P8106 HW4

Lin Yang

```
library(tidyverse)
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
library(ranger)
library(gbm)
library(ISLR)
```

Problem 1

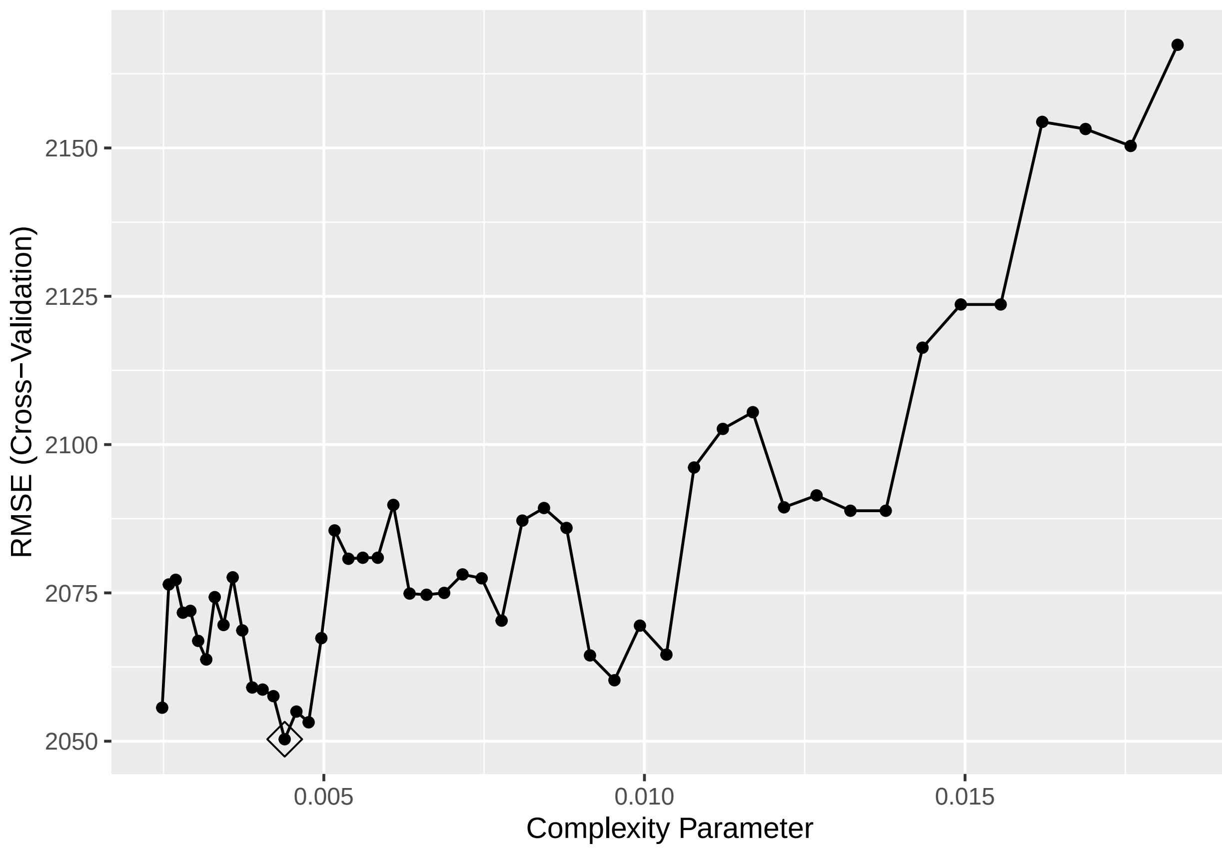
```
College <- read.csv("data/College.csv") %>%
  janitor::clean_names() %>%
  select(-1)

set.seed(2022)
trainRows <- createDataPartition(y = College$outstate, p = 0.8, list = FALSE)
College_train <- College[trainRows, ]
College_test <- College[-trainRows, ]
```

a. Regression Tree

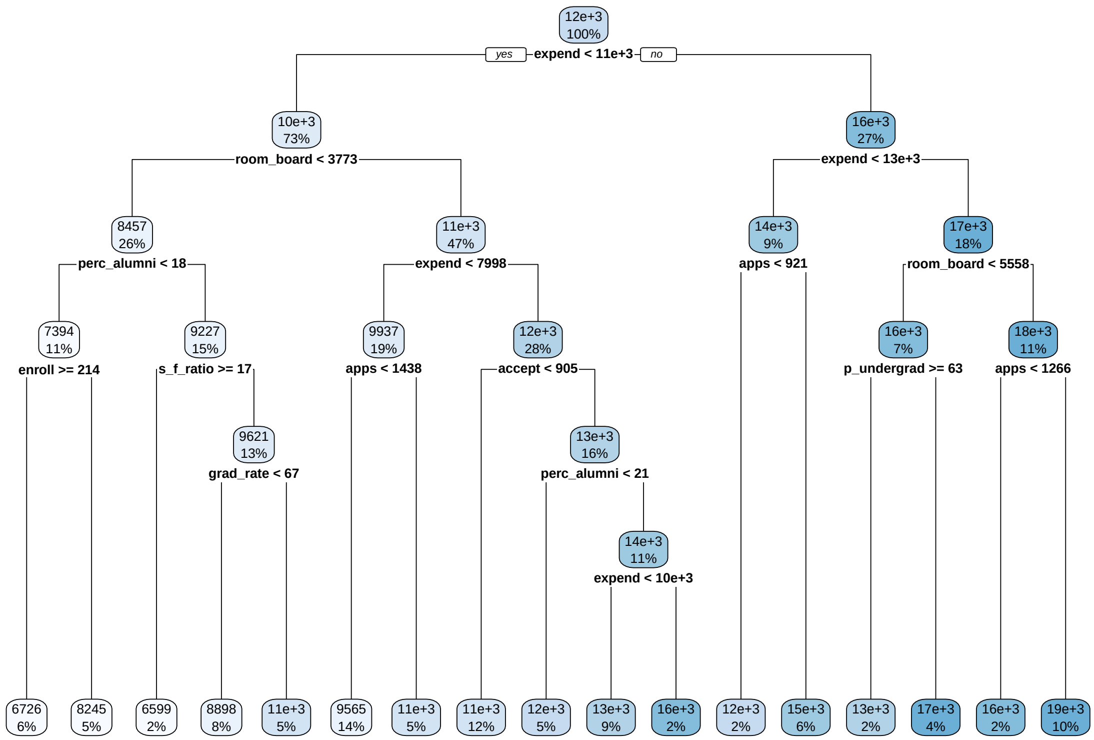
```
ctrl <- trainControl(method = "cv")
set.seed(2022)
r.tree <- train(outstate ~ . ,
                College_train,
                method = "rpart",
                tuneGrid = data.frame(cp = exp(seq(-6,-4, length = 50))),
                trControl = ctrl)
r.tree$bestTune
```

```
##             cp
## 15 0.004389362
```

```
ggplot(r.tree, highlight = TRUE)
```




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



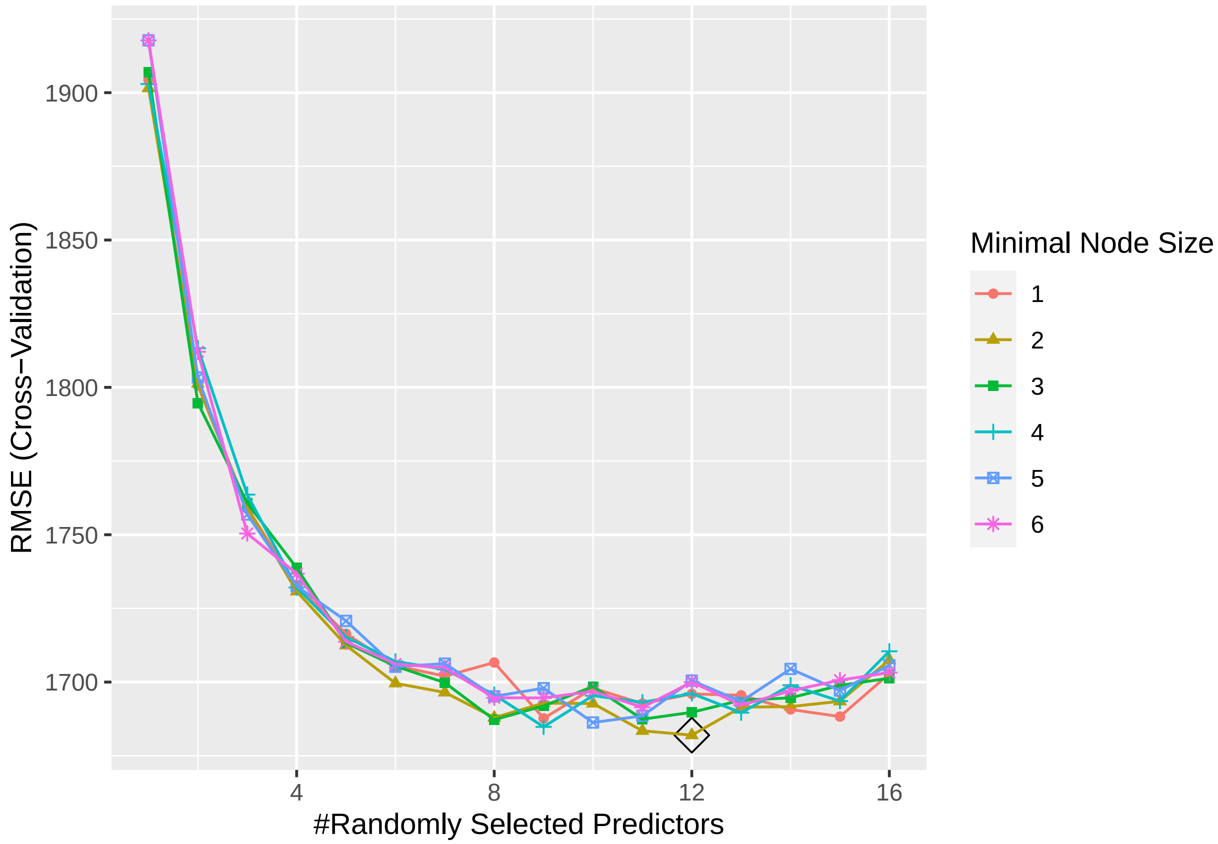
The best cp is selected to be 0.00438936184277844. The root node is `expend` less than 11000 or not. There are 17 terminal nodes, thus this is a large tree.

b. Random Forest

```
rf.grid <- expand.grid(mtry = 1:16,
                       splitrule = "variance",
                       min.node.size = 1:6)
set.seed(2022)
rf.fit <- train(outstate ~ . ,
                College_train,
                method = "ranger",
                tuneGrid = rf.grid,
                trControl = ctrl)
rf.fit$bestTune
```

```
##    mtry splitrule min.node.size
## 68   12  variance             2
```

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



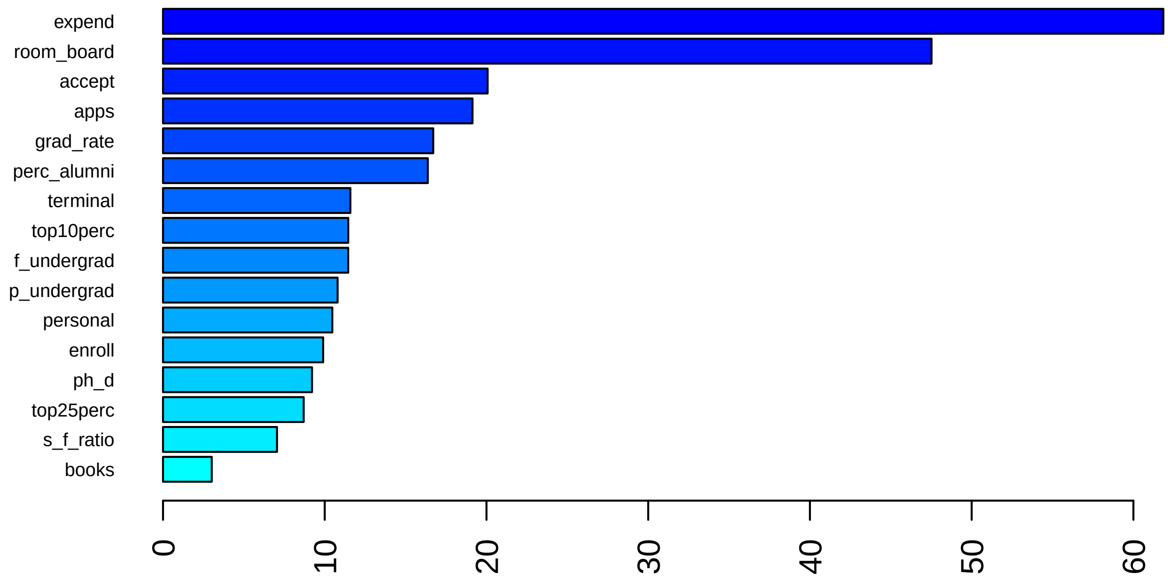

```
pred.rf <- predict(rf.fit, newdata = College_test)
te.rf <- RMSE(pred.rf, College_test$outstate)
te.rf
```

```
## [1] 1980.006
```

The best tuning parameters are found to be m = 12 and minimum node size = 2. The RMSE based on test data is 1980.0060684.

We then plotted a variable importance plot based on permutation importance. The most important variables are found to be `expend` and `room_board`. `accept`, `apps`, `grad_rate`, and `perc_alumni` are relatively important.

```
set.seed(2022)
rf.per <- ranger(outstate ~ . ,
                 College_train,
                 mtry = rf.fit$bestTune[[1]],
                 splitrule = "variance",
                 min.node.size = rf.fit$bestTune[[3]],
                 importance = "permutation",
                 scale.permutation.importance = TRUE)

barplot(sort(ranger::importance(rf.per), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.6,
        col = colorRampPalette(colors = c("cyan","blue"))(16))
```

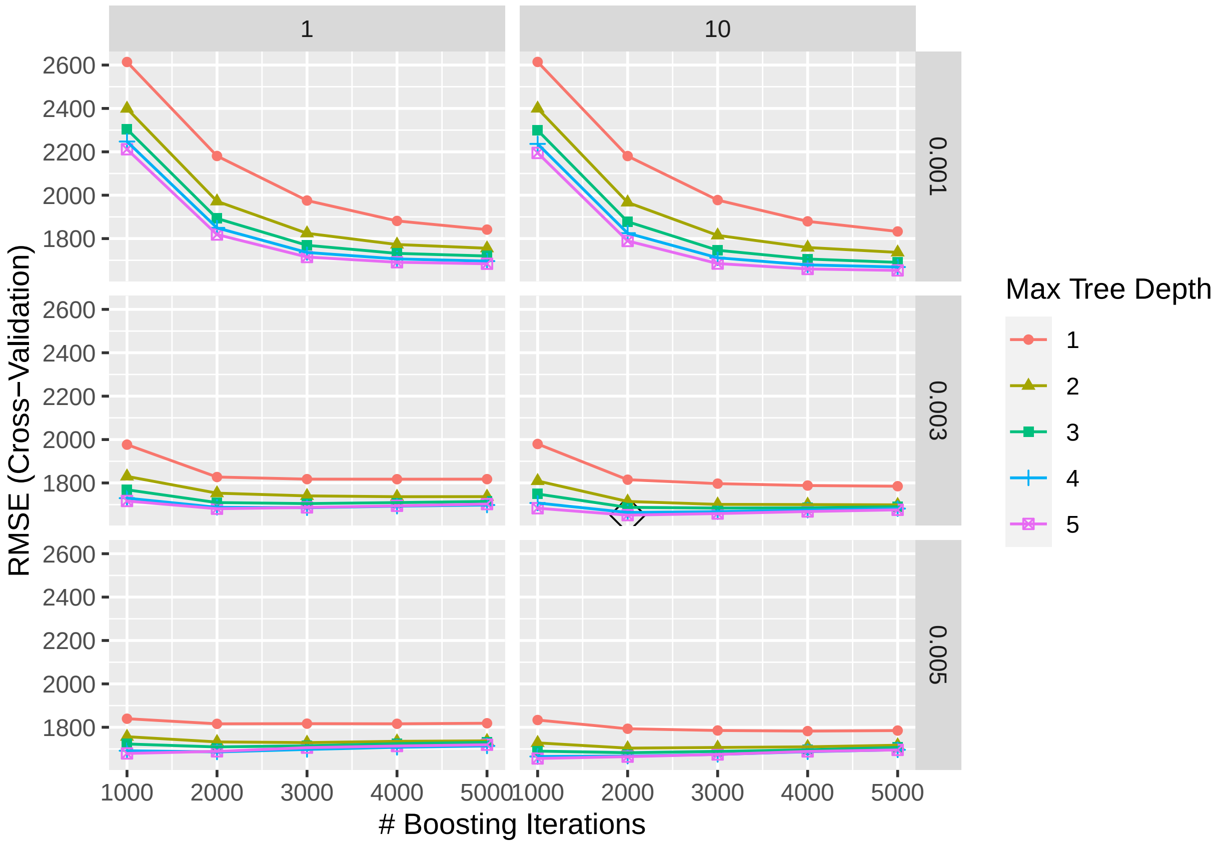


c. Boosting

```
gbm.grid <- expand.grid(n.trees = c(1000,2000,3000,4000,5000),
                        interaction.depth = 1:5,
                        shrinkage = c(0.001,0.003,0.005),
                        n.minobsinnode = c(1,10))

set.seed(8106)
gbm.fit <- train(outstate ~ . ,
                 College_train,
                 method = "gbm",
                 tuneGrid = gbm.grid,
                 trControl = ctrl,
                 verbose = FALSE)

gbm.fit$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 97    2000                 5     0.003             10
```

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



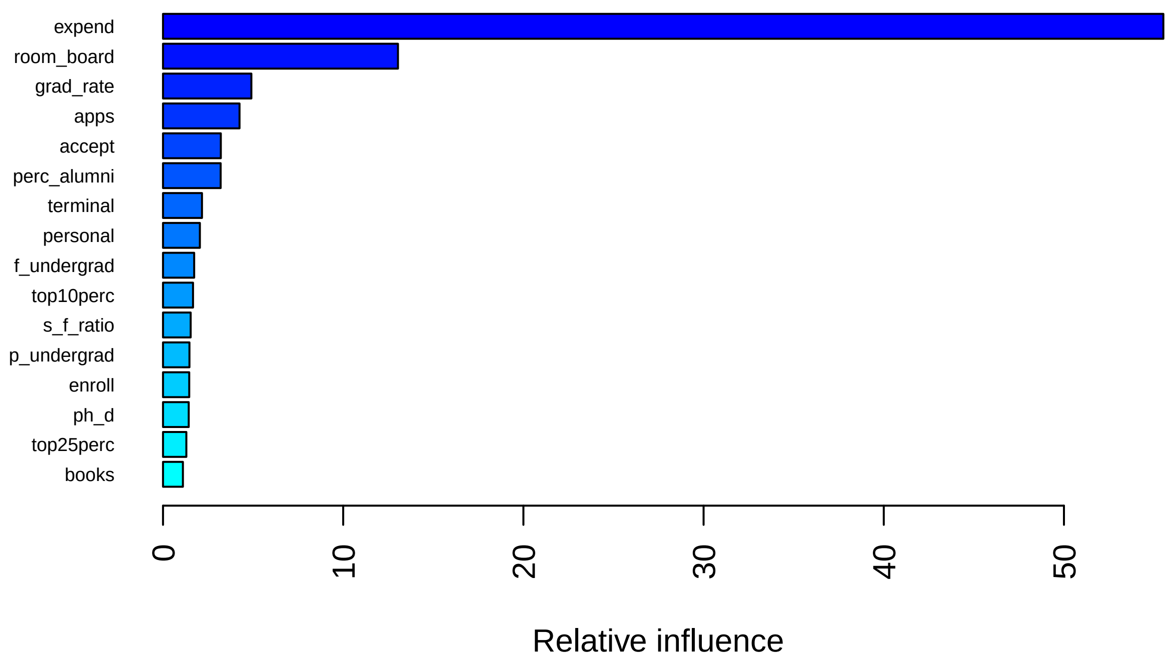

```
pred.bst <- predict(gbm.fit, newdata = College_test)
te.bst <- RMSE(pred.bst, College_test$outstate)
te.bst
```

```
## [1] 1897.383
```

The best tuning parameters of boosting are number of trees = 2000, number of splits = 5, shrinkage = 0.003, and minimum node size = 10. The RMSE based on test data is 1897.38325.

The variable importance plot shows that `expend`, `room_board`, `grad_rate`, and `apps` are important variables, which are similar to the results of random forest.

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




```
##                     var   rel.inf
## expend           expend 55.515875
## room_board   room_board 13.039251
## grad_rate     grad_rate  4.903971
## apps               apps  4.246584
## accept           accept  3.208611
## perc_alumni perc_alumni  3.198723
## terminal       terminal  2.163471
## personal       personal  2.046347
## f_undergrad f_undergrad  1.728995
## top10perc     top10perc  1.664884
## s_f_ratio     s_f_ratio  1.534733
## p_undergrad p_undergrad  1.466744
## enroll           enroll  1.454913
## ph_d               ph_d  1.427526
## top25perc     top25perc  1.295551
## books             books  1.103823
```

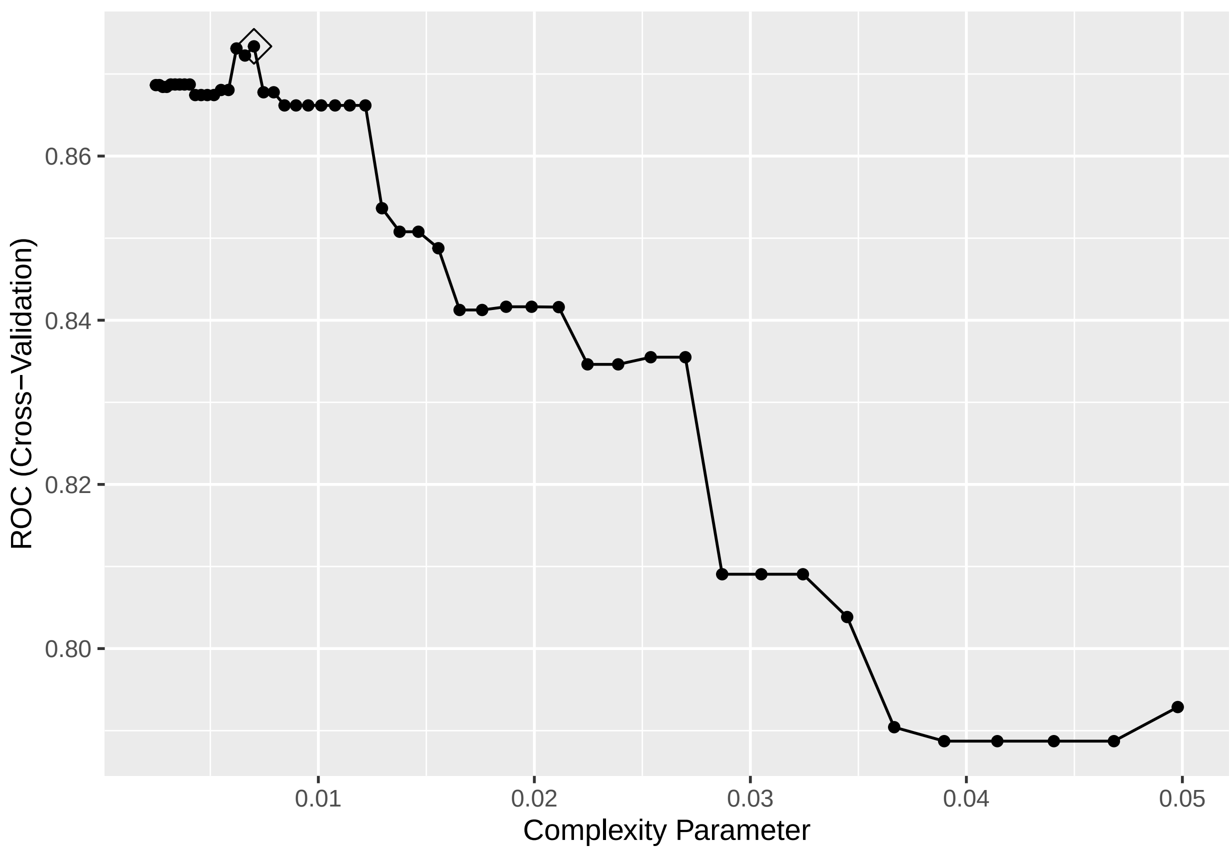
Problem 2

```
data(OJ)
oj <- OJ %>%
  janitor::clean_names() %>%
  na.omit()
```

```
set.seed(8106)
trainRows2 <- createDataPartition(y = oj$purchase, p = 0.653, list = FALSE)
oj_train <- oj[trainRows2, ]
oj_test <- oj[-trainRows2, ]
```

a. Classification Tree

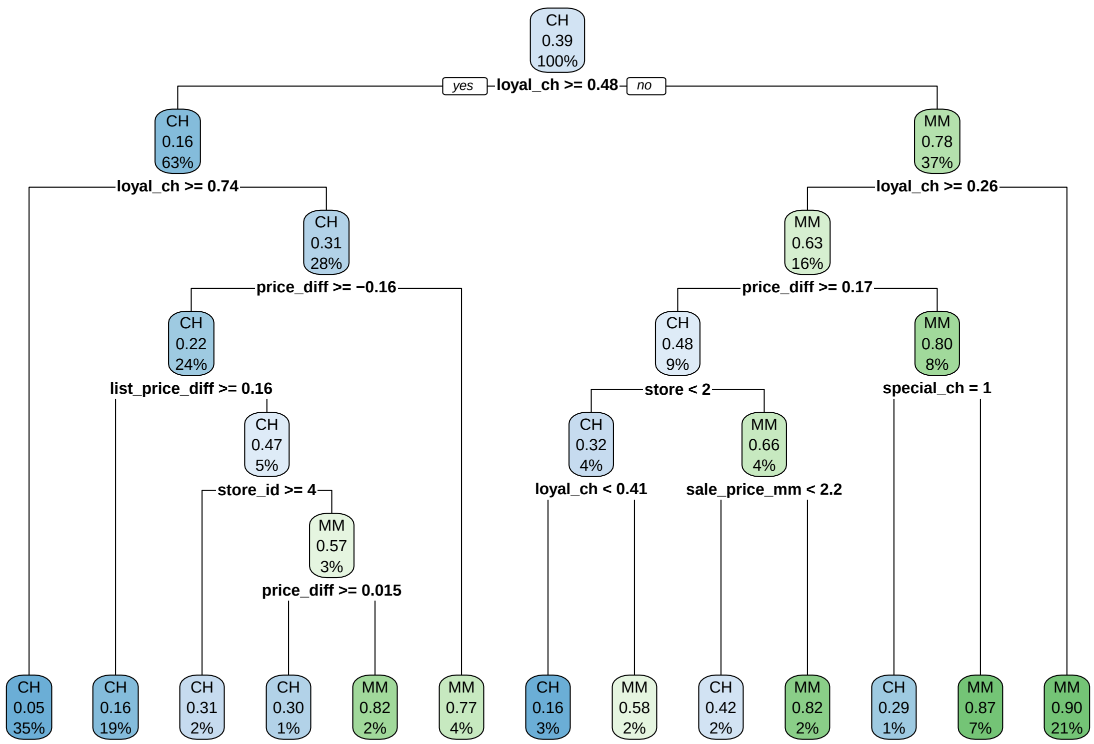
```
ctrl1 <- trainControl(method = "cv",
                      classProbs = TRUE,
                      summaryFunction = twoClassSummary)

set.seed(8106)
c.tree <- train(purchase ~ . ,
                oj_train,
                method = "rpart",
                tuneGrid = data.frame(cp = exp(seq(-6, -3, len = 50))),
                trControl = ctrl1,
                metric = "ROC")

ggplot(c.tree, highlight = TRUE)
```



```
c.tree$bestTune
```

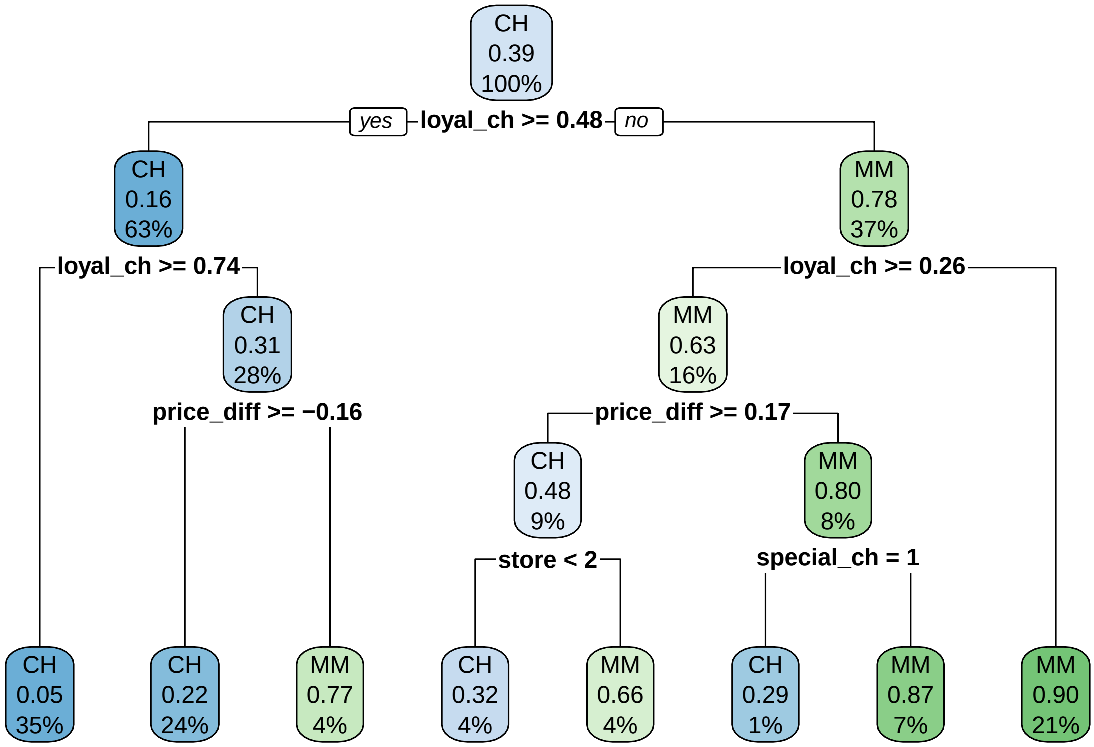
```
##              cp
## 18 0.007018655
```

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



The best cp is found to be 0.00701865500893739. The tree with highest AUC has 13 terminal nodes. However, the tree obtained using 1SE rule has 8 terminal nodes, which is a much smaller tree.

```
#1SE rule
set.seed(8106)
c.tree1 <- rpart(formula = purchase ~ . ,
                 data = oj_train,
                 control = rpart.control(cp = 0))

cpTable <- c.tree1$cptable
minErr <- which.min(cpTable[,4])

c.tree2 <- prune(c.tree1, cp = cpTable[cpTable[,4] < cpTable[minErr,4] + cpTable[minErr,5],1][1])
rpart.plot(c.tree2)
```

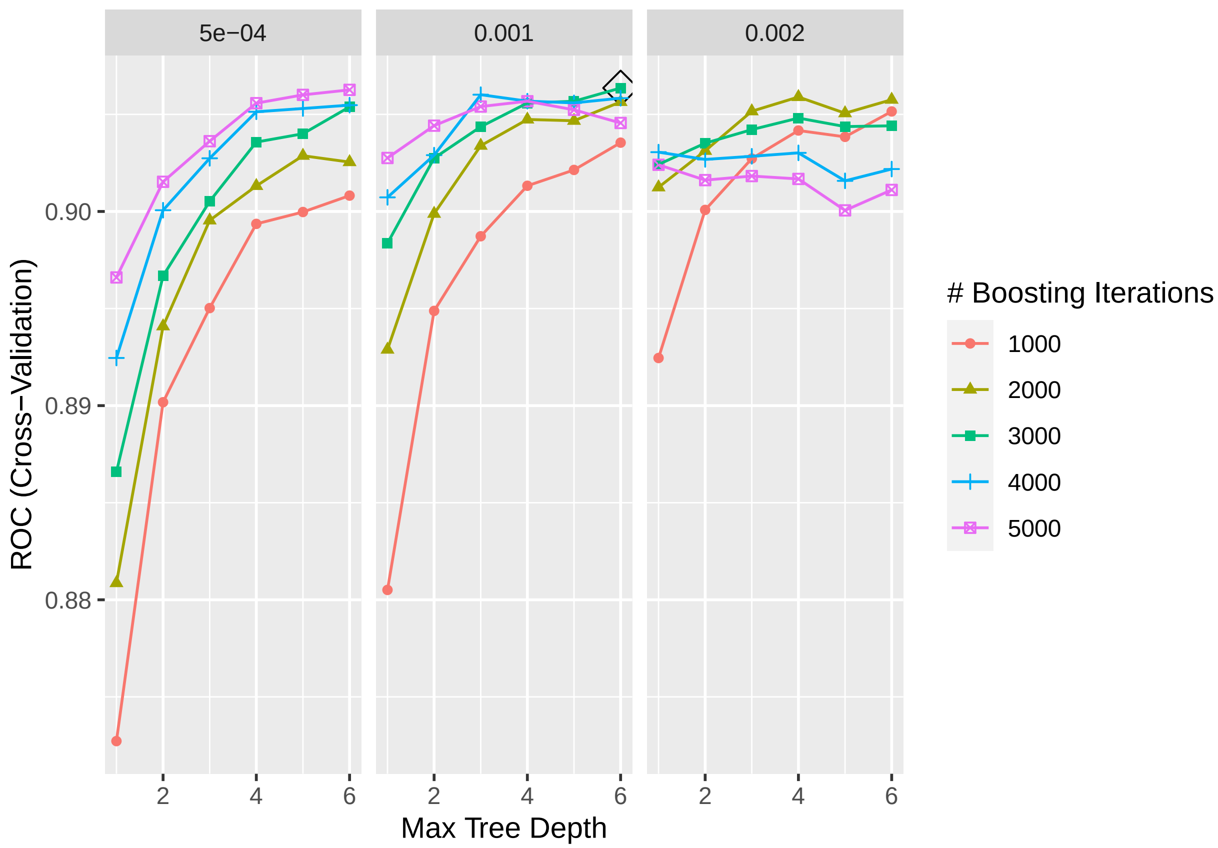


b. AdaBoost

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

gbmA.grid <- expand.grid(n.trees = c(1000,2000,3000,4000,5000),
                         interaction.depth = 1:6,
                         shrinkage = c(0.0005,0.001,0.002),
                         n.minobsinnode = 1)
set.seed(8106)
gbmA.fit <- train(purchase ~ . ,
                  oj_train,
                  tuneGrid = gbmA.grid,
                  trControl = ctrl1,
                  method = "gbm",
                  distribution = "adaboost",
                  metric = "ROC",
                  verbose = FALSE)

gbmA.fit$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 58    3000                 6     0.001              1
```

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




```r
gbmA.pred.class <- predict(gbmA.fit, newdata = oj_test)
confusionMatrix(gbmA.pred.class, oj_test$purchase)
```

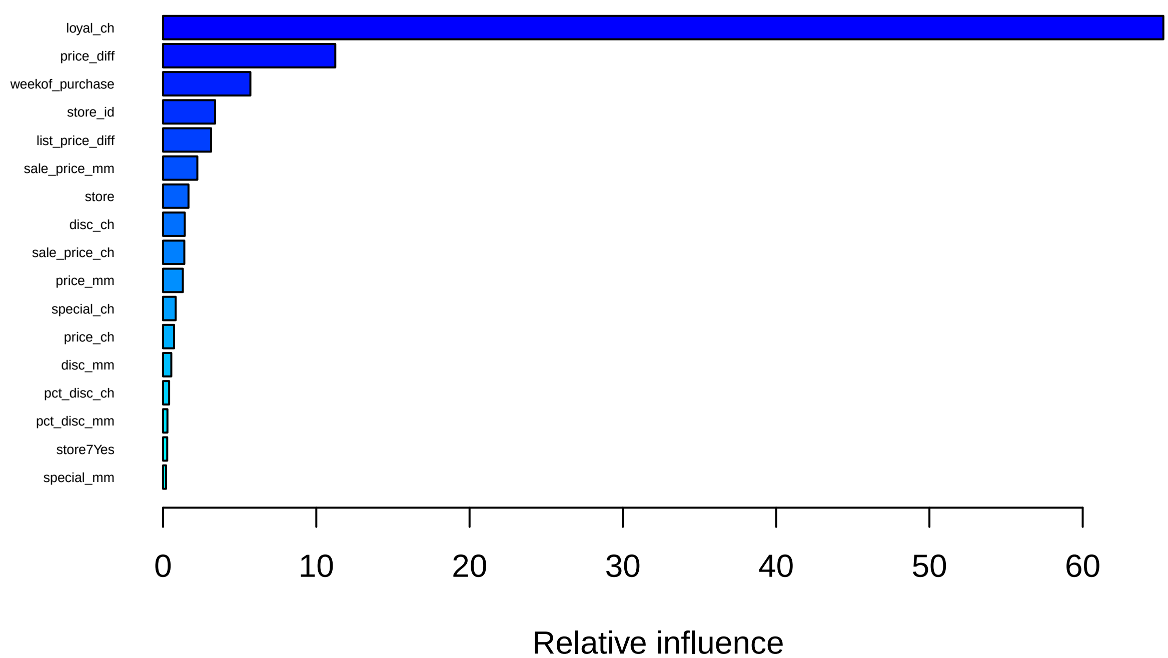
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 192  34
##         MM  34 110
##
##                Accuracy : 0.8162
##                  95% CI : (0.7729, 0.8544)
##     No Information Rate : 0.6108
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6134
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8496
##             Specificity : 0.7639
##          Pos Pred Value : 0.8496
##          Neg Pred Value : 0.7639
```

```
##               Prevalence : 0.6108
##           Detection Rate : 0.5189
##     Detection Prevalence : 0.6108
##        Balanced Accuracy : 0.8067
##
##         'Positive' Class : CH
##
```

The best tuning parameters of Adaboost are number of trees = 3000, number of splits = 6, shrinkage = 0.001, and minimum node size = 1. According to the confusion matrix, the prediction accuracy on test data is 0.8162, thus the test error rate is 0.1838.

The variable importance plot shows that `loyal_ch`, `price_diff`, and `weekof_purchase` are important variables in the Adaboost method.

```
summary(gbmA.fit$finalModel, las = 1.5, cBars = 17, cex.names = 0.45)
```




```
##                             var    rel.inf
## loyal_ch               loyal_ch 65.2617366
## price_diff           price_diff 11.2391291
## weekof_purchase weekof_purchase  5.7004700
## store_id               store_id  3.3991959
## list_price_diff list_price_diff  3.1378176
## sale_price_mm     sale_price_mm  2.2405522
## store                     store  1.6613080
```

```
## disc_ch                 disc_ch  1.4189926
## sale_price_ch     sale_price_ch  1.3902048
## price_mm               price_mm  1.2925784
## special_ch           special_ch  0.8270252
## price_ch               price_ch  0.7242783
## disc_mm                 disc_mm  0.5410329
## pct_disc_ch         pct_disc_ch  0.4009617
## pct_disc_mm         pct_disc_mm  0.2886543
## store7Yes             store7Yes  0.2754971
## special_mm           special_mm  0.2005654
```