Homework 4–Tree based methods

Ngoc Duong

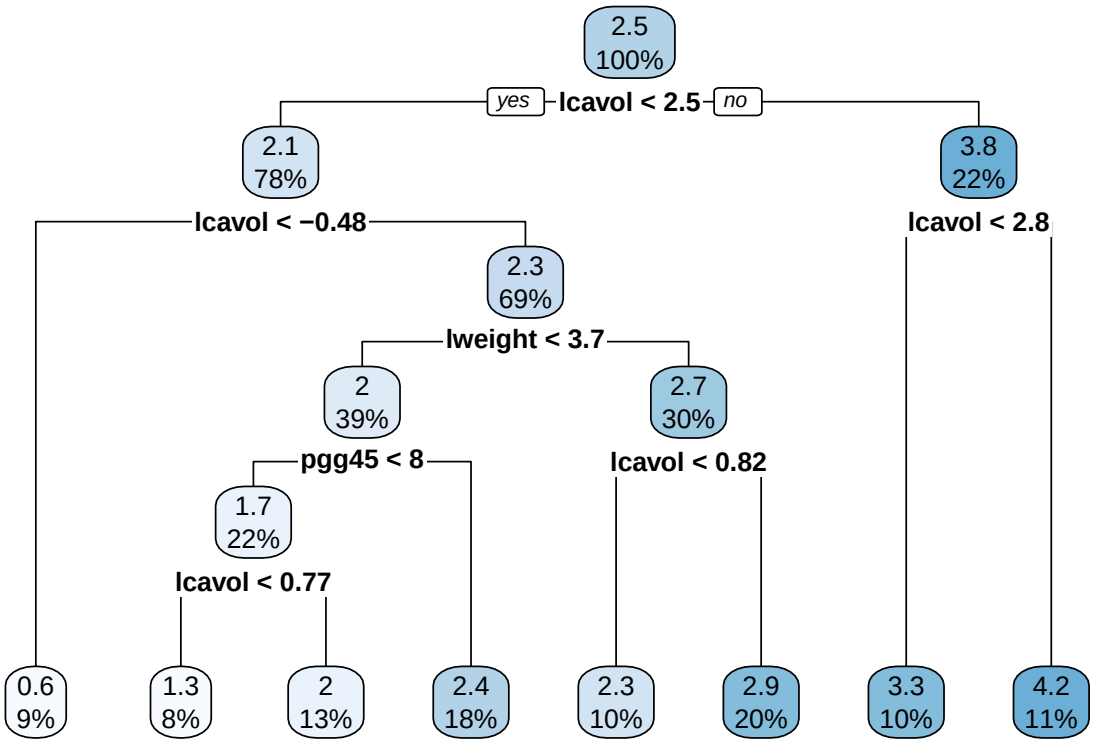
4/22/2020

Problem 1

```
data("Prostate") #import data
```

a. Fit a regression tree with "lpsa" as the response and the other variables as predictors

```
set.seed(13)
tree1 <-rpart(formula = lpsa~., data = Prostate)
rpart.plot(tree1)
```



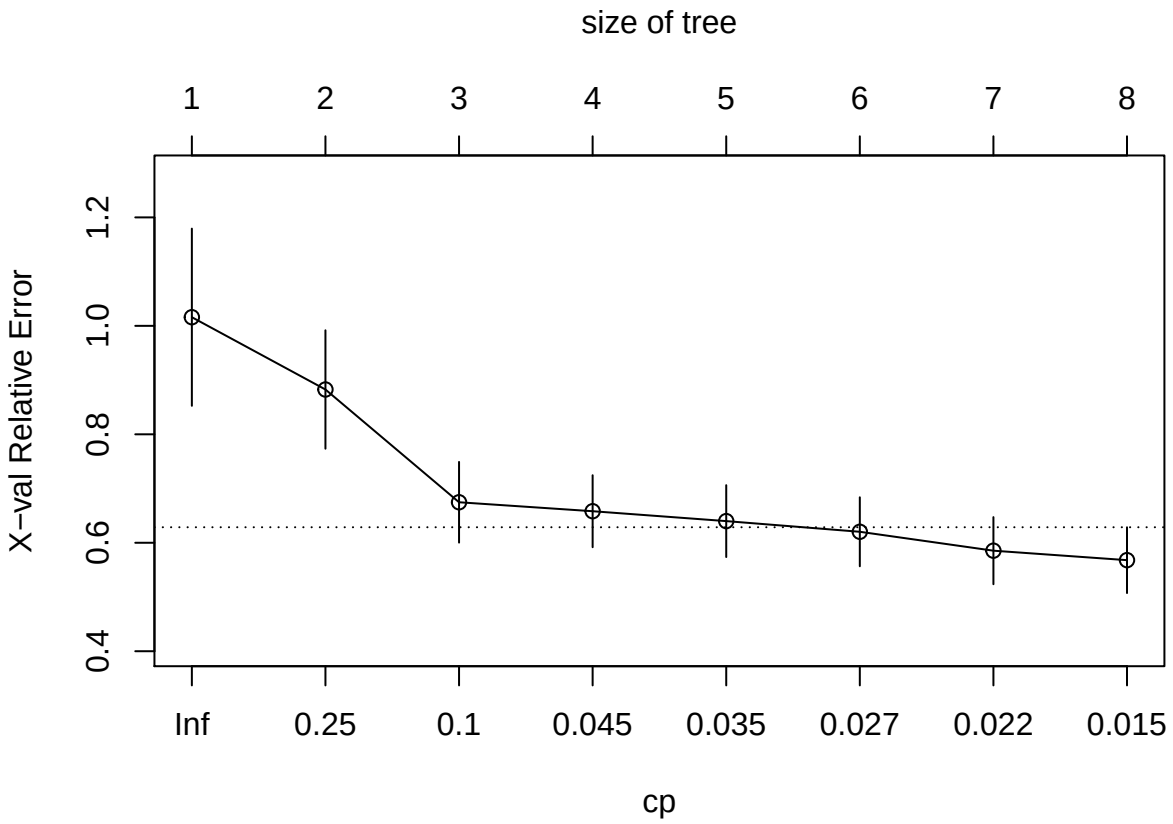
Next, we can use cross-validation to determine the optimal tree.

```
cpTable <-printcp(tree1)
```

```
##
## Regression tree:
## rpart(formula = lpsa ~ ., data = Prostate)
##
## 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 1.01591 0.163388
## 2 0.184647      1   0.65289 0.88262 0.109196
## 3 0.059316      2   0.46824 0.67467 0.074444
## 4 0.034756      3   0.40893 0.65816 0.066401
## 5 0.034609      4   0.37417 0.63994 0.066280
## 6 0.021564      5   0.33956 0.62026 0.063629
## 7 0.021470      6   0.31800 0.58540 0.061903
## 8 0.010000      7   0.29653 0.56793 0.060693
```

```
plotcp(tree1)
```



We can prune the tree based on the cp table

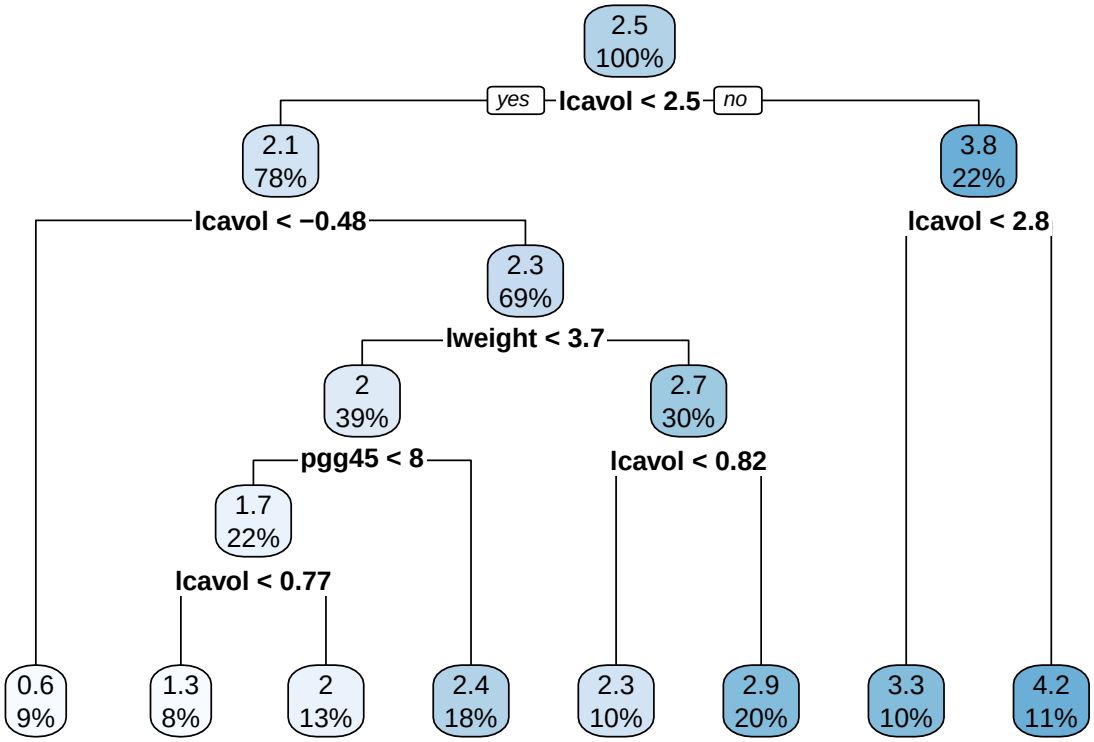
```
minErr <-which.min(cpTable[,4]) # minimum cross-validation error
tree2 <-prune(tree1, cp = cpTable[minErr,1])

tree3 <-prune(tree1, cp = cpTable[cpTable[,4]<cpTable[minErr,4]+cpTable[minErr,5],1][1])# 1SE rule
```

The tree size corresponding to the lowest cross-validation error is 7 (splits), compared to 5 splits in the tree obtained by the 1SE rule.

b. Plot the final tree choice (7 splits), and interpret a terminal node

```
rpart.plot(tree2)
```



Looking at the fifth terminal node from the left, we can interpret it as: the predicted log(prostate specific antigen) is 2.3 for log(cancer volume) between -0.48 and 0.82 and log(prostate weight) more than 3.7. The number of observations falling into this branch and have predicted value 2.3 is 10% of the total observations.

c. Perform bagging and report the variable importance

```
set.seed(13)
bagging = randomForest(lpsa ~ ., data = Prostate,
                       mtry = 8)
```

We can look at the lpsa prediction for first 10 subjects in the dataset using model obtained from bagging

```
predict(bagging, data = Prostate)[1:10]
```

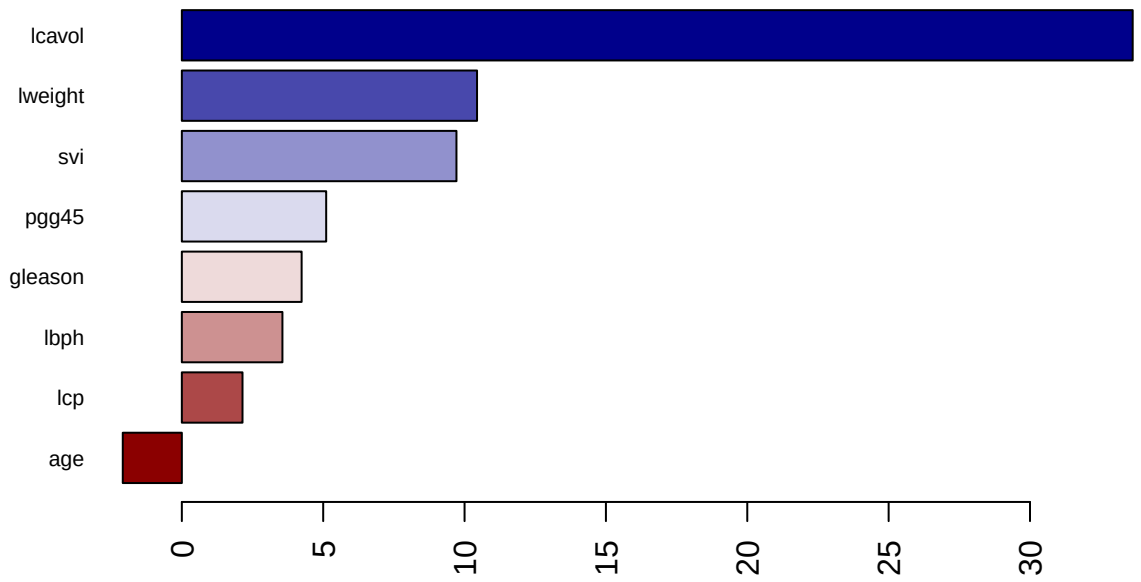
```
##         1         2         3         4         5         6         7         8
## 1.1920971 0.5413718 1.8896448 0.6318490 1.4172965 0.2620687 1.4207053 1.7537371
##         9        10
## 0.7110828 1.7251483
```

```
#variable importance
set.seed(13)
bagging2 = ranger(lpsa ~ ., data = Prostate, mtry = 8,
             splitrule = "variance", min.node.size = 5,
             importance = "permutation",
             scale.permutation.importance = TRUE)

ranger::importance(bagging2)
```

```
##    lcavol   lweight       age      lbph       svi       lcp   gleason     pgg45
## 33.632436 10.441518 -2.089082  3.558022  9.712803  2.146108  4.236473  5.105795
```

```r
barplot(sort(ranger::importance(bagging2), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("darkred","white","darkblue"))(8))
```



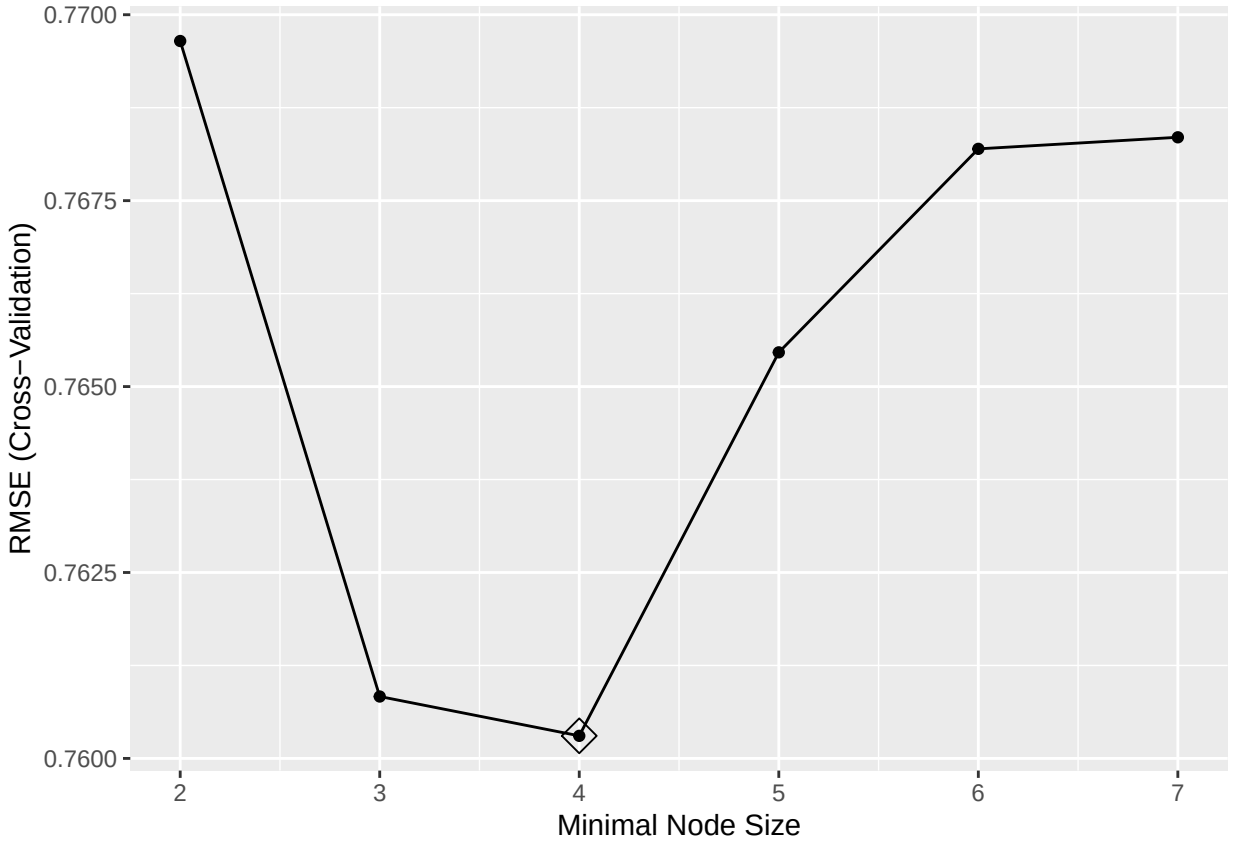
The variable importance above was computed from permuting out-of-bag data, suggesting log(cancer volume) and log(prostate weight) have the most influence in the fitted models. This can also be seen from the chosen decision tree in part b. On the other hand, age, log(capsular penetration), and log(benign prostatic hyperplasia amount) carry the least importance in the fitted models.

```r
ctrl = trainControl(method = "cv")

bag.grid = expand.grid(mtry = 8,
                       splitrule = "variance",
                       min.node.size = 2:7)

set.seed(13)
bag.fit <- train(lpsa~., Prostate,
                 method = "ranger",
                 tuneGrid = bag.grid,
                 trControl = ctrl)

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



Tuned bagging model has minimal node size of 4.

d. Perform random forest and report the variable importance

```
set.seed(13)
#fast implementation
rf = randomForest(lpsa ~ ., data = Prostate,
                        mtry = 3) #set smaller number of subset of trees to decorrelate
```

We can also look at the lpsa prediction for first 10 subjects in the dataset using model obtained from random forest

```
#prediction
predict(rf, data = Prostate)[1:10]
```

```
##         1         2         3         4         5         6         7         8
## 1.1835869 0.6213400 1.9305463 0.6712128 1.4362850 0.4365562 1.5750988 1.8031540
##         9        10
## 0.9680632 1.5524107
```

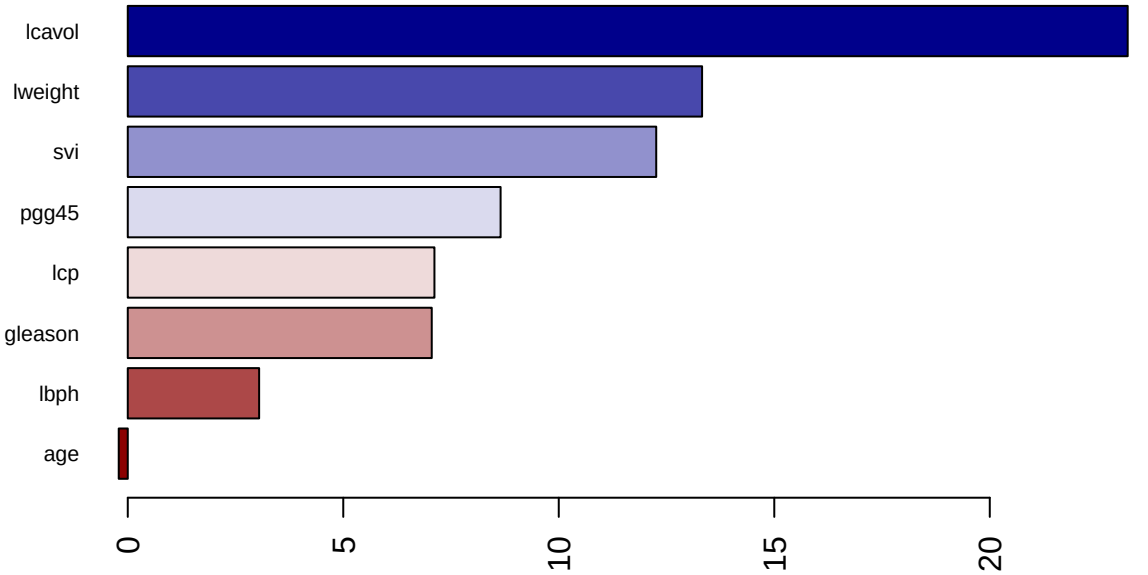
```
#importance importance
rf2 = ranger(lpsa ~ ., data = Prostate, mtry = 3, #decorrelate by picking mtry = 1/3 mtry in bagging
             splitrule = "variance", min.node.size = 5,
             importance = "permutation",
             scale.permutation.importance = TRUE)

ranger::importance(rf2)
```

```
##     lcavol    lweight        age       lbph        svi        lcp    gleason
## 23.2000217 13.3252950 -0.2096104  3.0477725 12.2613884  7.1149779  7.0526599
```

```
##      pgg45
##  8.6502531
```

```
barplot(sort(ranger::importance(rf2), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("darkred","white","darkblue"))(8))
```



From the variable importance plot, we can see log(cancer volume) and log(prostate weight) have the most influence in the fitted models. The plot also suggests age, log(capsular penetration), and log(benign prostatic hyperplasia amount) carry the least importance in the fitted models. This is similar to the variable importance plot obtained from bagging models above.

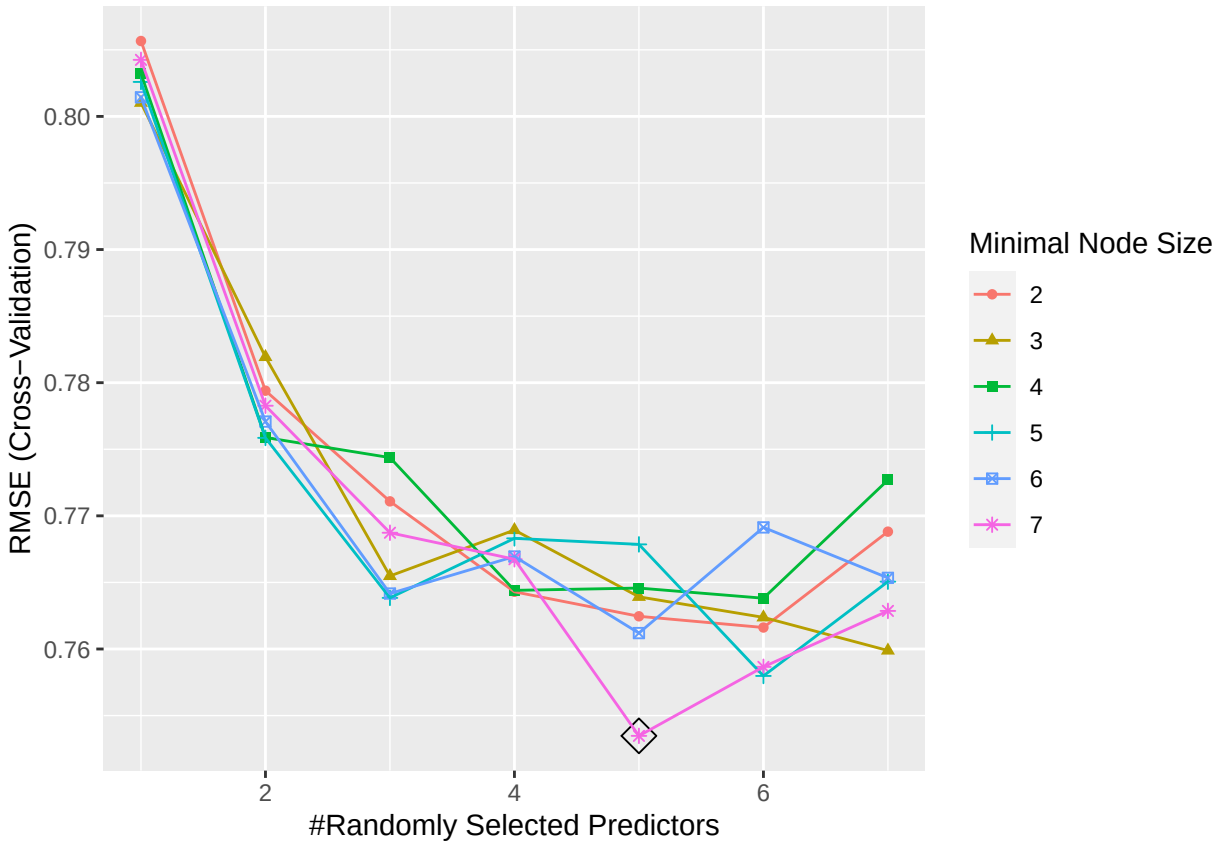
We can also try doing a grid search by caret for Random Forest

```
ctrl = trainControl(method = "cv")

rf.grid = expand.grid(mtry = 1:7,
                      splitrule = "variance",
                      min.node.size = 2:7)

set.seed(13)
rf.fit <- train(lpsa~., Prostate,
                method = "ranger",
                tuneGrid = rf.grid,
                trControl = ctrl)

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



The tuned RF model has the best minimal node size of 7 and the number of randomly selected predictors mtry of 4.

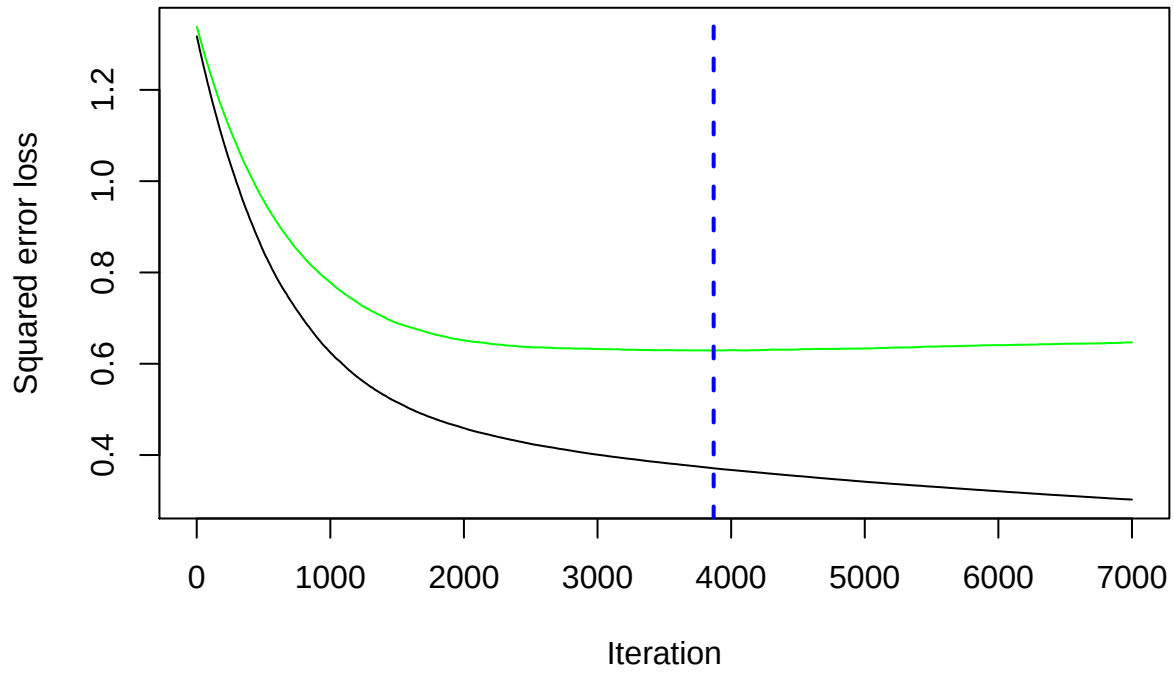
e. Perform boosting and report variable importance

We first fit a gradient boosting model using Gaussian loss function and 7000 iterations. This model only tunes over B (number of trees/iterations) given we have specified fixed interaction depth (d) and shrinkage (λ), although the latter two paramaters can also be tuned to select optimal model.

```
set.seed(13)
bst <- gbm(lpsa ~., Prostate,
           distribution = "gaussian",
           n.trees = 7000,
           interaction.depth = 3,
           shrinkage = 0.001,
           cv.folds = 10)
```

We then plot loss function as a result of number of trees added to the ensemble

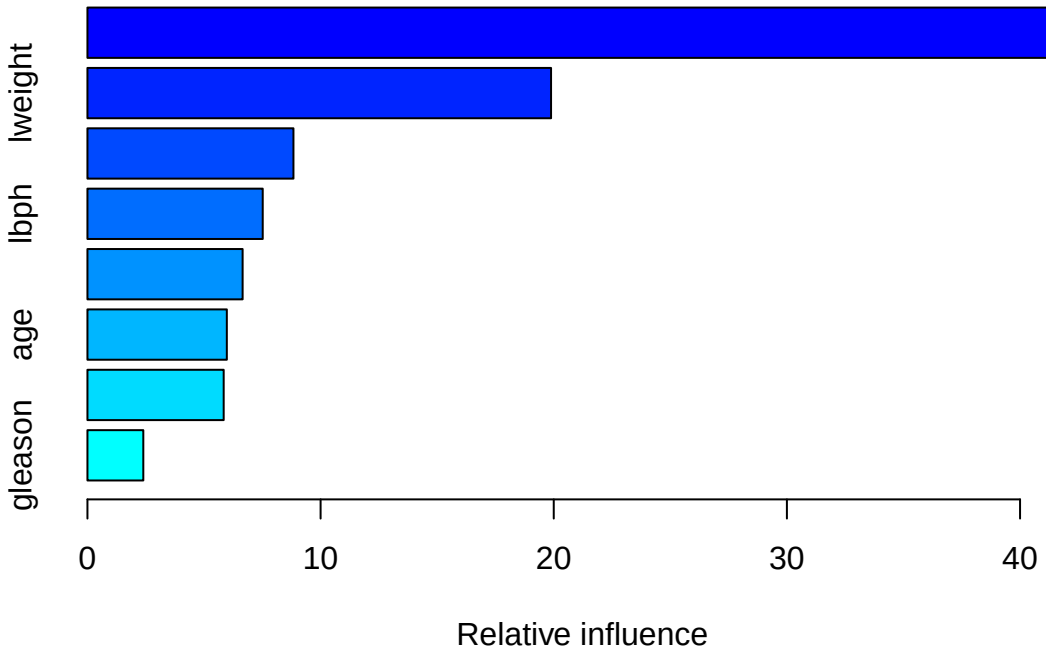
```
nt = gbm.perf(bst, method = "cv")
```



We can see that after 3869 trees, the cross-validated error for the RF model increases. So the optimal number of tree is 3869 for this RF model, given depth = 3, and shrinkage = 0.005.

Obtain variable importance for GBM

```
summary(bst)
```




```
##              var   rel.inf
## lcavol   lcavol 42.888088
## lweight lweight 19.885056
## svi         svi  8.835254
## lbph       lbph  7.518372
## pgg45     pgg45  6.655528
```

```
## age          age  5.978024
## lcp          lcp  5.843455
## gleason gleason  2.396224
```

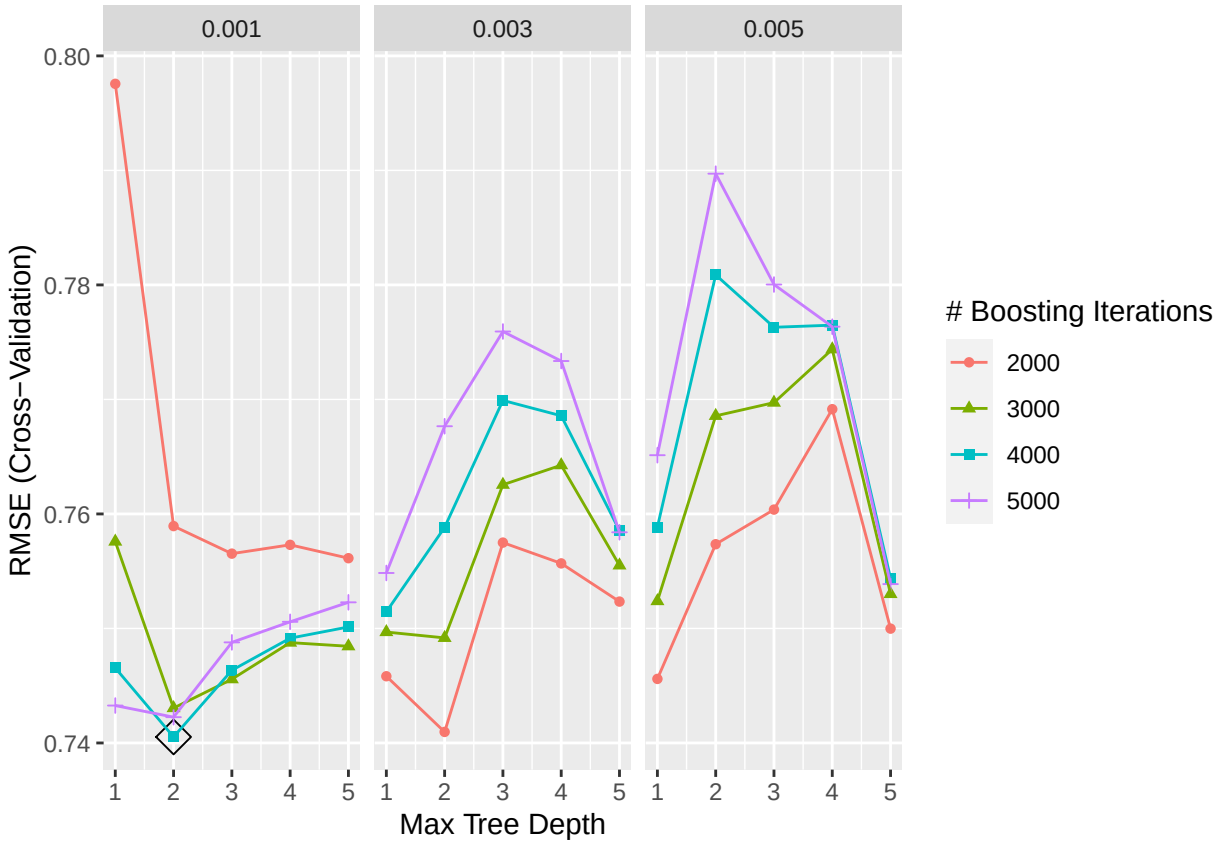
From the variable importance matrix and plot we can see log(cancer volume) and log(prostate weight) have the most influence in the fitted models. The plot also suggests age, log(capsular penetration), and gleason carry the least importance in the fitted models. This result is consistent with the result regarding variable importance obtained above.

We next try doing a grid search by caret for Boosting, tuning over all 3 parameters

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

set.seed(13)
gbm.fit = train(lpsa~., Prostate,
                method = "gbm",
                tuneGrid = gbm.grid,
                trControl = ctrl,
                verbose = FALSE)

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




```
gbm.fit$finalModel$shrinkage
```

```
## [1] 0.001
```

```
gbm.fit$finalModel$interaction.depth
```

```
## [1] 2
```

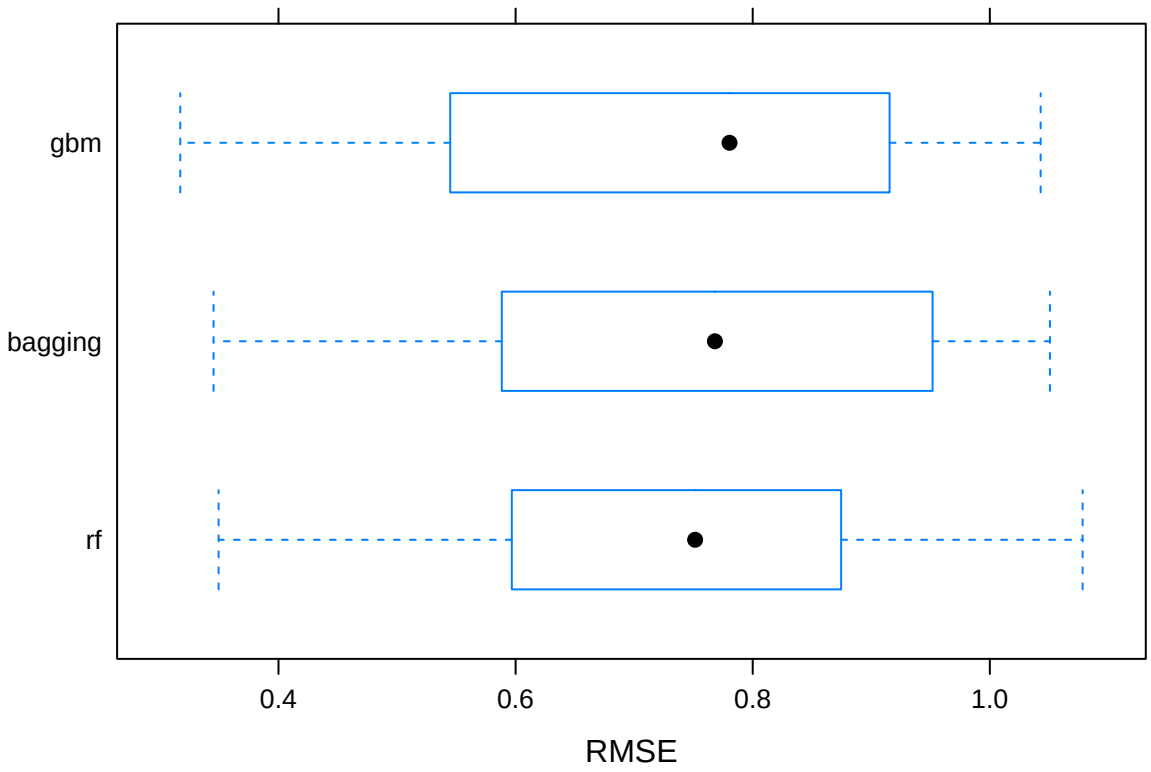
The gradient boosting model obtained from tuning all three parameters has number of trees selected to be 4000, tree depth = 2, and shrinkage = 0.001.

f. Final choice of model for PSA level prediction

```
resamp <- resamples(list(bagging = bag.fit, 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 0.2518361 0.5233915 0.6562976 0.6293077 0.7303459 0.8512513    0
## rf      0.2661378 0.5086105 0.6521511 0.6230123 0.7403321 0.8810701    0
## gbm     0.2421600 0.4984687 0.6684562 0.6160968 0.7384831 0.8378686    0
##
## RMSE
##              Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## bagging 0.3452152 0.6114582 0.7681143 0.7603038 0.9224418 1.050687    0
## rf      0.3494874 0.6157560 0.7514178 0.7534756 0.8615486 1.078265    0
## gbm     0.3171598 0.5885724 0.7804513 0.7405259 0.8946206 1.042860    0
##
## Rsquared
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## bagging 0.4152898 0.5724902 0.6682937 0.6495903 0.7490910 0.8865991    0
## rf      0.4533729 0.5485383 0.6803109 0.6681447 0.7654453 0.8666382    0
## gbm     0.4024557 0.5718816 0.6786513 0.6728525 0.7597220 0.9111992    0
```

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



From the table, we can see the model obtained from gradient boosting has slightly lower mean RMSE and slighly higher mean R-squared than the other two models. However, it has lower median RMSE (and median R-squared) as seen from the plot.

In the end, I decide to go with the RF model because although its performance is consistent (both low median and mean RMSE and high mean and median R-squared). Gradient boosting model might give better results but will require more effort and computational power to tune the three parameters. In this case, my tuned GBM did not outperform the RF model.

Problem 2

Import data (use data OJ in the ISLR package)

```
data("OJ")
oj_data = OJ %>% janitor::clean_names() %>%
  mutate(purchase = as.factor(purchase))
```

a. Fit a classification tree to dataset

```
set.seed(77)
rowTrain = createDataPartition(y = oj_data$purchase,
                               p = 2/3,
                               list = FALSE)

oj_train <- oj_data[rowTrain,]
oj_test <- oj_data[-rowTrain,]
```

Using "rpart"

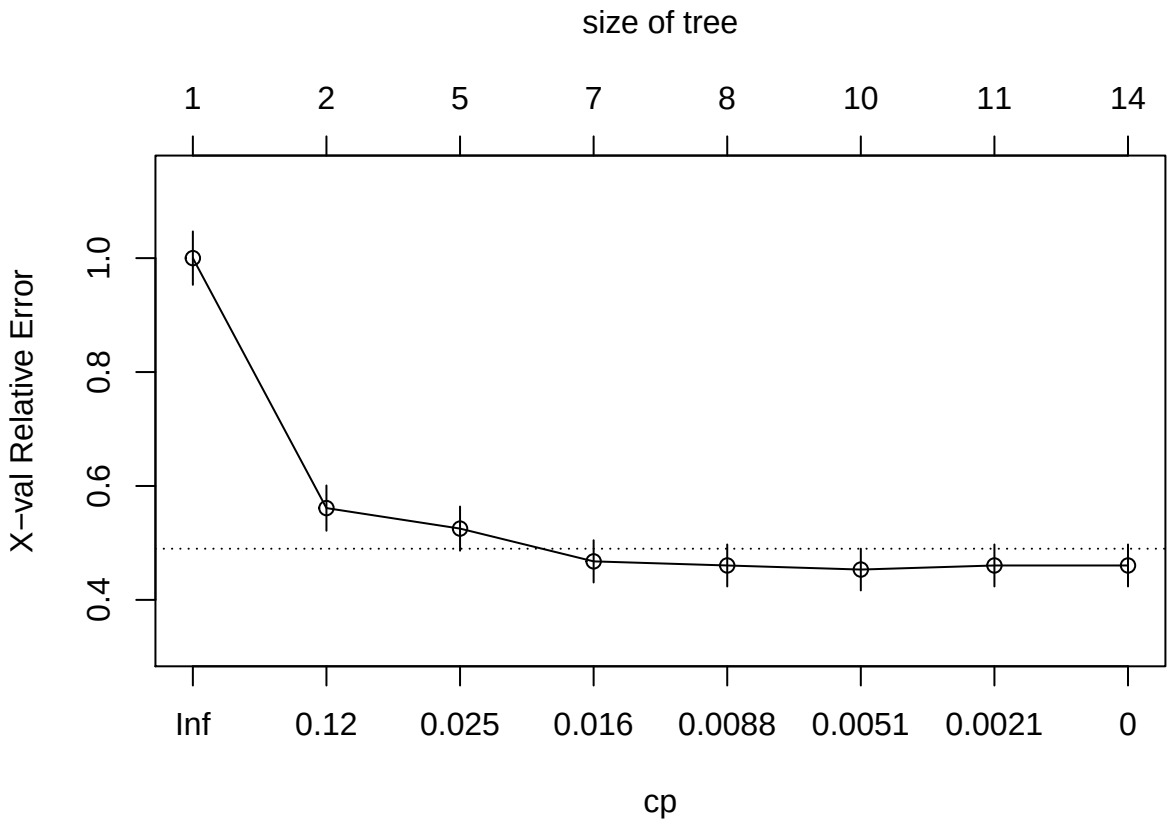
```
set.seed(77)
cl_tree = rpart(formula = purchase~., data = oj_train,
```

```
             control = rpart.control(cp = 0))

cpTable = printcp(cl_tree)
```

```
##
## Classification tree:
## rpart(formula = purchase ~ ., data = oj_train, control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
## [1] list_price_diff loyal_ch        price_diff      special_ch
## [5] store           weekof_purchase
##
## Root node error: 278/714 = 0.38936
##
## n= 714
##
##          CP nsplit rel error  xerror     xstd
## 1 0.5000000      0   1.00000 1.00000 0.046867
## 2 0.0275779      1   0.50000 0.56115 0.039718
## 3 0.0233813      4   0.41727 0.52518 0.038767
## 4 0.0107914      6   0.37050 0.46763 0.037092
## 5 0.0071942      7   0.35971 0.46043 0.036869
## 6 0.0035971      9   0.34532 0.45324 0.036642
## 7 0.0011990     10   0.34173 0.46043 0.036869
## 8 0.0000000     13   0.33813 0.46043 0.036869
```

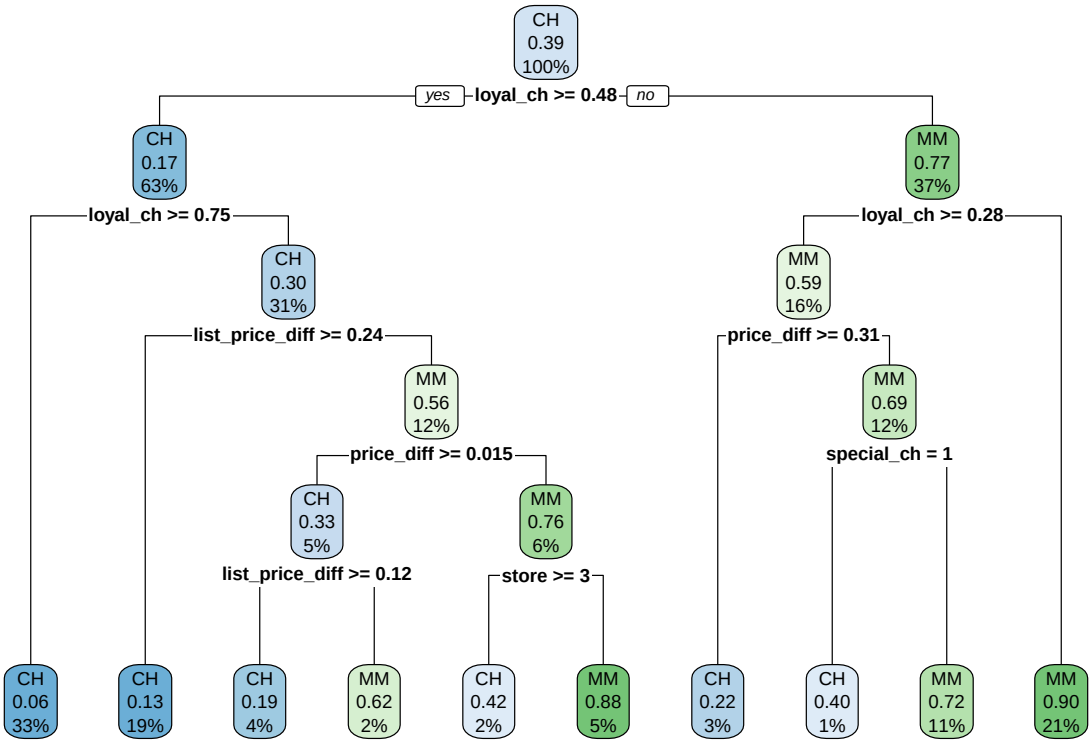
```
plotcp(cl_tree)
```



Use cross-validation to determine the tree size and create a plot of the final tree.

```
# min CV error
minErr = which.min(cpTable[,4])

# pruning
cl_tree1 = prune(cl_tree, cp = cpTable[minErr,1])
rpart.plot(cl_tree1)
```



Predict the response on the test data and find the test error rate.

```
tree_pred <- predict(cl_tree1, oj_test, type = "class")
head(tree_pred, 10)
```

```
##  1  4  6  7  8  9 10 15 17 19
## CH MM CH CH CH CH CH MM CH CH
## Levels: CH MM
```

The predicted class labels for the first 10 observations in the test data can be observed as above.

```
table(tree_pred, oj_test$purchase)
```

```
##
## tree_pred  CH  MM
##        CH 187  37
##        MM  30 102
```

From the confusion matrix, we can calculate the test error rate as: $(30+37)/(187+37+30+102) = 0.19$.

b. Perform random forests on the training set and report variable importance. Find test error rate

```
set.seed(77)
rf = ranger(purchase~., data = oj_train,
```

```
            mtry = 6, probability = TRUE)

rf.pred = predict(rf, oj_test, type = "response")$predictions[,1]
head(rf.pred, 20)
```

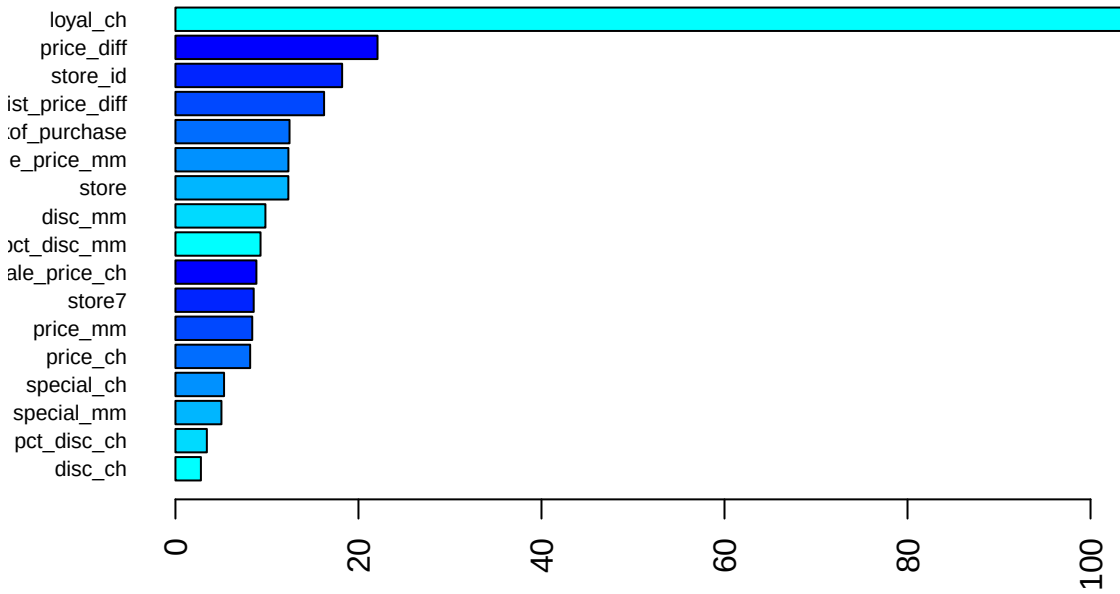
```
##  [1] 0.5882220 0.0711364 0.9301036 0.9474025 0.8083132 0.9239975 0.9226934
##  [8] 0.7440718 0.5922476 0.6993254 0.9868731 0.3223393 0.3679458 0.9799079
## [15] 0.9186659 0.4733796 0.8794714 0.9085238 0.9237571 0.7149212
```

We can see the first 20 predictions in the test dataset. Observations with probablity < 0.5 will be classified into "Minute Maid", while with probability > 0.5 will be classified into "Citrus Hill."

Obtain variable importance

```
set.seed(77)
rf2.final.per <-ranger(purchase~., oj_train,
                       mtry = 6,min.node.size = 5,
                       splitrule = "gini",
                       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"))(8))
```



We can see loyal_ch (customer brand loyaltyf for Citrus Hill), price_diff (sale price of MM minus sale price of CH), and store ID are the most influential features, whereas percent discount and discount offered for Citrus Hill are the least important in the random forest fitted model.

```
rf.pred = ifelse(rf.pred > 0.5, "CH","MM")
table(rf.pred, oj_test$purchase)
```

```
##
## rf.pred  CH  MM
##      CH 187  43
##      MM  30  96
```

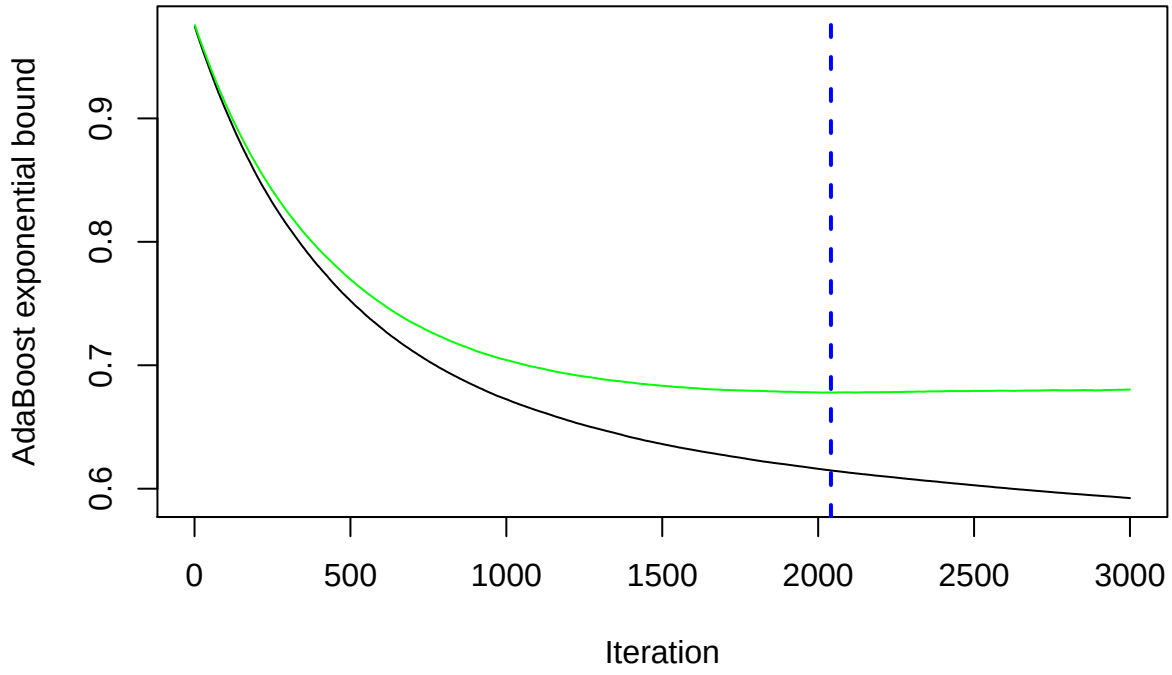
The test error rate can be calculated as $(30+43)/(187+43+30+96) = 0.21$

c. Perform boosting and report variable importance. Obtain test error rate

```
oj_train$purchase <-as.numeric(oj_train$purchase=="MM")
oj_test$purchase <-as.numeric(oj_test$purchase=="MM")
```

```
set.seed(77)
bst1 <-gbm(purchase~., oj_train,
           distribution = "adaboost",
           n.trees = 3000,interaction.depth = 2,
           shrinkage = 0.002,
           cv.folds = 10)

nt1 <-gbm.perf(bst1, method = "cv")
```

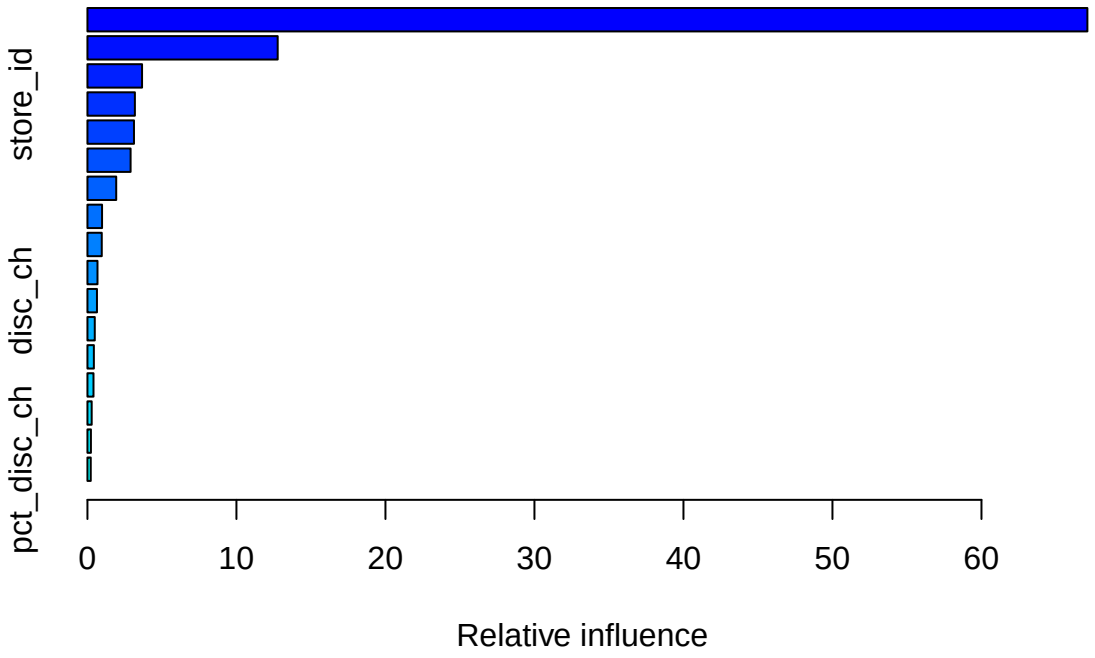



```
nt1
```

```
## [1] 2041
```

The number of optimal trees/iterations picked by gbm function is 2041, which minimizes the error loss given interaction depth = 3 an shrinkage = 0.005.

```
summary(bst1)
```




```
##                             var    rel.inf
## loyal_ch               loyal_ch 67.1094987
## price_diff           price_diff 12.7678944
## weekof_purchase weekof_purchase  3.6661785
## store_id               store_id  3.1836302
## list_price_diff list_price_diff  3.1234904
## sale_price_mm     sale_price_mm  2.8956804
## store                     store  1.9303689
## sale_price_ch     sale_price_ch  0.9811661
## price_mm               price_mm  0.9602485
## pct_disc_mm         pct_disc_mm  0.6748082
## disc_ch                 disc_ch  0.6385541
## disc_mm                 disc_mm  0.4914635
## store7                   store7  0.4315498
## special_mm           special_mm  0.4064008
## price_ch               price_ch  0.2843065
## special_ch           special_ch  0.2329142
## pct_disc_ch         pct_disc_ch  0.2218467
```

We can observe similar results as in RF model regarding variable importance, e.g., "loyal_ch" (customer brand loyaltyf for Citrus Hill), price_diff (sale price of MM minus sale price of CH) and store id are still the most influential features, whereas percent discount and discount offered for Citrus Hill are still among the least important variables.

Obtain test error rate

```
gbm.pred = predict(bst1, newdata = oj_test, type = "response")
```

```
## Using 2041 trees...
```

```
head(gbm.pred, 20)
```

```
##  [1] 0.43654903 0.76034684 0.03923052 0.06378494 0.06837687 0.06865853
##  [7] 0.06733617 0.12179071 0.55082177 0.48784798 0.03567037 0.53423840
## [13] 0.66647244 0.07401541 0.04042834 0.67841139 0.11231166 0.11092811
```

```
## [19] 0.10771026 0.05215241
```

Similarly, we can see the first 20 predictions in the test dataset. Observations with probablity < 0.5 will be classified into "Minute Maid", while with probability > 0.05 will be classified into "Citrus Hill."

```
gbm.pred = ifelse(gbm.pred > 0.5, "CH","MM")
table(gbm.pred, oj_test$purchase)
```

```
##
## gbm.pred   0   1
##       CH  27 102
##       MM 190  37
```

With CH = 1, MM = 0, the test error rate for this model is (27+37)/(27+190+102+37) = 0.18

Use caret totune all models

```
data("OJ")
oj_data = OJ %>% janitor::clean_names() %>%
  mutate(purchase = as.factor(purchase))
```

```
set.seed(77)
rowTrain = createDataPartition(y = oj_data$purchase,
                               p = 2/3,
                               list = FALSE)

oj_train <- oj_data[rowTrain,]
oj_test <- oj_data[-rowTrain,]
```

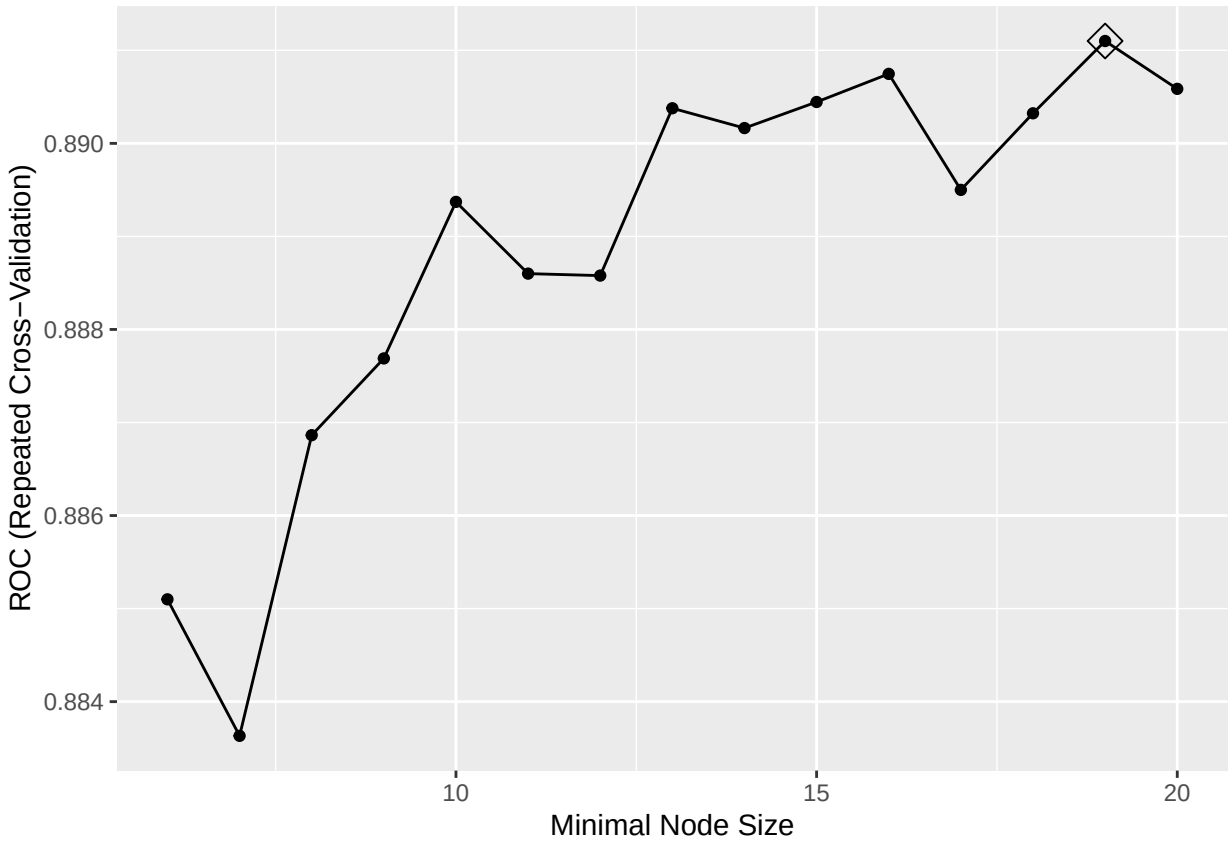
Bagging

```
ctrl_cl = trainControl(method = "repeatedcv",
                    summaryFunction = twoClassSummary,
                    classProbs = TRUE)

cl.bag.tunegrid = expand.grid(mtry = 17,
                              splitrule = "gini",
                              min.node.size = 6:20)

set.seed(77)
class.bag.fit = train(purchase~.,
                  data = oj_train,
                  tuneGrid = cl.bag.tunegrid,
                  trControl = ctrl_cl,
                  method = "ranger",
                  metric = "ROC",
                  verbose = FALSE)

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




```
summary(class.bag.fit$finalModel, las = 2, cBars = 8, cex.names = 0.6)
```

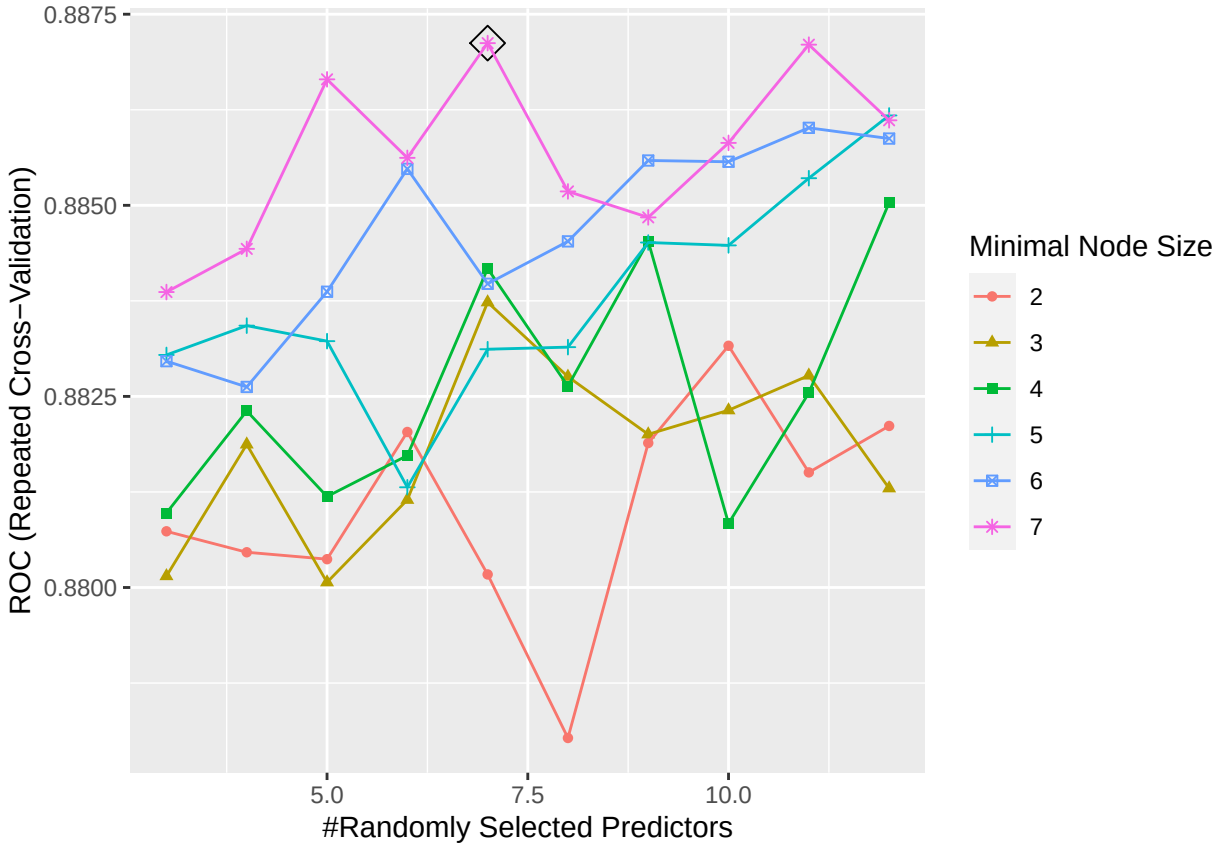
```
##                             Length Class         Mode
## predictions                 1428   -none-        numeric
## num.trees                      1   -none-        numeric
## num.independent.variables      1   -none-        numeric
## mtry                           1   -none-        numeric
## min.node.size                  1   -none-        numeric
## prediction.error               1   -none-        numeric
## forest                        10   ranger.forest list
## splitrule                      1   -none-        character
## treetype                       1   -none-        character
## call                           9   -none-        call
## importance.mode                1   -none-        character
## num.samples                    1   -none-        numeric
## replace                        1   -none-        logical
## xNames                        17   -none-        character
## problemType                    1   -none-        character
## tuneValue                      3   data.frame    list
## obsLevels                      2   -none-        character
## param                          1   -none-        list
```

Random forest

```
cl.rf.tunegrid = expand.grid(mtry = 3:12,
                             splitrule = "gini",
                             min.node.size = 2:7)
```

```
set.seed(77)
class.rf.fit = train(purchase~.,
                     data = oj_train,
                     tuneGrid = cl.rf.tunegrid,
                     trControl = ctrl_cl,
                     method = "ranger",
                     metric = "ROC",
                     verbose = FALSE)

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




```
summary(class.rf.fit$finalModel, las = 2, cBars = 8, cex.names = 0.6)
```

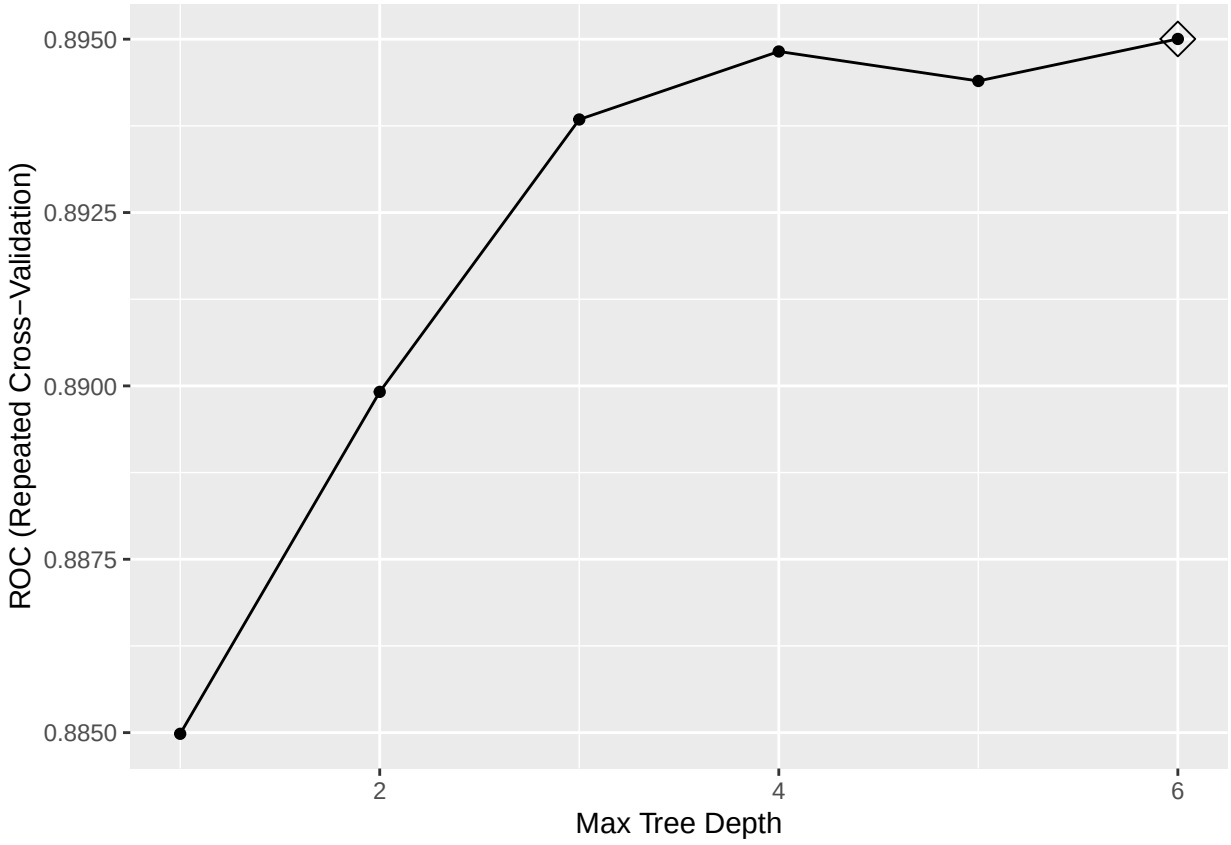
```
##                             Length Class         Mode
## predictions                 1428   -none-        numeric
## num.trees                      1   -none-        numeric
## num.independent.variables      1   -none-        numeric
## mtry                           1   -none-        numeric
## min.node.size                  1   -none-        numeric
## prediction.error               1   -none-        numeric
## forest                        10   ranger.forest list
## splitrule                      1   -none-        character
## treetype                       1   -none-        character
## call                           9   -none-        call
## importance.mode                1   -none-        character
## num.samples                    1   -none-        numeric
## replace                        1   -none-        logical
```

```
## xNames                    17   -none-        character
## problemType                1   -none-        character
## tuneValue                  3   data.frame    list
## obsLevels                  2   -none-        character
## param                      1   -none-        list
```

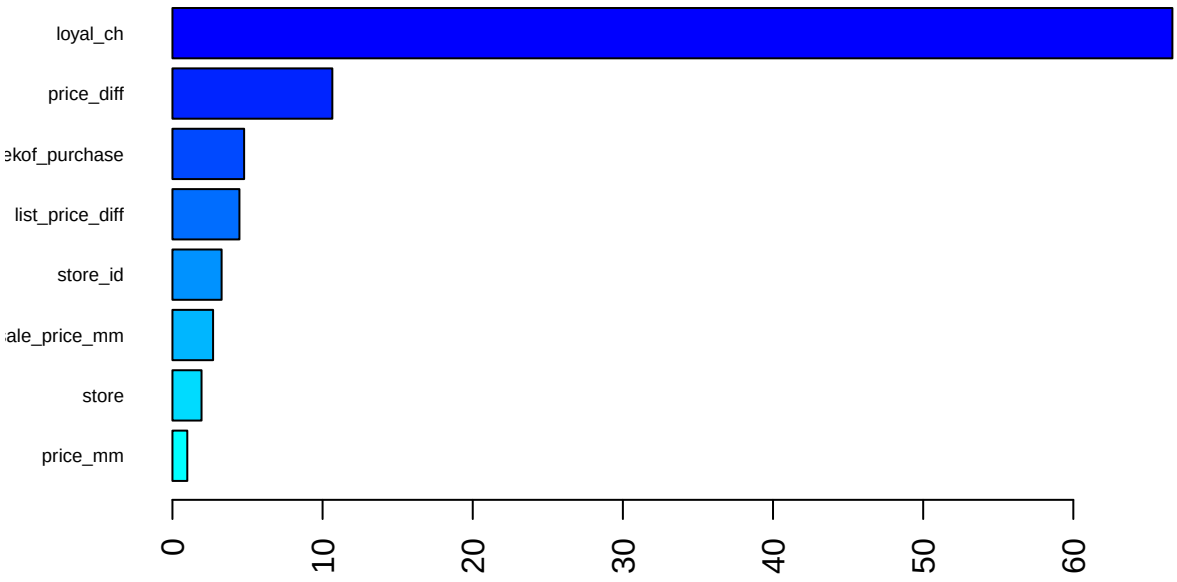
Gradient boosting

```
cl.gbm.tunegrid = expand.grid(n.trees = 3000,
                        interaction.depth = 1:6,
                        shrinkage = 0.001,
                        n.minobsinnode = 1)

set.seed(77)
# Binomial loss function
class.gbm.fit = train(purchase~.,
                  data = oj_train,
                  tuneGrid = cl.gbm.tunegrid,
                  trControl = ctrl_cl,
                  method = "gbm",
                  distribution = "bernoulli",
                  metric = "ROC",
                  verbose = FALSE)
ggplot(class.gbm.fit, highlight = TRUE)
```




```
summary(class.gbm.fit$finalModel, las = 2, cBars = 8, cex.names = 0.6)
```




```
##                             var    rel.inf
## loyal_ch               loyal_ch 66.5982421
## price_diff           price_diff 10.6485938
## weekof_purchase weekof_purchase  4.7743012
## list_price_diff list_price_diff  4.4608345
## store_id               store_id  3.2757946
## sale_price_mm     sale_price_mm  2.7074506
## store                     store  1.9370192
## price_mm               price_mm  0.9894043
## special_ch           special_ch  0.8315732
## sale_price_ch     sale_price_ch  0.7846356
## disc_mm                disc_mm  0.6647832
## disc_ch                disc_ch  0.5808898
## pct_disc_mm         pct_disc_mm  0.5108233
## price_ch               price_ch  0.4997065
## special_mm           special_mm  0.3775586
## store7Yes             store7Yes  0.2387455
## pct_disc_ch         pct_disc_ch  0.1196442
```

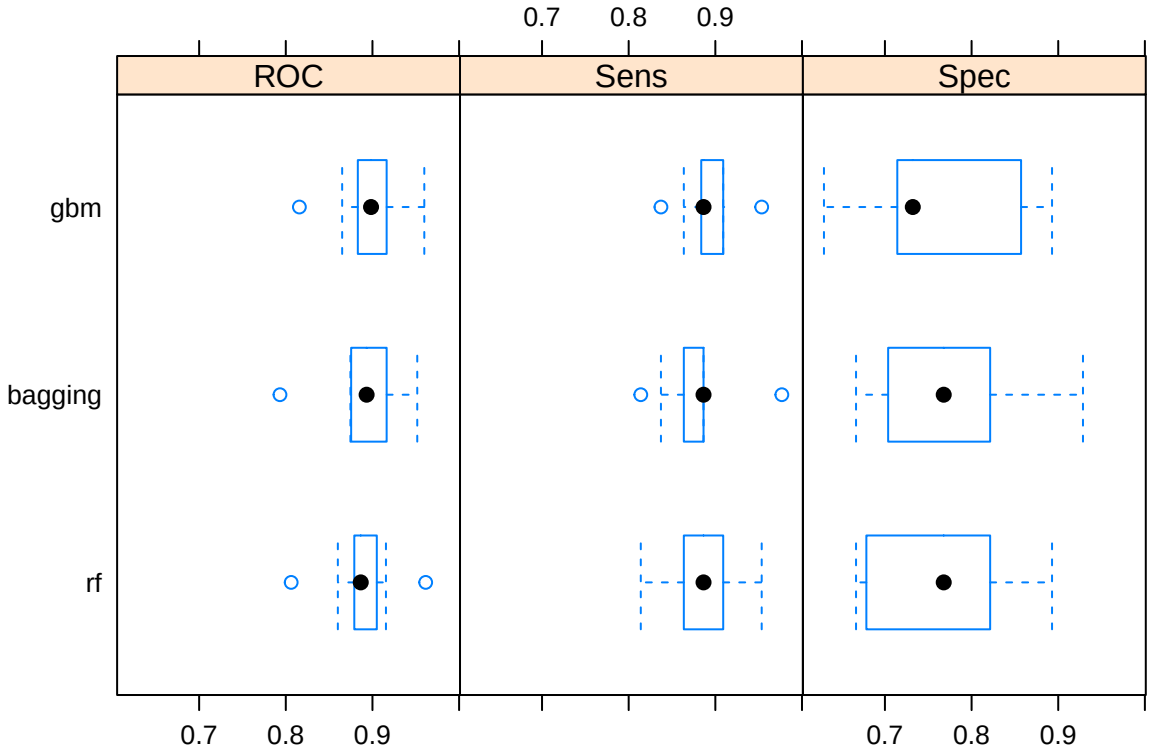
Resampling

```
cl_resample = resamples(list(bagging = class.bag.fit, rf = class.rf.fit, gbm = class.gbm.fit))
summary(cl_resample)
```

```
##
## Call:
## summary.resamples(object = cl_resample)
##
## Models: bagging, rf, gbm
## Number of resamples: 10
##
## ROC
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## bagging 0.7932817 0.8795728 0.8932630 0.8910999 0.9108228 0.9517045    0
```

```
## rf      0.8062016 0.8794117 0.8865052 0.8871219 0.9013657 0.9614448    0
## gbm     0.8156761 0.8840985 0.8983886 0.8950031 0.9135222 0.9598214    0
##
## Sens
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## bagging 0.8139535 0.8686575 0.8863636 0.8807082 0.8863636 0.9767442    0
## rf      0.8139535 0.8686575 0.8863636 0.8852537 0.9034091 0.9534884    0
## gbm     0.8372093 0.8837209 0.8863636 0.8899049 0.9034091 0.9534884    0
##
## Spec
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## bagging 0.6666667 0.7063492 0.7678571 0.7763228 0.8214286 0.9285714    0
## rf      0.6666667 0.6941138 0.7678571 0.7728836 0.8214286 0.8928571    0
## gbm     0.6296296 0.7142857 0.7321429 0.7619048 0.8303571 0.8928571    0
```

```
bwplot(cl_resample, metrics = "RMSE")
```



We can see that all three models perform consistently well and are very comparable. However, the tunded GBM model seems to have the highest median AUC (at 0.898) and specificity. This comes with a trade-off of lower specificity.