

# Machine Learning Hw4

*Ekta Chaudhary*

*20/04/2020*

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

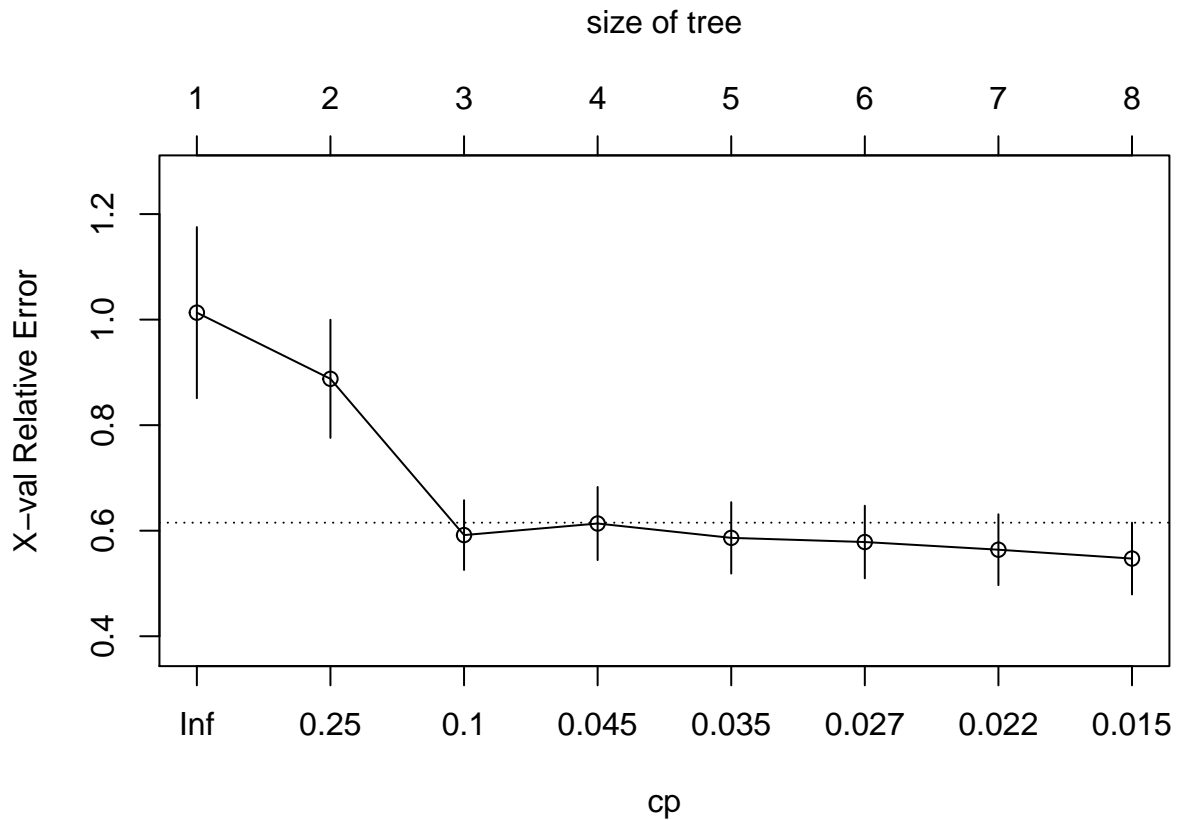
- (a) Fit a regression tree with lpsa as the response and the other variables as predictors. Use cross-validation to determine the optimal tree size. Which tree size corresponds to the lowest cross-validation error? Is this the same as the tree size obtained using the 1 SE rule?

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

```
set.seed(1)
tree <- rpart(formula = lpsa ~ ., data = Prostate,
              control = rpart.control(cp = 0.01))
cpTable <- printcp(tree)
```

```
##
## Regression tree:
## rpart(formula = lpsa ~ ., data = Prostate, control = rpart.control(cp = 0.01))
##
## Variables actually used in tree construction:
## [1] lcavol lweight pgg45
##
## Root node error: 127.92/97 = 1.3187
##
## n= 97
##
##      CP nsplit rel error  xerror   xstd
## 1 0.347108     0  1.00000 1.01323 0.162162
## 2 0.184647     1  0.65289 0.88779 0.111915
## 3 0.059316     2  0.46824 0.59168 0.066102
## 4 0.034756     3  0.40893 0.61359 0.069269
## 5 0.034609     4  0.37417 0.58640 0.067630
## 6 0.021564     5  0.33956 0.57853 0.068772
## 7 0.021470     6  0.31800 0.56398 0.067155
## 8 0.010000     7  0.29653 0.54721 0.068034
```

```
plotcp(tree)
```



```
minErr <- which.min(cpTable[,4])
minErr
```

```
## 8
## 8
```

The tree size 8 corresponds to the lowest cross-validation error.

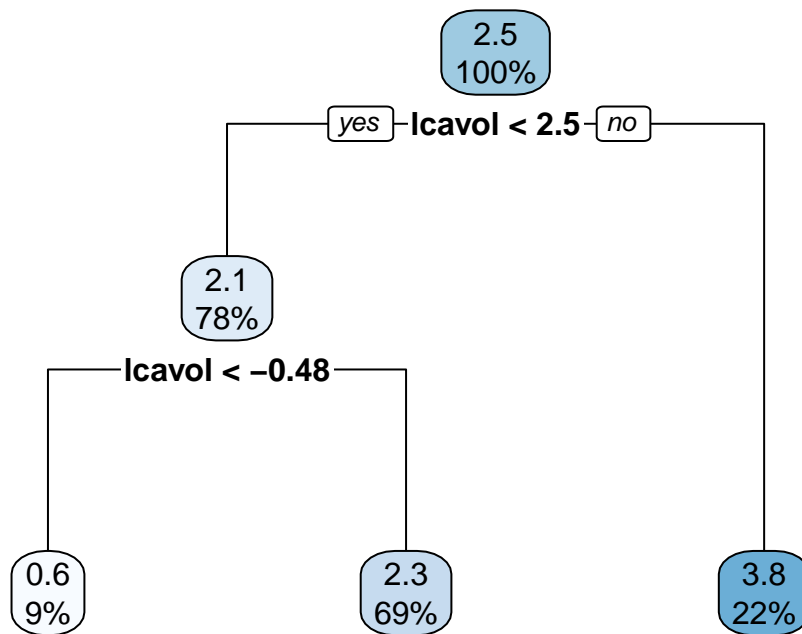
```
cpTable[cpTable[,4] < cpTable[minErr,4] + cpTable[minErr,5],1][1]
```

```
##      3
## 0.05931585
```

The tree size obtained using the 1 SE rule is 3.

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

```
tree_a = prune(tree, cp = cpTable[cpTable[,4] < cpTable[minErr,4] + cpTable[minErr,5], 1][1])
rpart.plot(tree_a)
```



(c) Perform bagging and report the variable importance

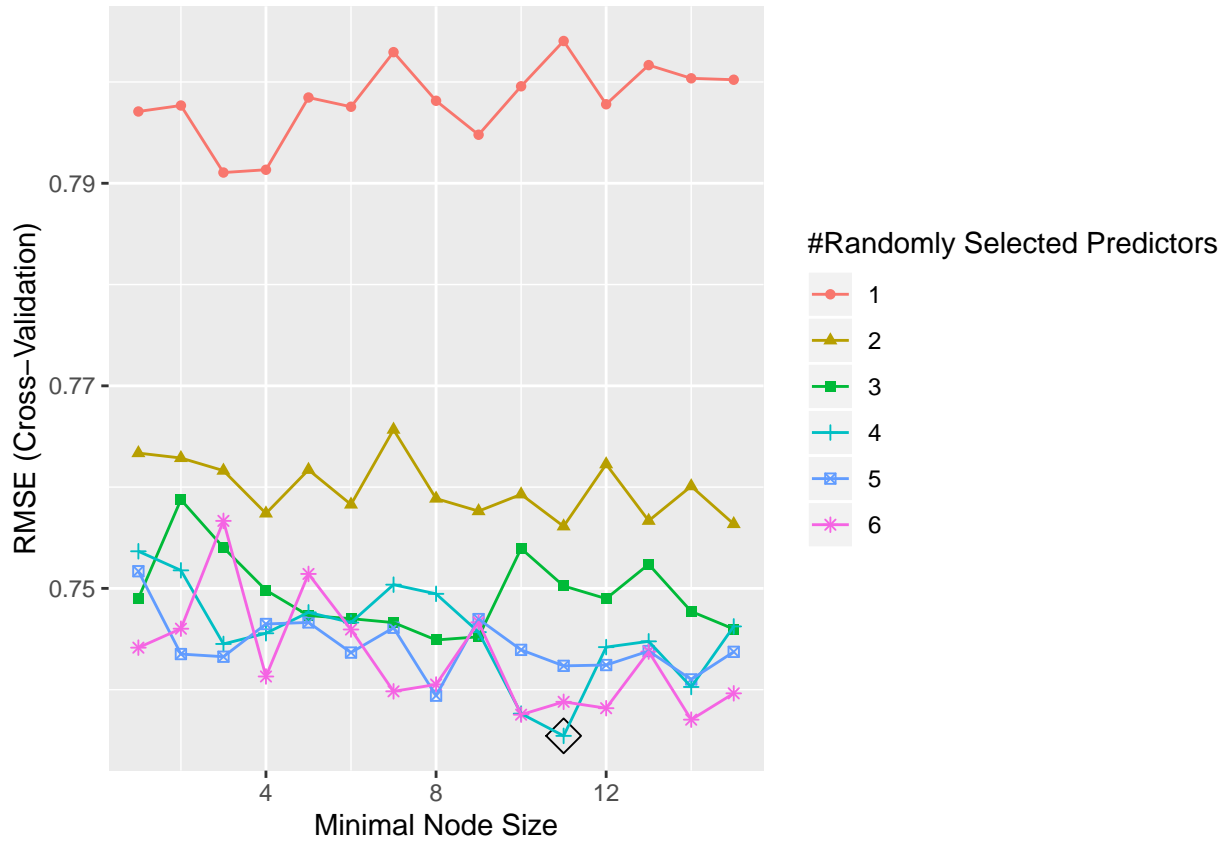
```

bagging.grid <- expand.grid(mtry = 1:6,
                           splitrule = "variance",
                           min.node.size = 1:15)

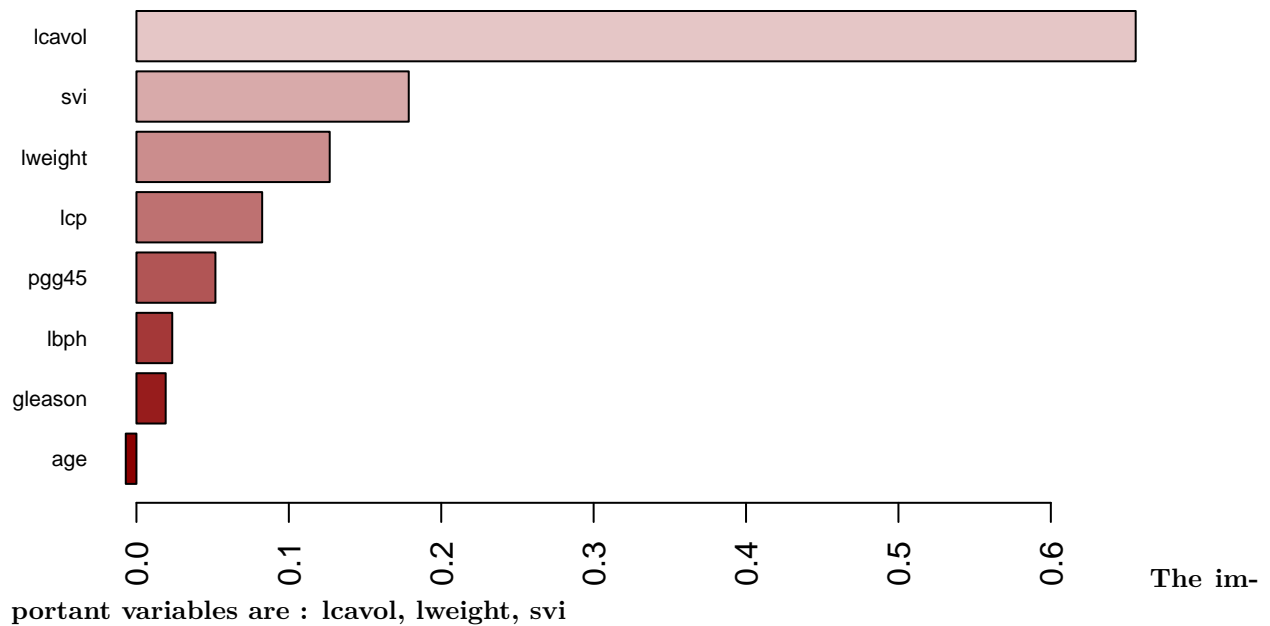
set.seed(1)
bagging <- train(lpsa~., Prostate,
                 method = "ranger",
                 tuneGrid = bagging.grid,
                 trControl = ctrl,
                 importance = "permutation")

ggplot(bagging, highlight = TRUE)

```



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



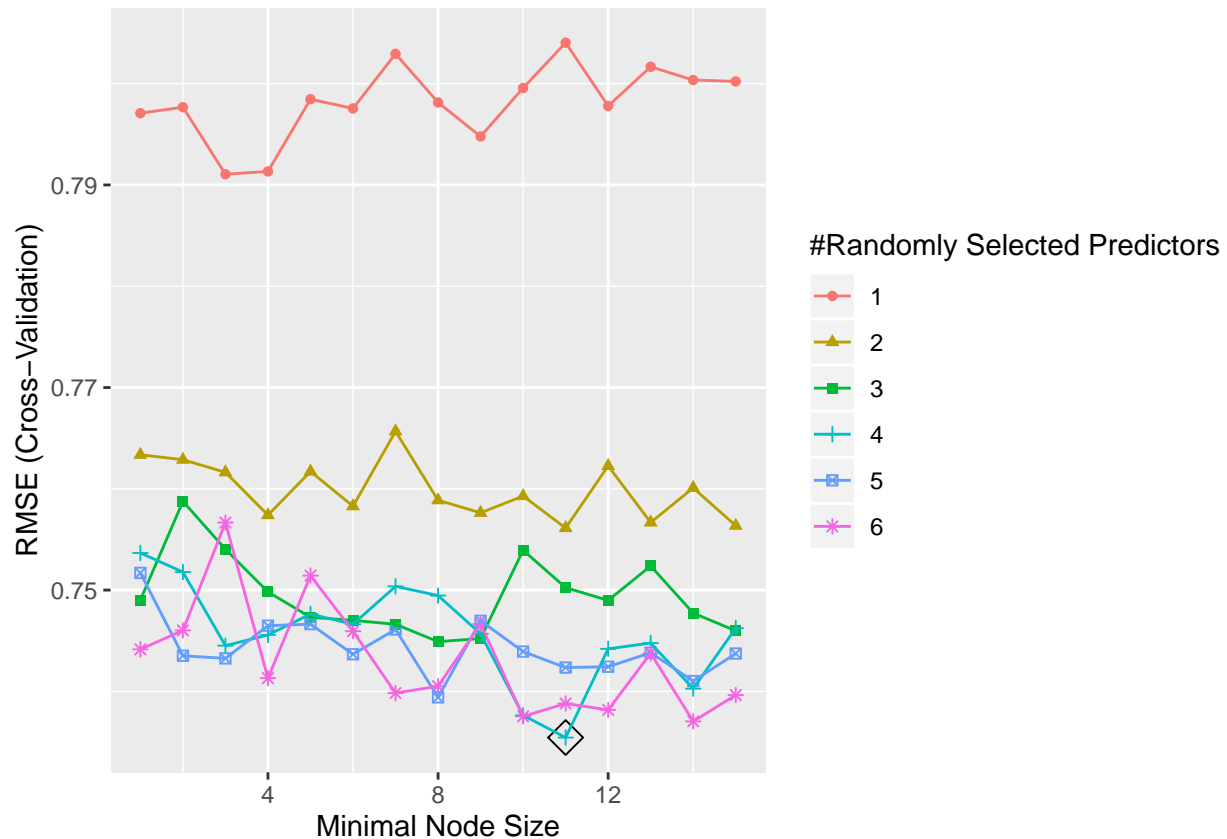
(d) Perform random forests and report the variable importance.

```

rf.grid = expand.grid(mtry = 1:6,
                     splitrule = "variance",
                     min.node.size = 1:15)

set.seed(1)
rf.fit = train(lpsa~., Prostate,
               method = "ranger",
               tuneGrid = rf.grid,
               trControl = ctrl,
               importance = 'permutation')
ggplot(rf.fit, highlight = TRUE)

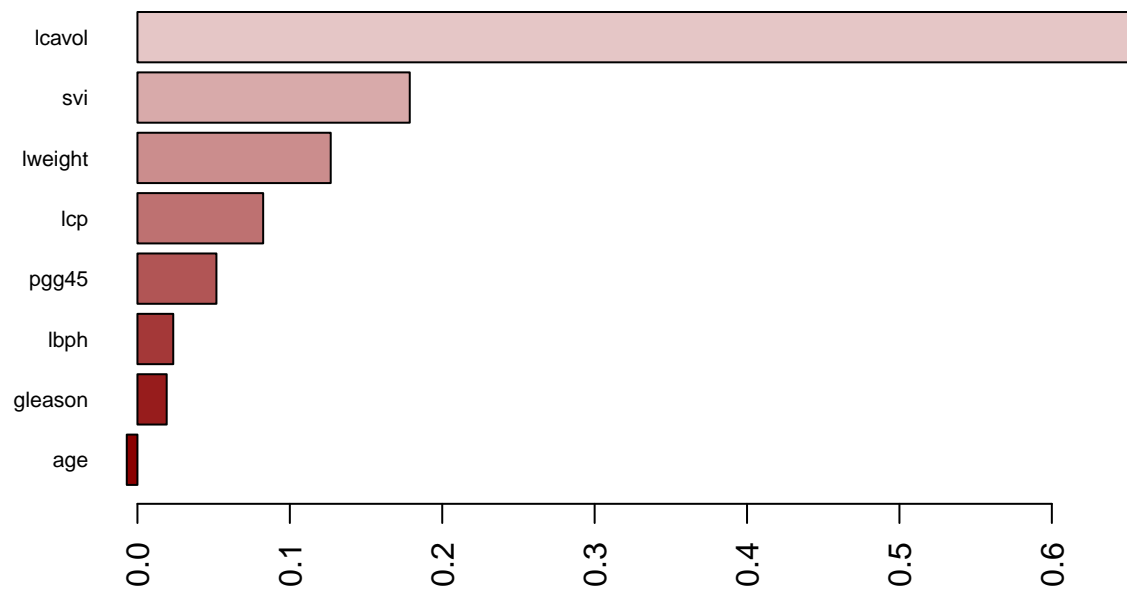
```



```

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))

```



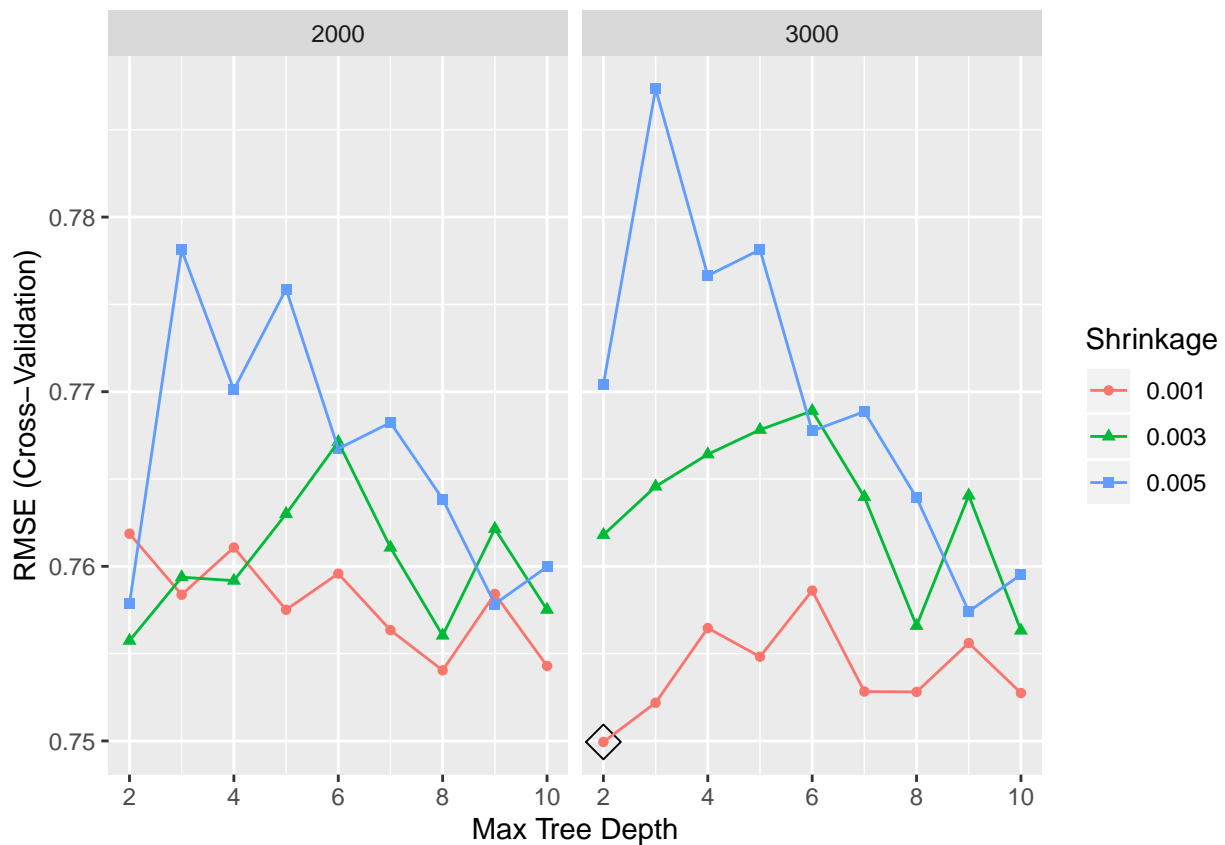
The important variables are : lcavol, svi, lweight

(e) Perform boosting and report the variable importance.

```
gbm.grid <- expand.grid(n.trees = c(2000,3000),
                      interaction.depth = 2:10,
                      shrinkage = c(0.001,0.003,0.005),
                      n.minobsinnode = 1)

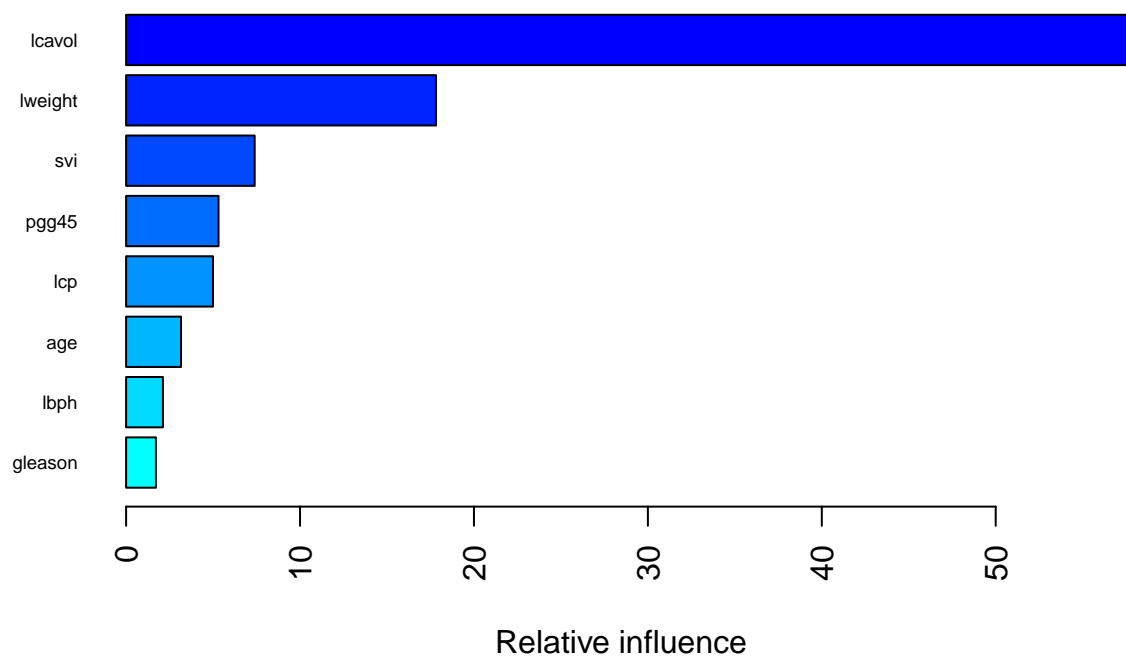
set.seed(1)
gbm.fit <- train(lpsa~., Prostate,
                method = "gbm",
                tuneGrid = gbm.grid,
                trControl = ctrl,
                verbose = FALSE)

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



Variable importance from boosting can be obtained using the `summary()` function.

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



```
##      var  rel.inf
```

```
## lcavol    lcavol 57.494112
## lweight  lweight 17.818879
## svi      svi    7.392286
## pgg45    pgg45  5.312763
## lcp      lcp    4.994192
## age      age    3.157564
## lbph     lbph   2.115772
## gleason  gleason 1.714432
```

The important variables are : lcavol, lweight, svi

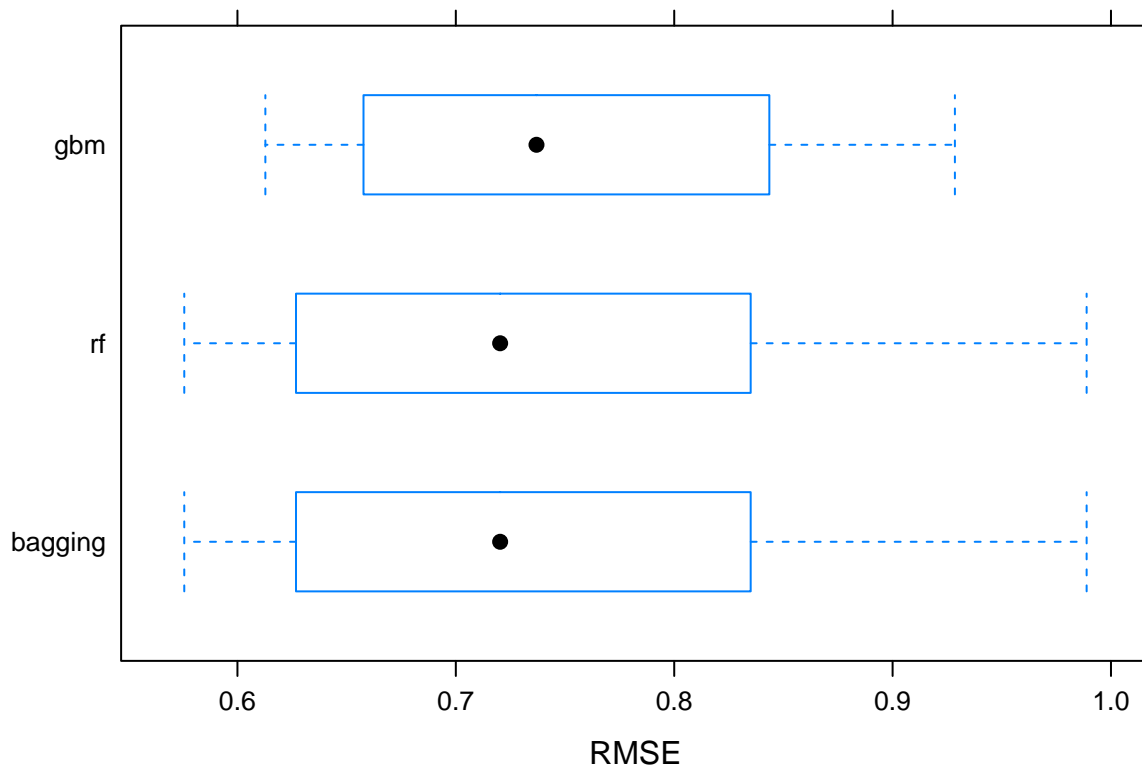
(f) Which of the above models will you select to predict PSA level? Explain.

```
resamp <- resamples(list(bagging = bagging, rf = rf.fit, gbm = gbm.fit))
summary(resamp)
```

```
##
## Call:
## summary.resamples(object = resamp)
##
## Models: bagging, rf, gbm
## Number of resamples: 10
##
## MAE
##      Min.    1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## bagging 0.4794948 0.5357349 0.6115087 0.5980269 0.6632726 0.7001466    0
## rf      0.4794948 0.5357349 0.6115087 0.5980269 0.6632726 0.7001466    0
## gbm     0.4988605 0.5382599 0.6238535 0.6060614 0.6541746 0.7189084    0
##
## RMSE
##      Min.    1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## bagging 0.5756510 0.6319992 0.7202955 0.7354396 0.8235912 0.9888492    0
## rf      0.5756510 0.6319992 0.7202955 0.7354396 0.8235912 0.9888492    0
## gbm     0.6127814 0.6613586 0.7369285 0.7499428 0.8330904 0.9285269    0
##
## Rsquared
##      Min.    1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## bagging 0.3250728 0.5203503 0.6199793 0.6083602 0.7499733 0.7763990    0
## rf      0.3250728 0.5203503 0.6199793 0.6083602 0.7499733 0.7763990    0
## gbm     0.3912307 0.4940316 0.6253058 0.6103438 0.7211938 0.8171347    0
```

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





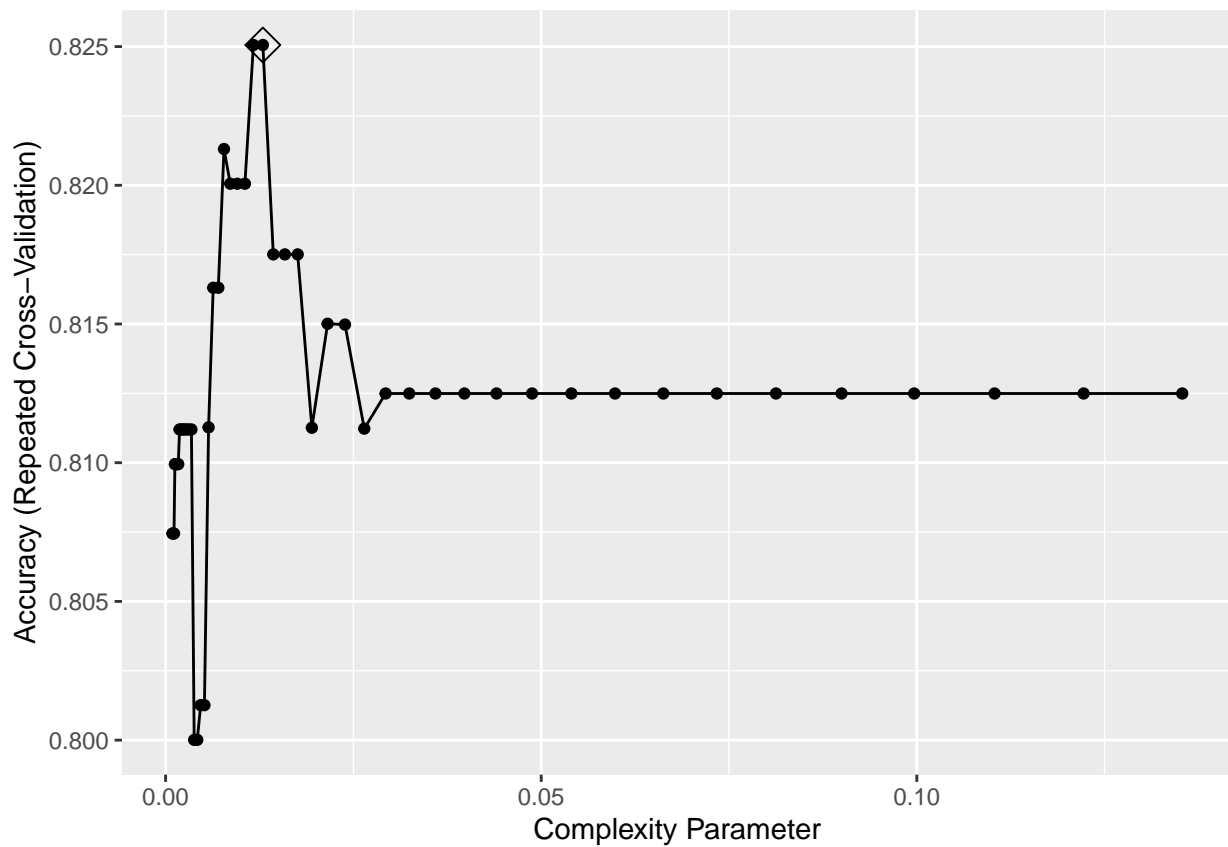
I would choose the Random forest model since the mean RMSE is lowest for the rf model.

2. This problem involves the OJ data in the ISLR package. The data contains 1070 purchases where the customers either purchased Citrus Hill or Minute Maid Orange Juice. A number of characteristics of customers and products are recorded. Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations. Use `set.seed()` for reproducible results.

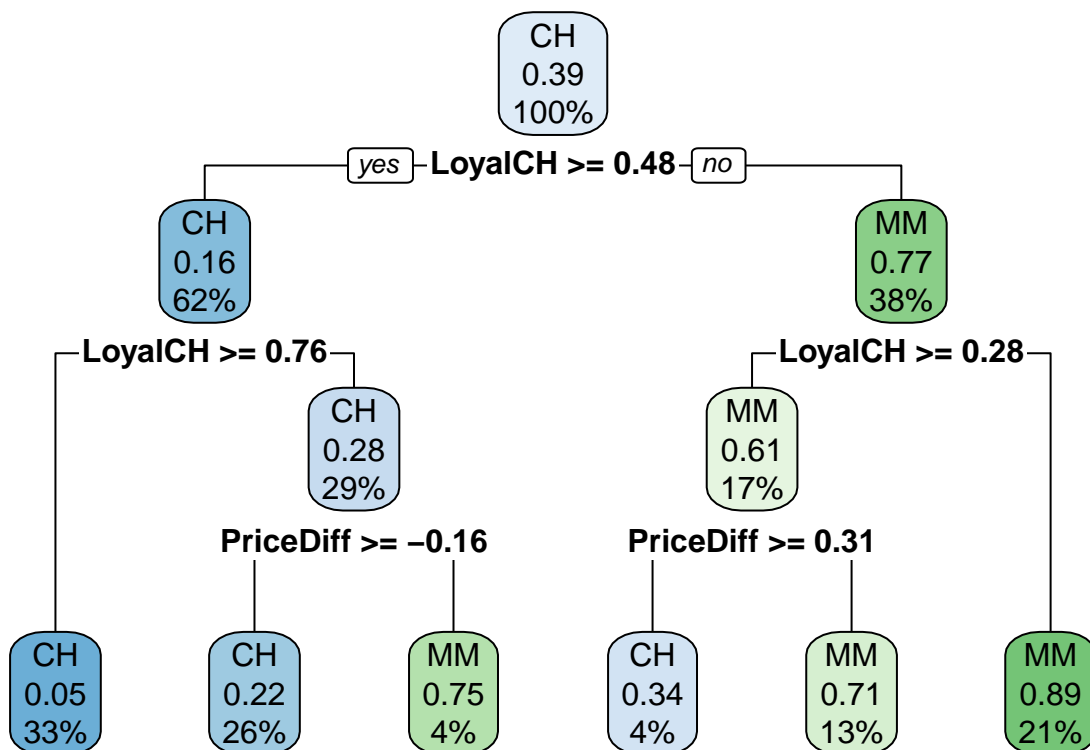
```
data("OJ")
set.seed(1)
rowTrain = createDataPartition(y = OJ$Purchase,
                               p = 0.747,
                               list = FALSE)
ctrl <- trainControl(method = "repeatedcv")
```

(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?

```
set.seed(1)
rpart.class <- train(Purchase ~., OJ,
                    subset = rowTrain,
                    method = "rpart",
                    tuneGrid = data.frame(cp = exp(seq(-7, -2, len = 50))),
                    trControl = ctrl,
                    metric = "Accuracy")
ggplot(rpart.class, highlight = T)
```



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



```
rpart.class$bestTune
```

```
##           cp  
## 27 0.01294638
```

Predicting the response on the test data.

```
rpart.pred <- predict(rpart.class, newdata = OJ[-rowTrain,])  
confusionMatrix(rpart.pred,  
                 reference = OJ$Purchase[-rowTrain])
```

```
## Confusion Matrix and Statistics  
##  
##           Reference  
## Prediction  CH  MM  
##           CH 142  24  
##           MM  23  81  
##  
##           Accuracy : 0.8259  
##           95% CI : (0.7753, 0.8692)  
##           No Information Rate : 0.6111  
##           P-Value [Acc > NIR] : 1.626e-14  
##  
##           Kappa : 0.6331  
##  
##           Mcnemar's Test P-Value : 1  
##  
##           Sensitivity : 0.8606  
##           Specificity : 0.7714  
##           Pos Pred Value : 0.8554  
##           Neg Pred Value : 0.7788  
##           Prevalence : 0.6111  
##           Detection Rate : 0.5259  
##           Detection Prevalence : 0.6148  
##           Balanced Accuracy : 0.8160  
##  
##           'Positive' Class : CH  
##
```

```
error_rate = mean(rpart.pred != OJ$Purchase[-rowTrain]) * 100  
error_rate
```

```
## [1] 17.40741
```

The test classification error rate is 17.407%

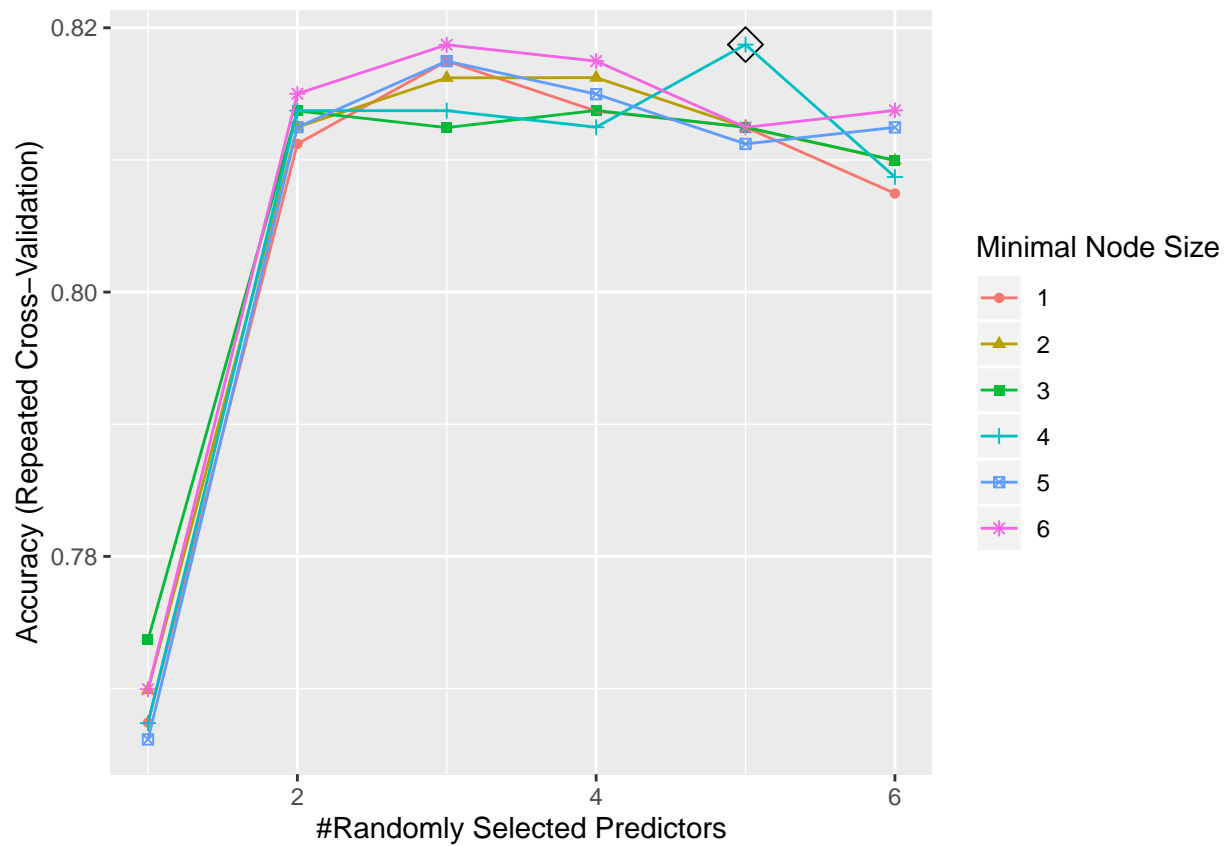
- (b) Perform random forests on the training set and report variable importance. What is the test error rate?

```

rf.grid1 = expand.grid(mtry = 1:6,
                      splitrule = "gini",
                      min.node.size = 1:6)

set.seed(1)
rf.fit1 = train(Purchase~., OJ,
                subset = rowTrain,
                method = "ranger",
                tuneGrid = rf.grid1,
                trControl = ctrl,
                importance = 'permutation')
ggplot(rf.fit1, highlight = TRUE)

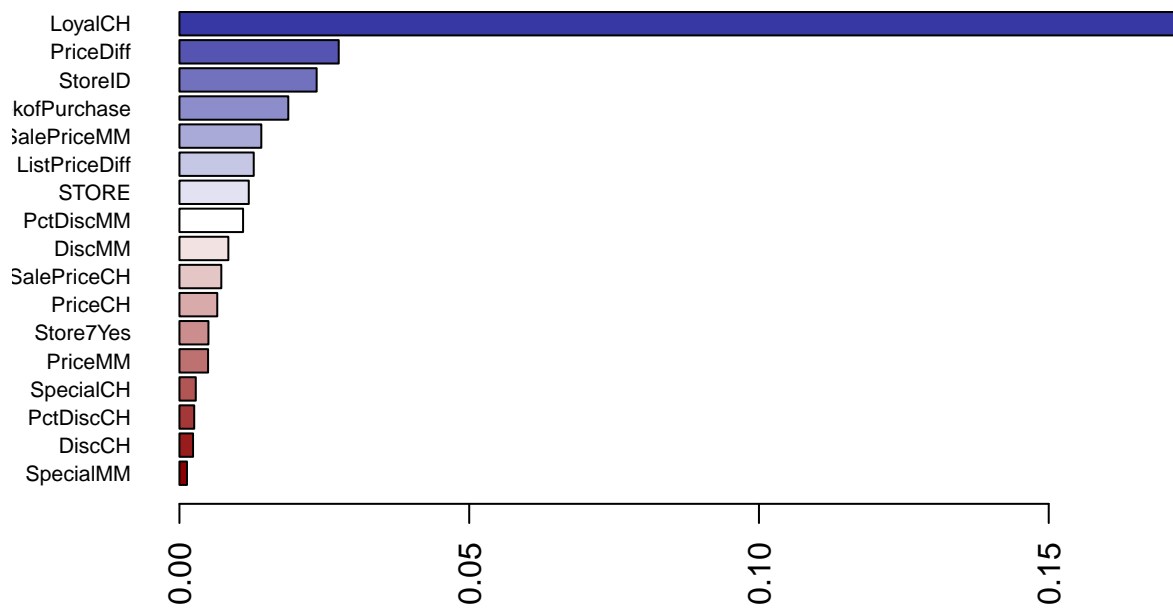
```



```

barplot(sort(ranger::importance(rf.fit1$finalModel), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("darkred", "white", "darkblue"))(19))

```



The important variables are LoyalCH

```
rf.pred <- predict(rf.fit1, newdata = OJ[-rowTrain,])
confusionMatrix(rf.pred,
                 reference = OJ$Purchase[-rowTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##           CH 140  28
##           MM  25  77
##
##           Accuracy : 0.8037
##           95% CI : (0.7512, 0.8494)
##           No Information Rate : 0.6111
##           P-Value [Acc > NIR] : 8.12e-12
##
##           Kappa : 0.5849
##
##           McNemar's Test P-Value : 0.7835
##
##           Sensitivity : 0.8485
##           Specificity : 0.7333
##           Pos Pred Value : 0.8333
##           Neg Pred Value : 0.7549
##           Prevalence : 0.6111
##           Detection Rate : 0.5185
##           Detection Prevalence : 0.6222
##           Balanced Accuracy : 0.7909
##
##           'Positive' Class : CH
##
```

```
error_rate_1 = mean(rf.pred != OJ$Purchase[-rowTrain]) * 100
error_rate_1
```

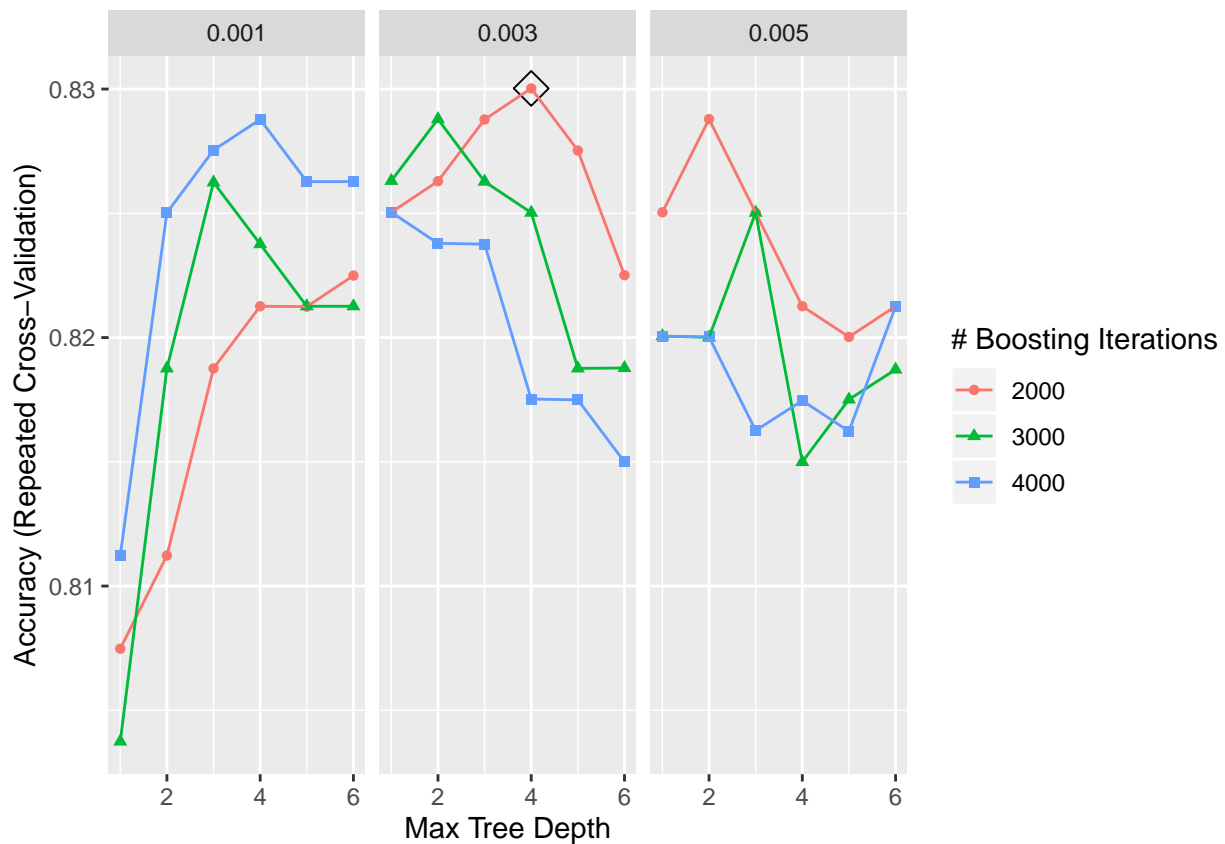
```
## [1] 19.62963
```

The test error rate is 19.63%

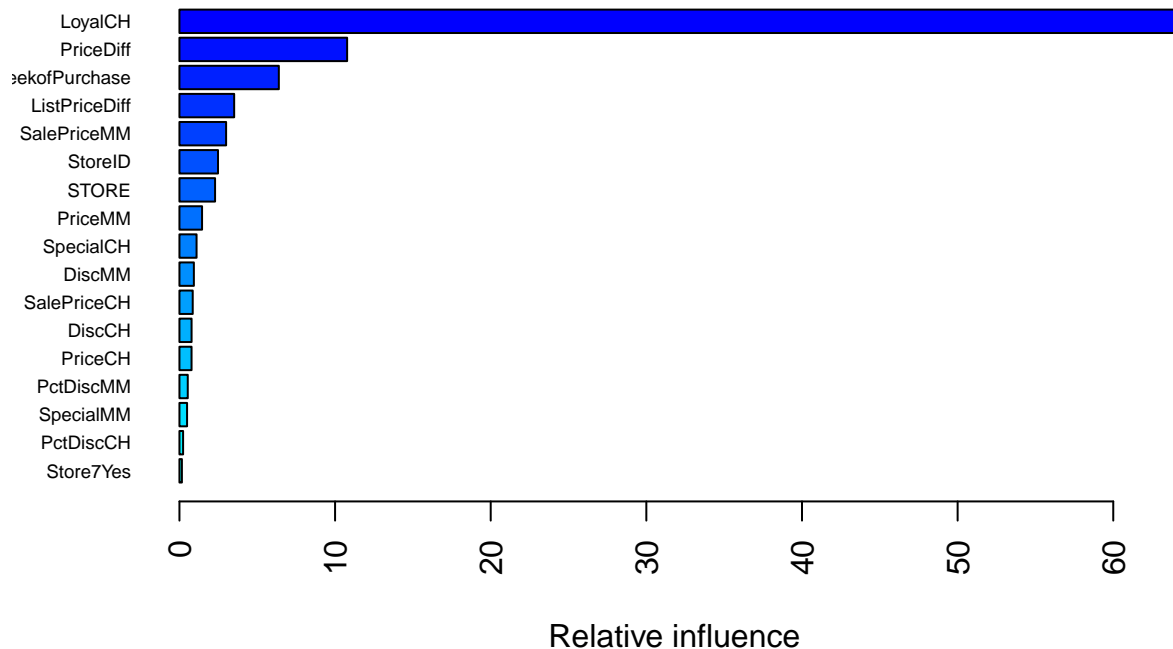
(c) Perform boosting on the training set and report variable importance. What is the test error rate?

```
gbm2.grid <- expand.grid(n.trees = c(2000,3000,4000),
                        interaction.depth = 1:6,
                        shrinkage = c(0.001,0.003,0.005),
                        n.minobsinnode = 1)

set.seed(1)
gbm2.fit <- train(Purchase ~., OJ,
                  subset = rowTrain,
                  tuneGrid = gbm2.grid,
                  trControl = ctrl,
                  method = "gbm",
                  distribution = "adaboost",
                  metric = "Accuracy",
                  verbose = FALSE)
ggplot(gbm2.fit, highlight = TRUE)
```



```
summary(gbm2.fit$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```



```
##           var    rel.inf
## LoyalCH      LoyalCH 64.2497567
## PriceDiff    PriceDiff 10.7738464
## WeekofPurchase WeekofPurchase 6.3868678
## ListPriceDiff ListPriceDiff 3.5154509
## SalePriceMM   SalePriceMM 2.9987268
## StoreID       StoreID 2.4774727
## STORE        STORE 2.2892477
## PriceMM       PriceMM 1.4530011
## SpecialCH     SpecialCH 1.1007800
## DiscMM        DiscMM 0.9273452
## SalePriceCH   SalePriceCH 0.8540572
## DiscCH        DiscCH 0.7788827
## PriceCH       PriceCH 0.7777353
## PctDiscMM     PctDiscMM 0.5350004
## SpecialMM     SpecialMM 0.4903835
## PctDiscCH     PctDiscCH 0.2340908
## Store7Yes     Store7Yes 0.1573546
```

The important variable is LoyalCH

```
gbm2.pred <- predict(gbm2.fit, newdata = OJ[-rowTrain,])
confusionMatrix(gbm2.pred,
                 reference = OJ$Purchase[-rowTrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
```

```
##          CH 146  25
##          MM   19  80
##
##          Accuracy : 0.837
##          95% CI : (0.7875, 0.879)
##    No Information Rate : 0.6111
##    P-Value [Acc > NIR] : 5.198e-16
##
##          Kappa : 0.6535
##
##    Mcnemar's Test P-Value : 0.451
##
##          Sensitivity : 0.8848
##          Specificity : 0.7619
##    Pos Pred Value : 0.8538
##    Neg Pred Value : 0.8081
##          Prevalence : 0.6111
##    Detection Rate : 0.5407
##    Detection Prevalence : 0.6333
##    Balanced Accuracy : 0.8234
##
##    'Positive' Class : CH
##
```

```
error_rate_2 = mean(gbm2.pred != OJ$Purchase[-rowTrain]) * 100
error_rate_2
```

```
## [1] 16.2963
```

The test error rate is 16.29%