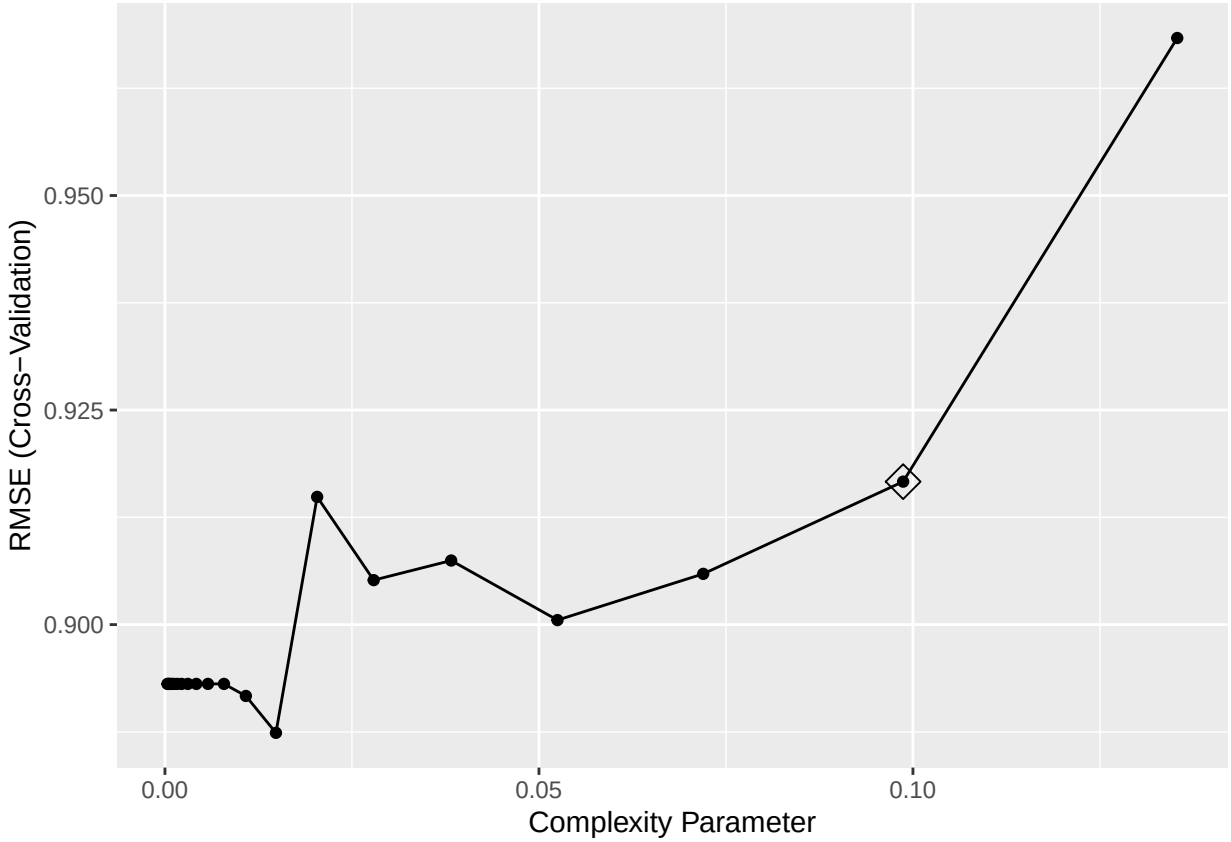

```
tree.fit$finalModel$cptable
```

```
##           CP nsplit rel error
## 1 0.34710828      0 1.0000000
## 2 0.18464743      1 0.6528917
## 3 0.05931585      2 0.4682443
## 4 0.03475635      3 0.4089284
## 5 0.03460901      4 0.3741721
## 6 0.02156368      5 0.3395631
## 7 0.02146995      6 0.3179994
## 8 0.00000000      7 0.2965295
```

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



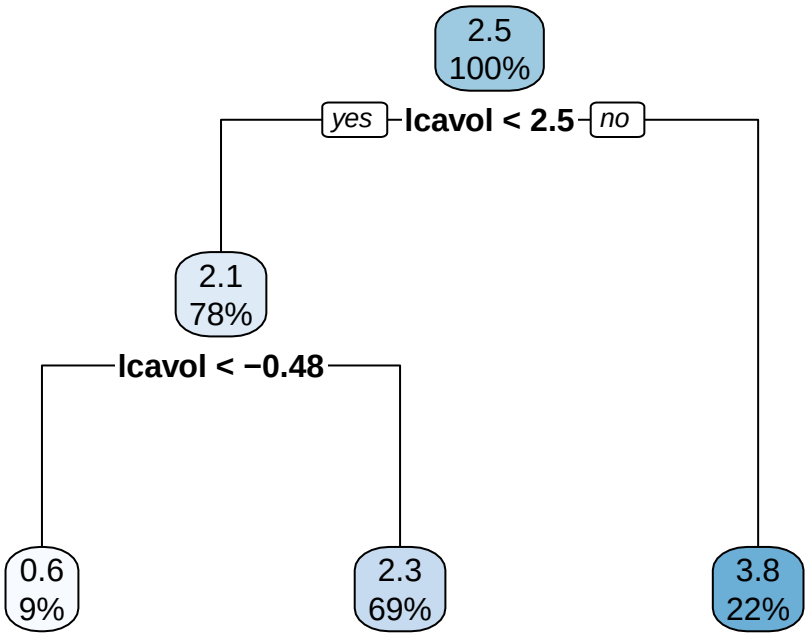

```
#1 SE rule
set.seed(2021)
tree.fit2 = train(lpsa~.,
            data = Prostate,
            method = "rpart",
            tuneGrid = data.frame(cp =exp(seq(-8,-2, length = 20))),
            trControl = trainControl(method = "cv",
                                     number = 10,
                                     selectionFunction = "oneSE"))
ggplot(tree.fit2, highlight = TRUE)
```




```
tree.fit2$finalModel$cptable
```

```
##           CP nsplit rel error
## 1 0.34710828      0 1.0000000
## 2 0.18464743      1 0.6528917
## 3 0.09868824      2 0.4682443
```

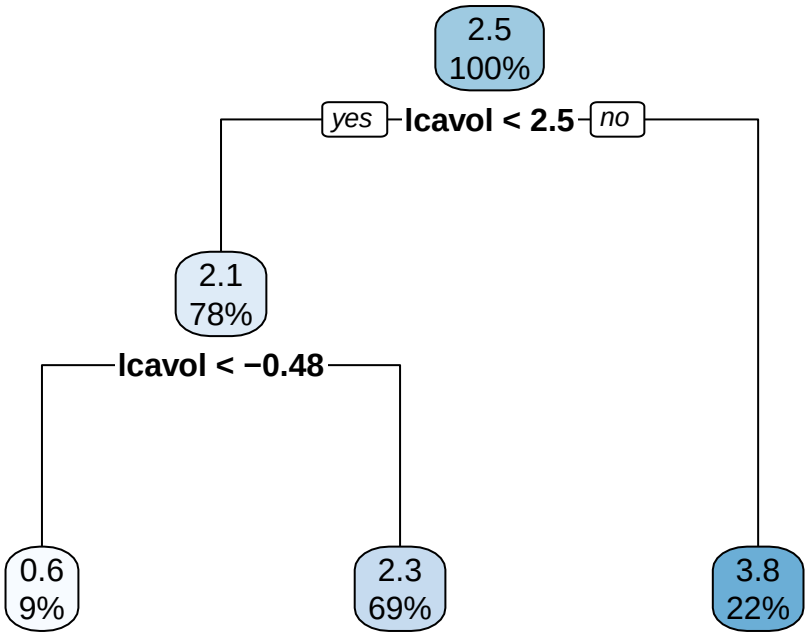
```
rpart.plot(tree.fit2$finalModel)
```




The optimal tree size corresponds to the lowest cross validation error is 8. However, the optimal tree size obtained using the 1 SE rule is 3. Therefore, these two methods' optimal tree sizes are different.

**(b) Create a plot of the final tree you choose. Pick one of the terminal nodes, and interpret the information displayed.**

```
set.seed(2021)
final_tree <- rpart(lpsa~., data = Prostate,
                    control = rpart.control(cp = 0.1))

rpart.plot(final_tree)
```

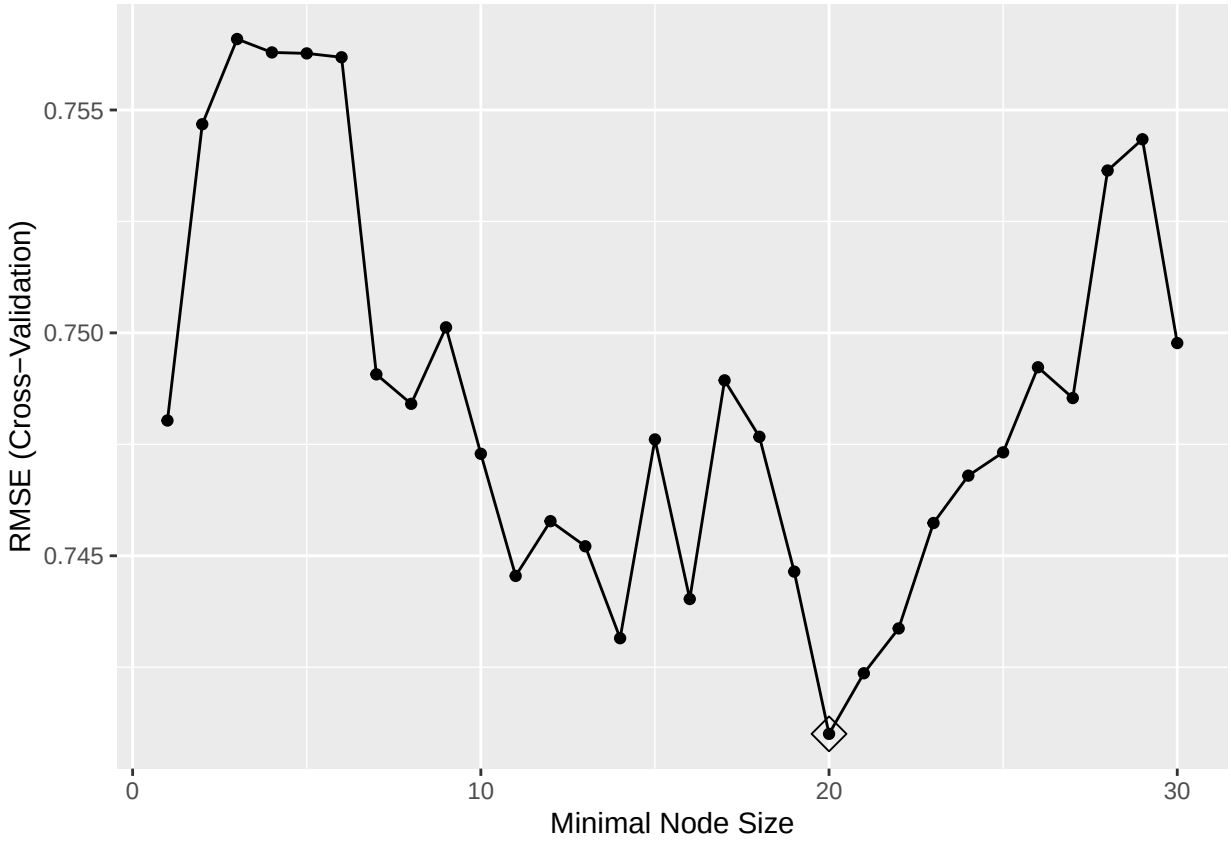


I choose the final model with cp = 0.1 and tree size = 3.

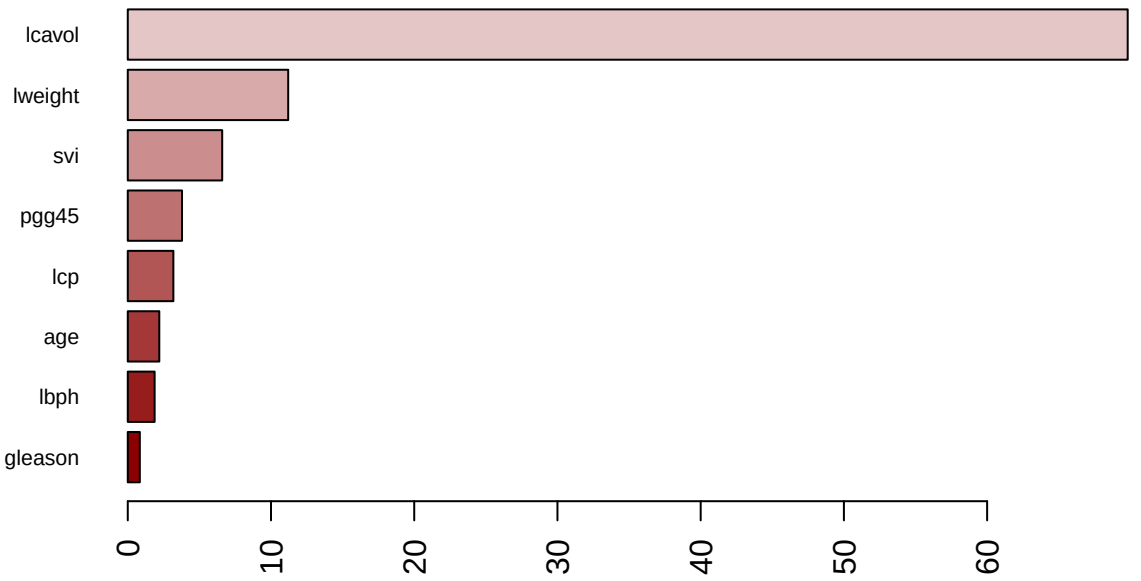
Interpretation: From the tree plot, we can see that in the terminal node where log cancer volumn(lcavol) < 2.5, there is 78% chance for log prostate specific antigen to be 2.1. If the log cancer volumn((lcavol)) is not < 2.5, there is 22% chance for log prostate specific antigen to be 3.8.

**(c) Perform bagging and report the variable importance.**

```
set.seed(1)
ctrl <- trainControl(method = "cv")

bag.grid <- expand.grid(mtry = 8,
                        splitrule = "variance",
                        min.node.size = 1:30)

bag.fit <- train(lpsa~., Prostate,
                 method = "ranger",
                 tuneGrid = bag.grid,
                 trControl = ctrl,
                 importance = 'impurity')

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



```
barplot(sort(ranger::importance(bag.fit$finalModel), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("darkred","white","darkblue"))(19))
```

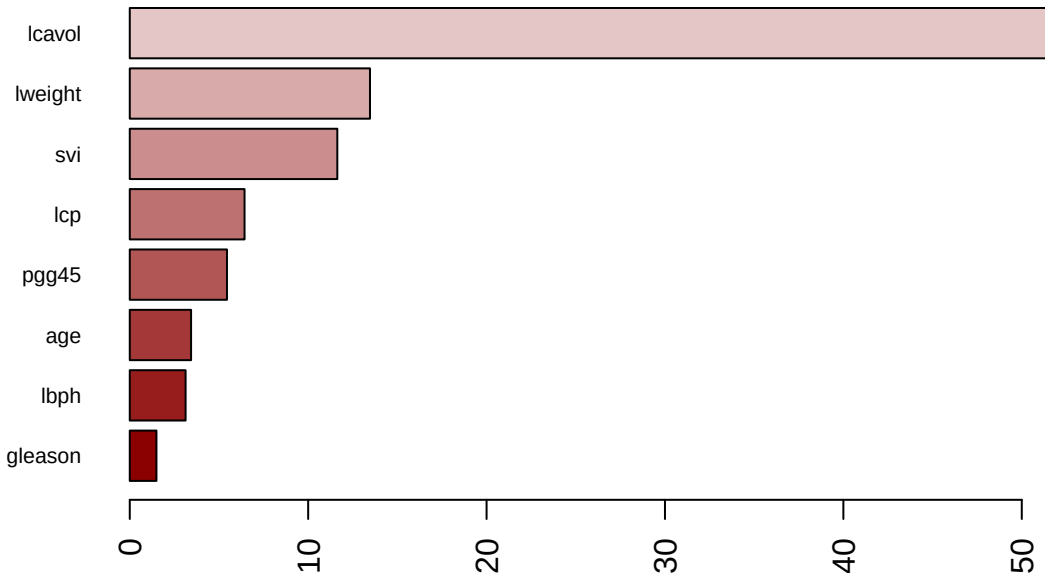



The importance of variables from highest to lowest using bagging method is: lcavol, lweight, svi, pgg45, lcp, age, lbph, and gleason.

**(d) Perform random forest and report the variable importance.**

```
rf.grid <- expand.grid(mtry = 1:6,
                       splitrule = "variance",
                       min.node.size = 1:30)

set.seed(2021)
rf.fit <- train(lpsa~., Prostate,
                method = "ranger",
                tuneGrid = rf.grid,
                trControl = ctrl,
                importance = 'impurity')
ggplot(rf.fit, highlight = TRUE)
```

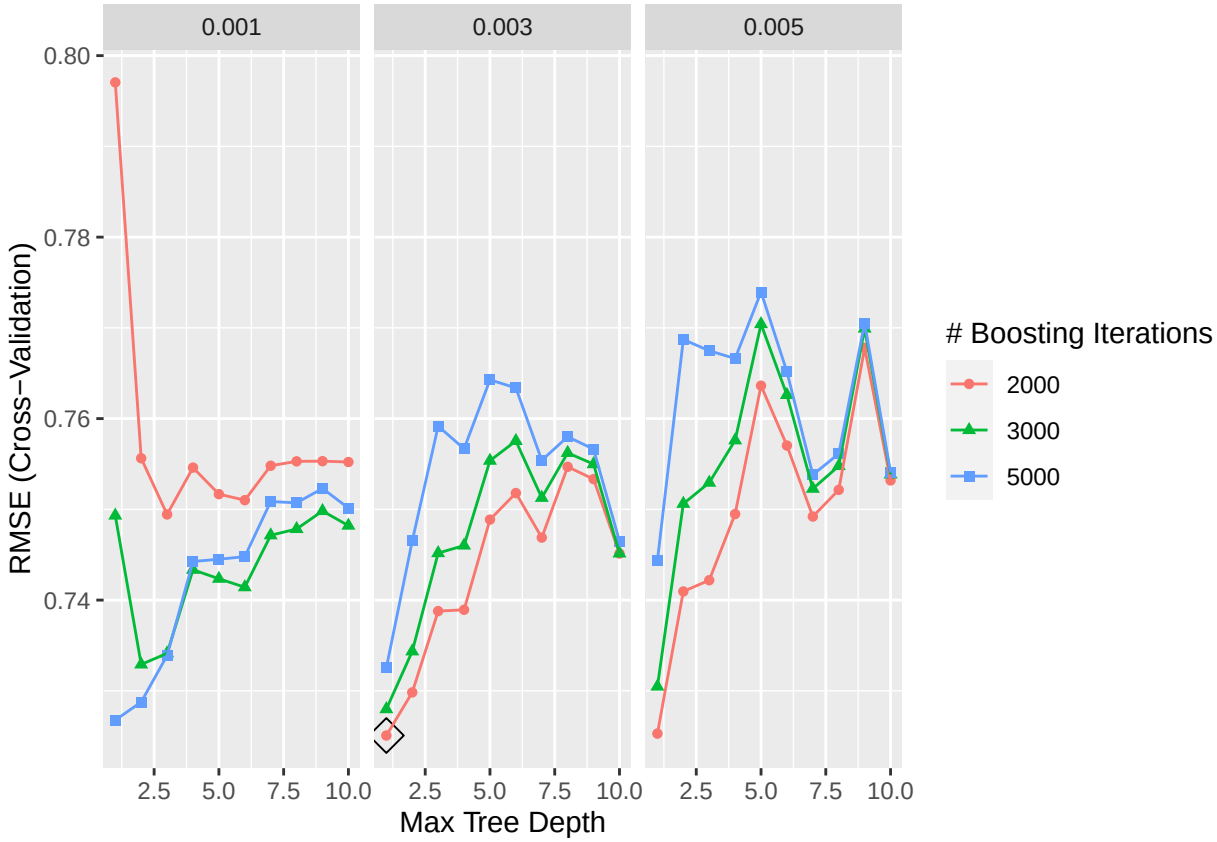

```
barplot(sort(ranger::importance(rf.fit$finalModel), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("darkred","white","darkblue"))(19))
```



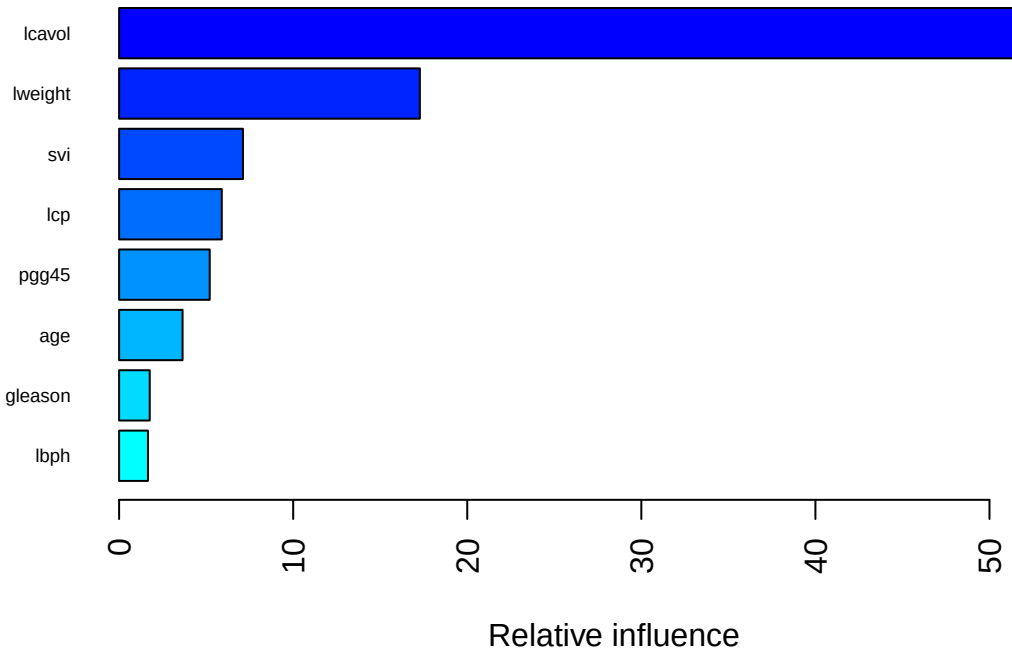

The importance of variables using the random forest method from highest to lowest is: lcavol, lweight, svi, lcp, pgg45, age, lbph, and gleason.

**(e) Perform boosting and report the variable importance.**

```
gbm.grid <- expand.grid(n.trees = c(2000, 3000, 5000),
                        interaction.depth = 1:10,
                        shrinkage = c(0.001,0.003,0.005),
                        n.minobsinnode = 1)

set.seed(2021)
gbm.fit <- train(lpsa~., Prostate,
                 method = "gbm",
                 tuneGrid = gbm.grid,
                 trControl = ctrl,
                 verbose = FALSE)

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




```
summary(gbm.fit$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```




```
##             var   rel.inf
## lcavol   lcavol 57.443112
## lweight lweight 17.277195
## svi         svi  7.113658
## lcp         lcp  5.897296
## pgg45     pgg45  5.205067
## age         age  3.645591
## gleason gleason  1.759958
## lbph       lbph  1.658123
```

The importance of variables from highest to lowest using the GBM method is: lcavol, lweight, svi, lcp, pgg45, age, gleason, and lbph.

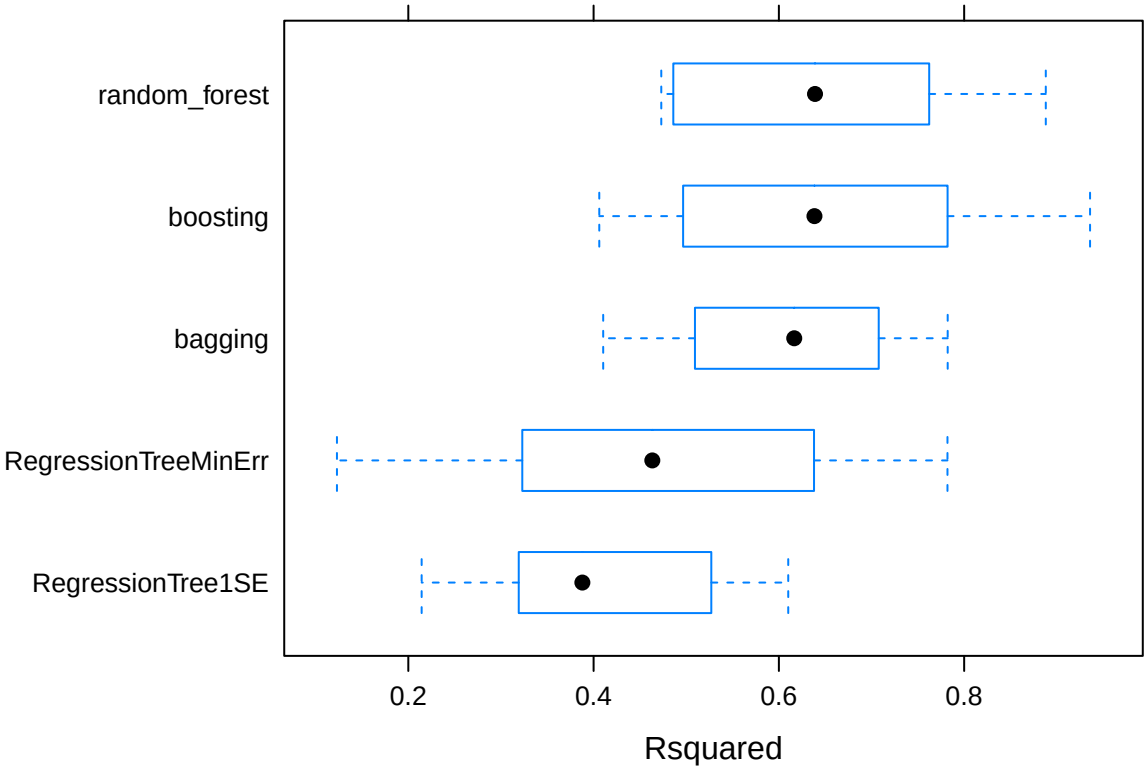
**(f) Which of the above models will you select to predict PSA level?**

```
resample = resamples(
  list(
  RegressionTreeMinErr = tree.fit,
  RegressionTree1SE = tree.fit2,
  bagging = bag.fit,
  random_forest = rf.fit,
  boosting = gbm.fit
  ))
summary(resample)
```

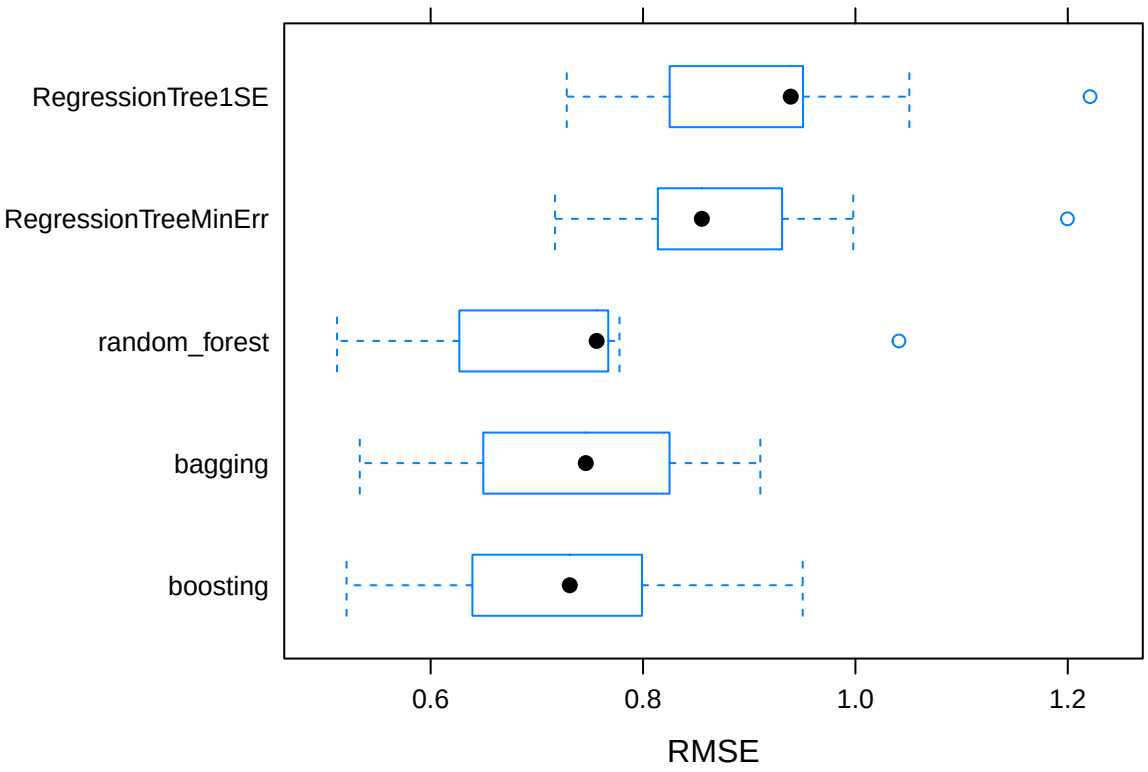
```
##
## Call:
## summary.resamples(object = resample)
##
## Models: RegressionTreeMinErr, RegressionTree1SE, bagging, random_forest, boosting
```

```
## Number of resamples: 10
##
## MAE
##                            Min.   1st Qu.    Median      Mean   3rd Qu.
## RegressionTreeMinErr 0.6109849 0.6583144 0.7600053 0.7431920 0.7795414
## RegressionTree1SE    0.5739719 0.6827443 0.7610476 0.7624917 0.8401602
## bagging              0.4433892 0.5506423 0.6187773 0.6028693 0.6563489
## random_forest        0.4407939 0.5289913 0.5751973 0.6074843 0.6921474
## boosting             0.3898938 0.5230991 0.5788507 0.5909836 0.6721499
##                            Max. NA's
## RegressionTreeMinErr 0.9526082    0
## RegressionTree1SE    0.9892325    0
## bagging              0.7295966    0
## random_forest        0.8532146    0
## boosting             0.7939069    0
##
## RMSE
##                            Min.   1st Qu.    Median      Mean   3rd Qu.
## RegressionTreeMinErr 0.7170878 0.8226949 0.8553933 0.8873890 0.9253618
## RegressionTree1SE    0.7281433 0.8274864 0.9390805 0.9166601 0.9500930
## bagging              0.5332508 0.6602017 0.7460857 0.7410050 0.8218336
## random_forest        0.5117386 0.6532834 0.7563223 0.7325754 0.7671621
## boosting             0.5208195 0.6447519 0.7309991 0.7250765 0.7832713
##                            Max. NA's
## RegressionTreeMinErr 1.1996699    0
## RegressionTree1SE    1.2209824    0
## bagging              0.9104530    0
## random_forest        1.0409883    0
## boosting             0.9503671    0
##
## Rsquared
##                            Min.   1st Qu.    Median      Mean   3rd Qu.
## RegressionTreeMinErr 0.1229707 0.3354216 0.4633757 0.4590799 0.6144145
## RegressionTree1SE    0.2144062 0.3241563 0.3878897 0.4125650 0.5154493
## bagging              0.4103590 0.5274679 0.6165821 0.6181705 0.7020880
## random_forest        0.4730590 0.4944753 0.6390078 0.6387462 0.7542338
## boosting             0.4061373 0.5119963 0.6383912 0.6412627 0.7548605
##                            Max. NA's
## RegressionTreeMinErr 0.7819655    0
## RegressionTree1SE    0.6101062    0
## bagging              0.7821978    0
## random_forest        0.8880868    0
## boosting             0.9358892    0
```

```
bwplot(resample, metric = "Rsquared")
```




```
bwplot(resample, metric = "RMSE")
```




From the boxplot of Rsquared, we can see that random forest has the largest Rsquared, and decision tree using 1 SE rule has the smallest Rsquared. From the boxplot of RMSE, boosting has the smallest mean of

RMSE, and decision tree using 1 SE rule has the largest mean of RMSE. I will choose the boosting method to predict PSA level since it has the smallest RMSE and large Rsquared.

# Problem 2

**(a) Fit a classification tree to the training set, with Purchase as the response and the other variables as predictors. Use cross-validation to determine the tree size and create a plot of the final tree. Predict the response on the test data. What is the test classification error rate?**

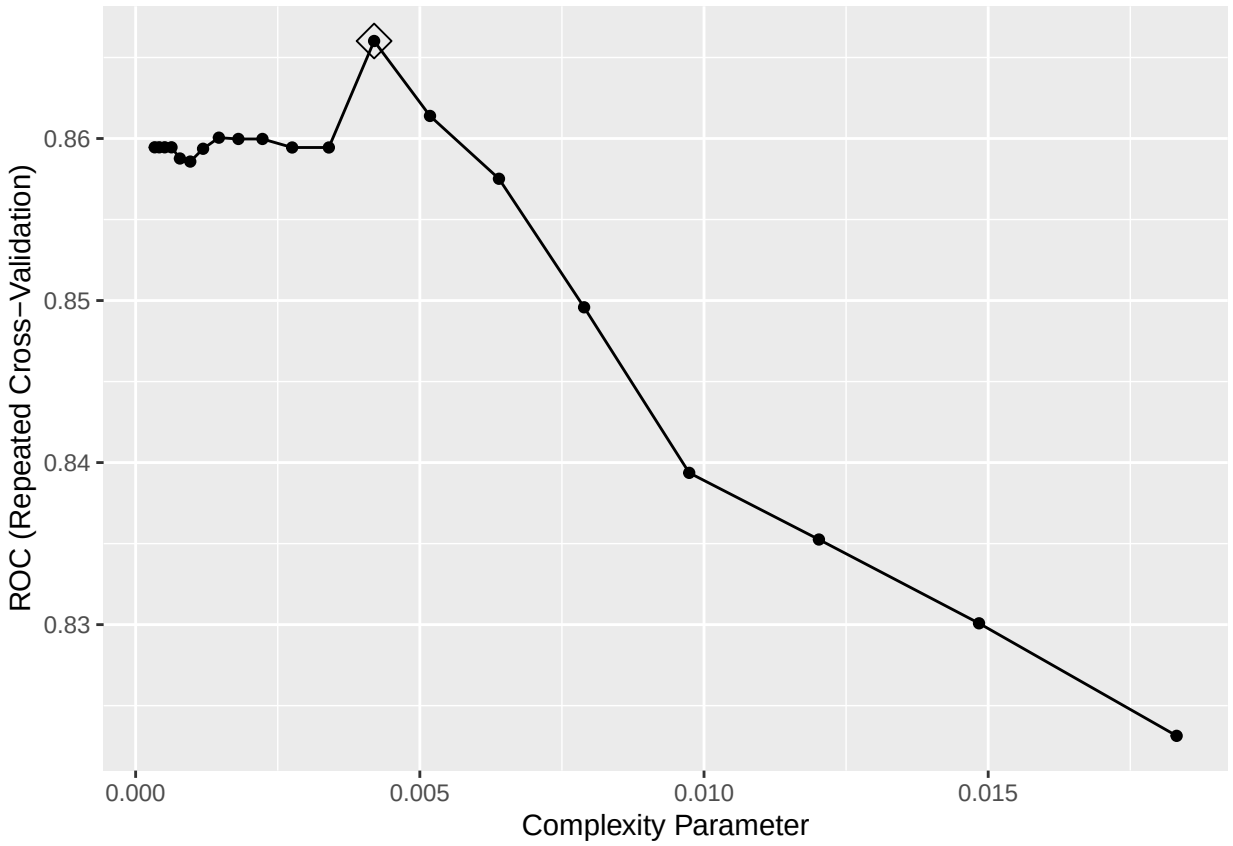
```
data("OJ")
#head(OJ)

set.seed(1)
rowTrain = createDataPartition(y = OJ$Purchase, p = 0.747, list = FALSE)

trainData = OJ[rowTrain, ]
testData = OJ[-rowTrain, ]

dim(trainData)
```

```
## [1] 800  18
```

```
dim(testData)
```

```
## [1] 270  18
```

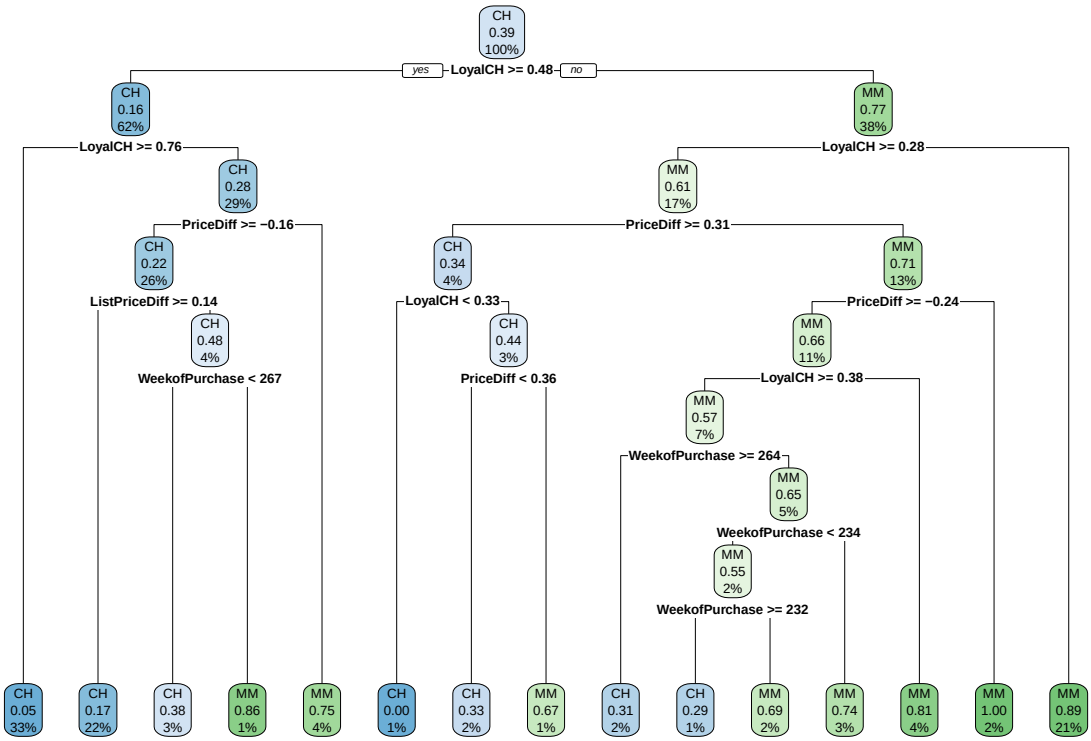
```
set.seed(1)
ctrl2 <- trainControl(method = "repeatedcv", summaryFunction = twoClassSummary, classProbs = TRUE)
rpart.fit_oj <- train(Purchase~.,
                      data = trainData,
                      method = "rpart",
                      tuneGrid = data.frame(cp = exp(seq(-8,-4, len = 20))),
                      trControl = ctrl2,
                      metric = "ROC")

ggplot(rpart.fit_oj, highlight = TRUE)
```




```
rpart.fit_oj$finalModel$cptable
```

```
##            CP nsplit rel error
## 1 0.519230769      0 1.0000000
## 2 0.022435897      1 0.4807692
## 3 0.017628205      3 0.4358974
## 4 0.008012821      5 0.4006410
## 5 0.005341880      7 0.3846154
## 6 0.004807692     10 0.3685897
## 7 0.004195746     14 0.3493590
```

```
rpart.plot(rpart.fit_oj$finalModel)
```



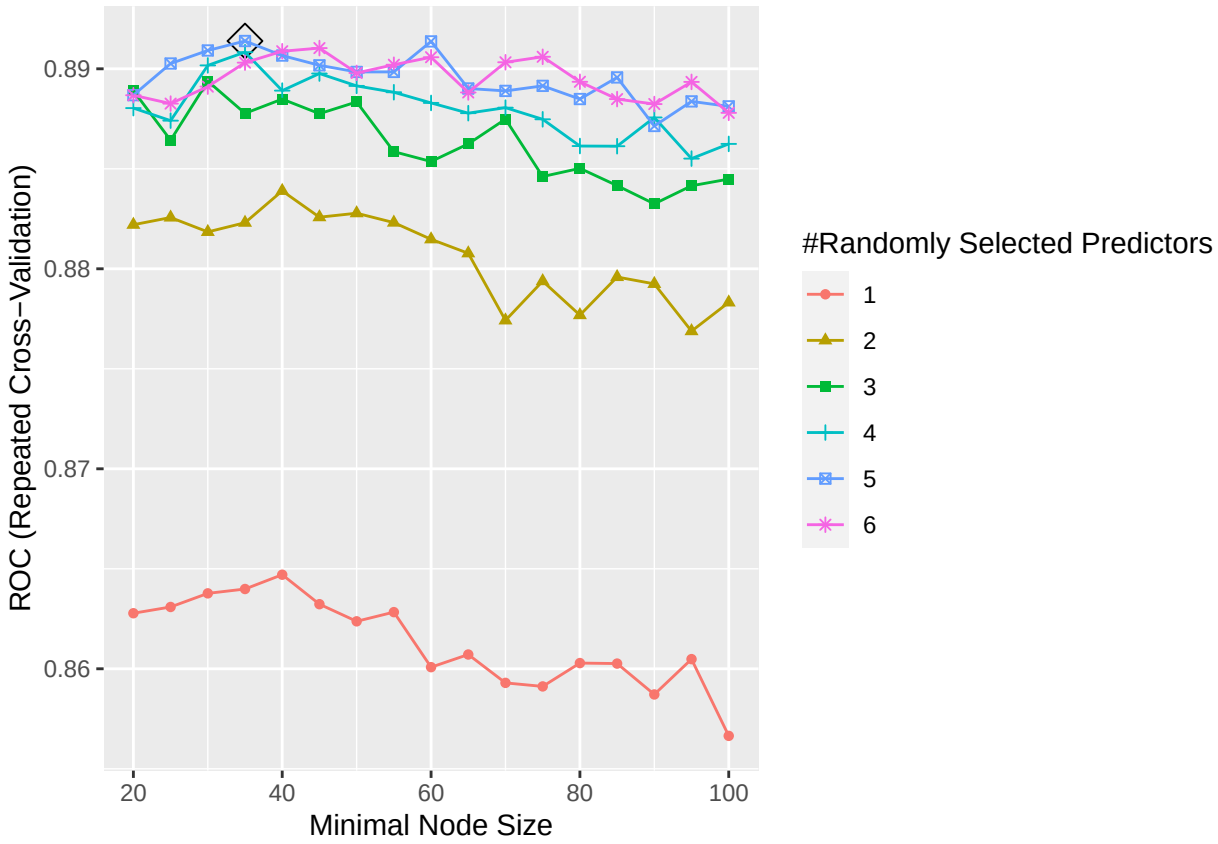

```
rpart.pred <- predict(rpart.fit_oj, newdata = testData)
error.rate <- 1 - mean(testData$Purchase == rpart.pred); error.rate
```

```
## [1] 0.1888889
```

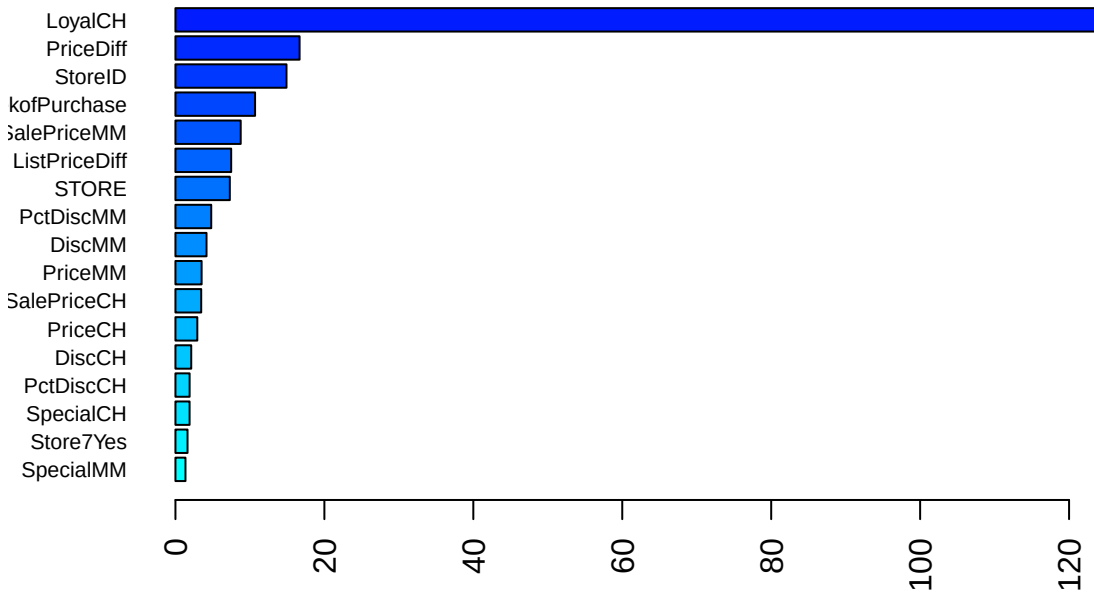
The tree size is 15 and test error rate is 0.1889.

**(b) Perform random forest on the training set and report variable importance. What is the test error rate?**

```
rf.grid2 <- expand.grid(mtry = 1:6,
                        splitrule = "gini",
                        min.node.size = seq(from = 20, to = 100, by = 5))
set.seed(1)
rf.fit_oj <- train(Purchase~.,
                   data = trainData,
                   method = "ranger",
                   tuneGrid = rf.grid2,
                   metric = "ROC",
                   trControl = ctrl2,
                   importance = "impurity")
ggplot(rf.fit_oj,highlight = TRUE)
```




```
barplot(sort(ranger::importance(rf.fit_oj$finalModel),decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(19))
```



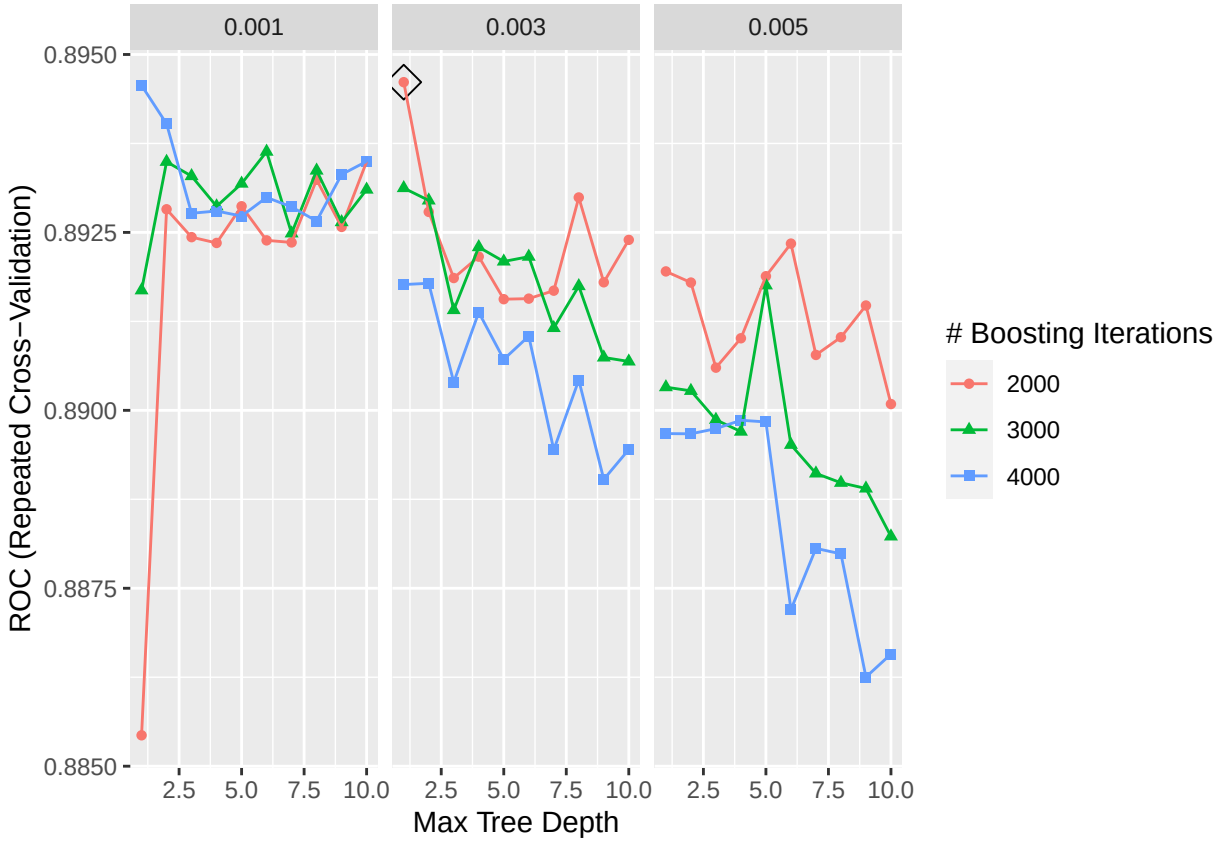

```
rf.pred2 <- predict(rf.fit_oj, newdata = testData, type = "raw")
error.rate2 <- 1 - mean(testData$Purchase == rf.pred2); error.rate2
```

```
## [1] 0.1703704
```

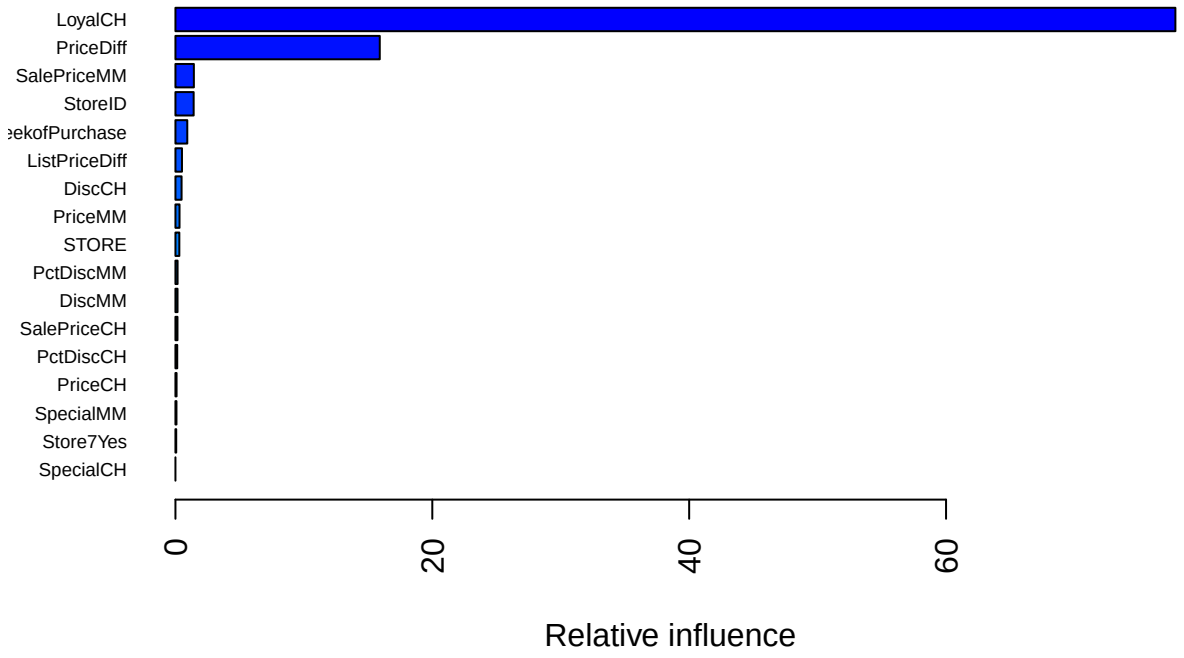
From the importance barplot, we can see that the top 3 most important variables are: LoyalCH > PriceDiff > StoreID. The least important variable is SpecialMM. The test error rate is 0.1704.

**(c) Perform boosting on the training set and report variable importance. What is the test error rate?**

```
boost.grid2 <- expand.grid(n.trees = c(2000,3000,4000),
                           interaction.depth = 1:10,
                           shrinkage = c(0.001,0.003,0.005),
                           n.minobsinnode = 1)

set.seed(1)
boost.fit_oj <- train(Purchase~.,
                  data = trainData,
                  tuneGrid = boost.grid2,
                  trControl = ctrl2,
                  method = "gbm",
                  distribution = "adaboost",
                  metric = "ROC",
                  verbose = FALSE)

ggplot(boost.fit_oj, highlight = TRUE)
```



```
summary(boost.fit_oj$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```




```
##                            var     rel.inf
## LoyalCH                LoyalCH 77.85723956
## PriceDiff            PriceDiff 15.90900877
## SalePriceMM        SalePriceMM  1.43433231
## StoreID                StoreID  1.41720661
## WeekofPurchase WeekofPurchase  0.92112737
## ListPriceDiff    ListPriceDiff  0.50861065
## DiscCH                  DiscCH  0.47741430
## PriceMM                PriceMM  0.31806576
## STORE                    STORE  0.30393196
## PctDiscMM            PctDiscMM  0.16651605
## DiscMM                  DiscMM  0.16115561
## SalePriceCH        SalePriceCH  0.15846947
## PctDiscCH            PctDiscCH  0.13941927
## PriceCH                PriceCH  0.08793492
## SpecialMM            SpecialMM  0.07901150
## Store7Yes            Store7Yes  0.06055589
## SpecialCH            SpecialCH  0.00000000
```

```
boost.pred2 <- predict(boost.fit_oj, newdata = testData)
error.rate3 <- 1 - mean(testData$Purchase == boost.pred2); error.rate3
```

```
## [1] 0.1851852
```

From the importance barplot, we can see that the top 3 most important variables are: LoyalCH >PriceDiff > SalePriceMM. The least important variable is SpecialCH. The test error rate is 0.1852.