Tree-BasedMethods Exercises Ch 8

Darshan Patel

2/21/2019

The following set of problems are from the applied exercises section in ISLR Chapter 8: Tree-Based Methods.

```
rm(list = ls())
library(MASS)
library(ISLR)
library(tidyverse)
```

```
## Warning: package 'tibble' was built under R version 3.4.4

## Warning: package 'tidyr' was built under R version 3.4.4

## Warning: package 'purrr' was built under R version 3.4.4

## Warning: package 'dplyr' was built under R version 3.4.4
```

```
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 3.4.4

## Warning: replacing previous import 'ggplot2::empty' by 'plyr::empty' when
## loading 'GGally'
```

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.4.4
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.4.4
```

```
library(glmnet)
```

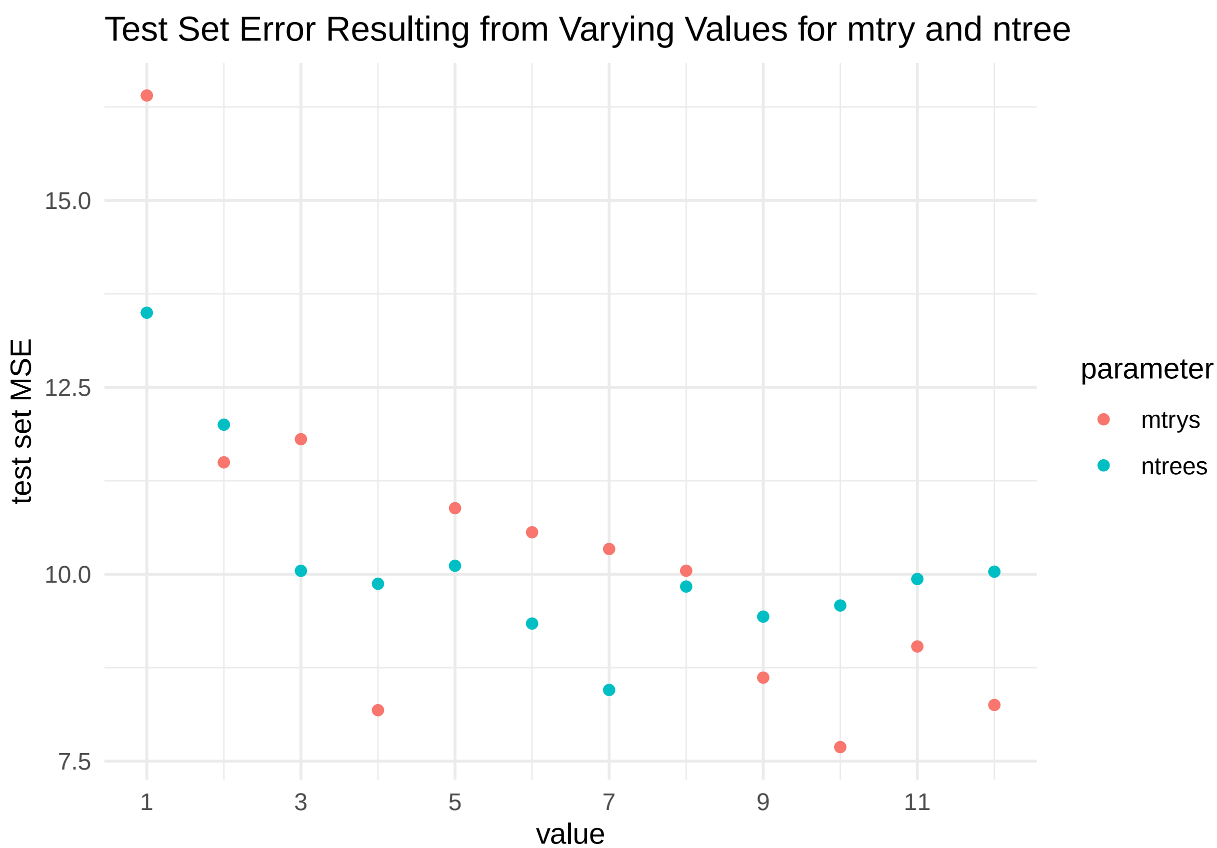
```
## Warning: package 'glmnet' was built under R version 3.4.4

## Warning: package 'Matrix' was built under R version 3.4.4
```

```
library(class)
```

Question 7: Using the `Boston` dataset, create a plot displaying the test error resulting from random forests on this data set for a comprehensive range of values for `mtry` and `ntree`. Describe the results obtained.

```
df = Boston
set.seed(7)
indices = sample(1:nrow(df), size = 0.7*nrow(df))
train = df[indices,]
test = df[-indices,]
mtrys = c()
```

```r
ntrees = c()
for(i in 1:12){
  temp_rf = randomForest(medv~., data = train,
                         mtry = 10, ntree = i)
  mtrys = c(mtrys, mean((predict(temp_rf, test) - test$medv)^2))
  temp_rf2 = randomForest(medv~., data = train,
                          mtry = i, ntree = 10)
  ntrees = c(ntrees, mean((predict(temp_rf2, test) - test$medv)^2))
}
rf_stats_df = data.frame(x = 1:12, mtrys, ntrees)
rf_stats_df %>% gather(parameter, mse, mtrys, ntrees) %>%
  ggplot(aes(x = x, y = mse, color = parameter)) + geom_point() +
  labs(x = "value", y = "test set MSE") +
  scale_x_continuous(breaks = seq(1, 12, by = 2)) +
  ggtitle("Test Set Error Resulting from Varying Values for mtry and ntree") +
  theme_minimal()
```



As the number of trees to grow increases, the test set MSE decreases until 7 trees are grown and then increases. As for the number of variables to randomly sample, randomly sampling from 10 variables provided the lowest test set MSE, followed closely by 4 variables.

Question 8: Using the `Carseats` dataset, predict `Sales` using regression trees and related approaches, treating the response as a quantitative variable.

(a) Split the data into a training set and a test set.

```r
df = Carseats
set.seed(8)
indices = sample(1:nrow(df), size = 0.7*nrow(df))
train = df[indices,]
test = df[-indices,]
```

(b) Fit a regression tree to the training set. Plot the tree and interpret the results. What test error rate was obtained?

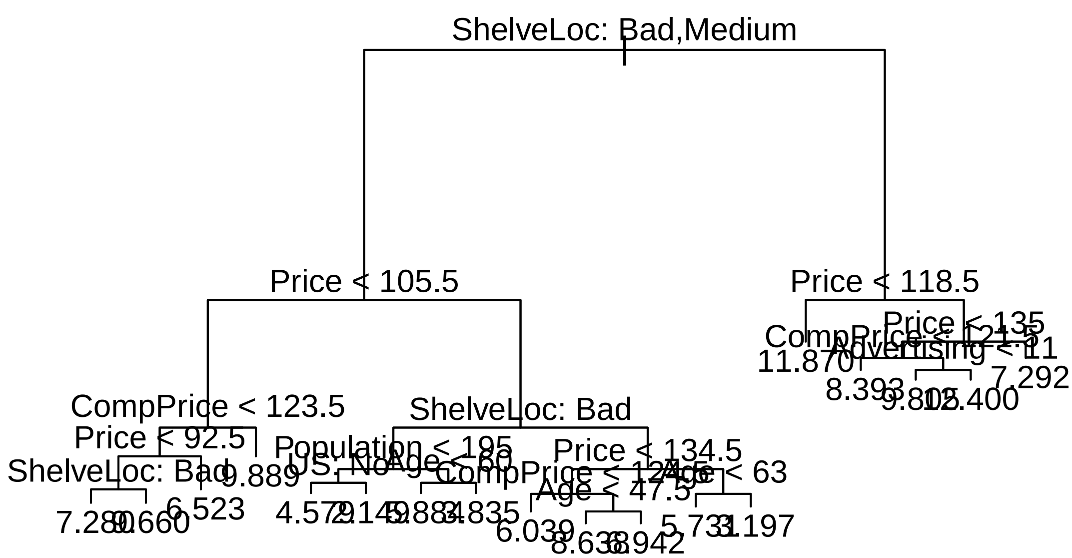
```r
set.seed(8)
model1 = tree(Sales~., data = train)
mean((predict(model1, test) - test$Sales)^2)
```

```
## [1] 4.454896
```

The test error rate is 4.454.

The tree is plotted below.

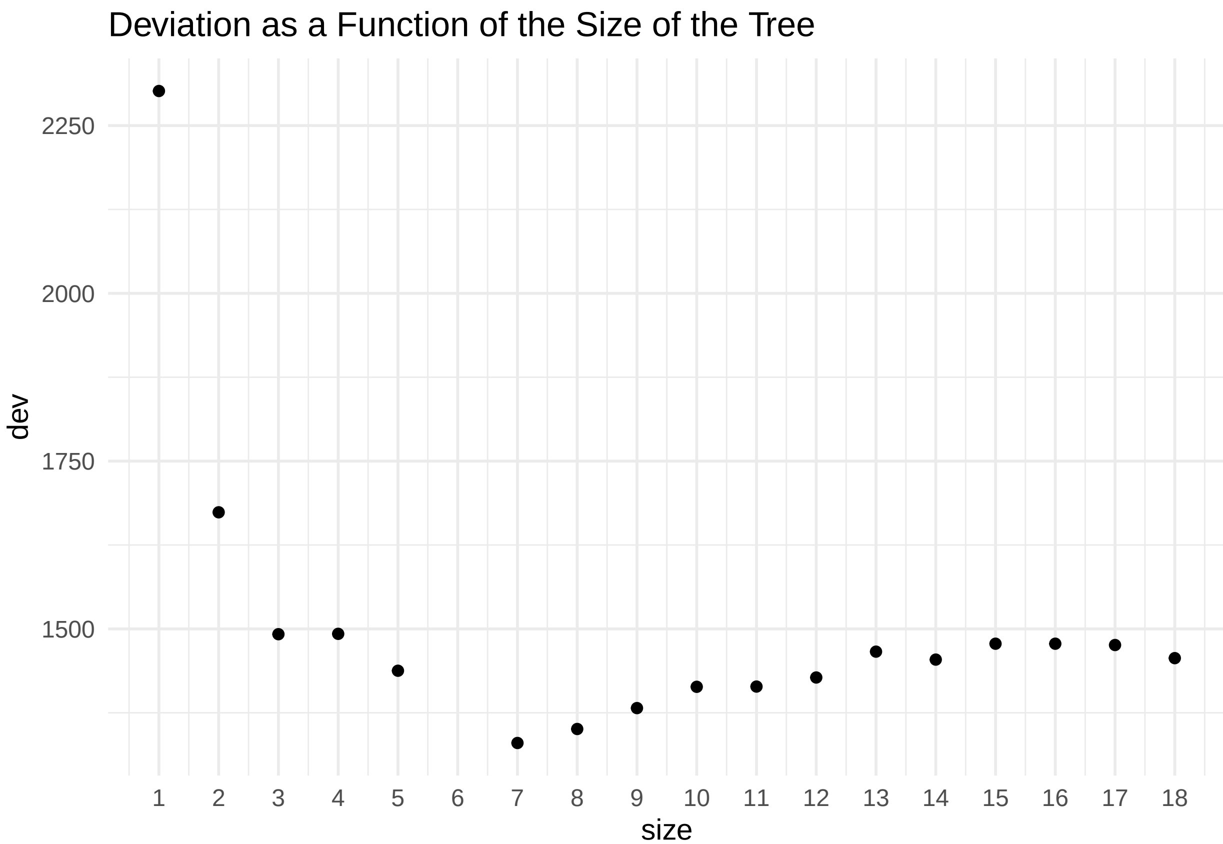
```r
plot(model1)
text(model1, pretty = 0)
```



It appears to be that `ShelveLoc` and `Price` are important indicators of `Sales`.

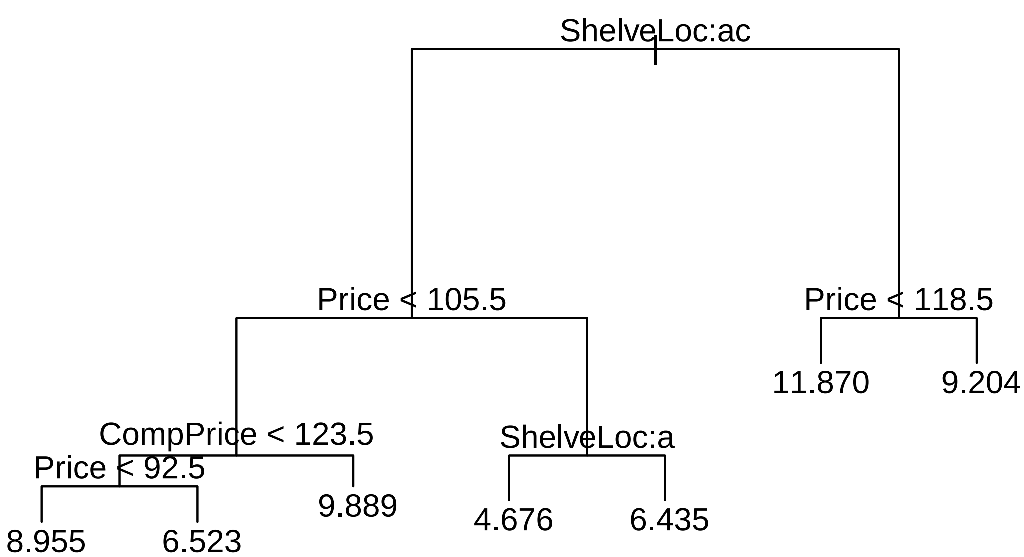
(c) Use cross-validation to determine the optimal level of tree complexity. Does pruning the tree improve the test error rate?

```r
set.seed(8)
model1_cv = cv.tree(model1)
model1_cv_df = data.frame(size = model1_cv$size,
                          dev = model1_cv$dev)
ggplot(model1_cv_df, aes(x = size, y = dev)) + geom_point() +
  scale_x_continuous(breaks = 1:18) +
  ggtitle("Deviation as a Function of the Size of the Tree") +
  theme_minimal()
```



The optimal level of tree complexity is 7 trees. Now, does pruning help?

```
model1_prune = prune.tree(model1, best = 7)
plot(model1_prune)
text(model1_prune)
```



Above shows the tree diagram. Only two variables are used: `ShelveLoc` and `Price`.

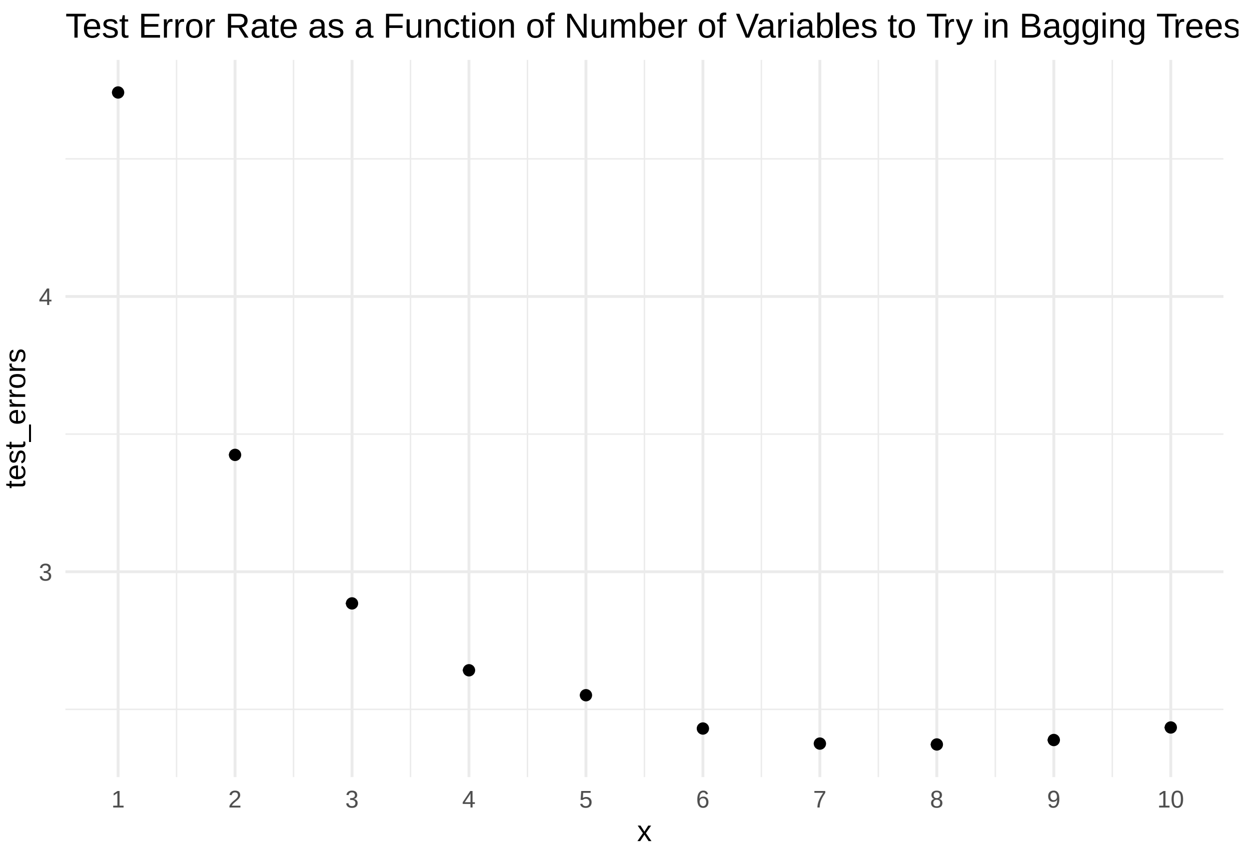
```
mean((predict(model1_prune, test) - test$Sales)^2)
```

```
## [1] 4.947714
```

The test error rate before pruning is 4.454. After pruning, it is 4.947714. Pruning the tree did not help improve the test error rate.

(d) Use the bagging approach in order to analyze this data. What test error rate was obtained? use the `importance()` function to determine which variables are most important.

```
set.seed(8)
test_errors = c()
for(i in 1:10){
  model2 = randomForest(Sales~., data = train,
                        mtry = i, importance = TRUE)
  test_errors = c(test_errors, (mean((predict(model2, test) - test$Sales)^2)))
}
test_errors_df = data.frame(x = 1:10, test_errors)
ggplot(test_errors_df, aes(x = x, y = test_errors)) + geom_point() +
  scale_x_continuous(breaks = 1:10) +
  ggtitle("Test Error Rate as a Function of Number of Variables to Try in Bagging Trees") +
  theme_minimal()
```



The best number of variables to try is 8.

```
model2best = randomForest(Sales~., data = train, mtry = 8, importance = TRUE)
importance(model2best)
```

```
##               %IncMSE IncNodePurity
## CompPrice   24.301928     187.70329
## Income      11.510839     138.32149
## Advertising 17.409891     133.58752
## Population  -1.806747      70.14546
```

```
## Price        61.432981     660.22020
## ShelveLoc    72.335437     705.84675
## Age          22.185591     204.10714
## Education     1.018820      60.23388
## Urban        -0.924055      10.03622
## US            5.088106      15.15826
```

```
test_errors = c(test_errors, (mean((predict(model2, test) - test$Sales)^2)))
```

According to this, if `Price` is removed from the model creation, then there is a 61% decrease in the accuracy of out of bag predictions. Likewise, if `ShelveLoc` is removed, then there is a 72% decrease in the accuracy of out of bag prediction. Using this bagged model, the test set MSE is

```
mean((predict(model2best, test) - test$Sales)^2)
```

```
## [1] 2.384532
```

This test error rate is lower than when a regular regression tree was created as well as after it was pruned.

(e) Use random forests to analyze this data. What test error rate was obtained? Use the `importance()` function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
set.seed(8)
test_errors = c()
for(i in 1:10){
  model3 = randomForest(Sales~., data = train,
                         mtry = i, ntree = 3, importance = TRUE)
  test_errors = c(test_errors, (mean((predict(model3, test) - test$Sales)^2)))
}
test_errors
```

```
##  [1] 5.960655 5.470976 4.673021 4.056446 3.999369 3.740645 2.951587
##  [8] 3.348464 3.312095 3.757623
```

As m increases, or the number of the variables considered at each split, the test error rate decreased until $m = 7$. Using the parameter $m = 7$,

```
model3best = randomForest(Sales~., data = train,
                           mtry = 7, ntree = 3, importance = TRUE)
importance(model3best)
```

```
##                %IncMSE IncNodePurity
## CompPrice     6.861041    212.264378
## Income        1.733153    114.597154
## Advertising   2.748178    118.741756
## Population   -1.193387     70.145861
## Price        16.091595    573.077267
## ShelveLoc    30.182280    684.999265
## Age           4.978486    151.967644
## Education    -1.379607     41.706894
## Urban        -1.435928     23.418042
## US           -1.206316      4.297039
```

`ShelveLoc` and `Price` continue to be important predictors for `Sales`.

Question 9: This problem involves the OJ dataset which is part of the ISLR package.

(a) Create a training set containing a random sample of 800 observations and a test set containing the remaining observations.

```
df = OJ
set.seed(800)
indices = sample(1:nrow(df), size = 800)
train = df[indices,]
test = df[-indices,]
```

(b) Fit a tree to the training data, with `Purchase` as the response and the other variables except for `Buy` as predictors. Use the `summary()` function to produce summary statistics about the tree and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
model4 = tree(Purchase ~ ., data = train)
summary(model4)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7519 = 596.2 / 793
## Misclassification error rate: 0.1538 = 123 / 800
```

Valuable variables that can predict `Purchase` are `LoyalCH`, `PriceDiff` and `ListPriceDiff`. The training error rate is 0.153. The tree has 7 terminal nodes.

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes and interpret the information displayed.

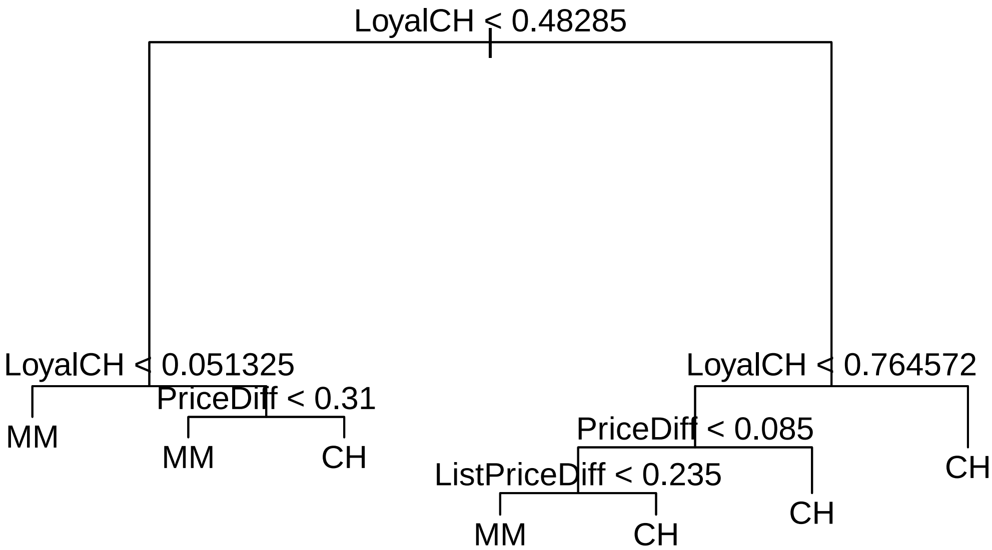
```
model4
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1067.00 CH ( 0.61375 0.38625 )
##    2) LoyalCH < 0.48285 305  333.40 MM ( 0.23607 0.76393 )
##      4) LoyalCH < 0.051325 68   18.05 MM ( 0.02941 0.97059 ) *
##      5) LoyalCH > 0.051325 237  287.70 MM ( 0.29536 0.70464 )
##       10) PriceDiff < 0.31 183  194.70 MM ( 0.22404 0.77596 ) *
##       11) PriceDiff > 0.31 54   74.56 CH ( 0.53704 0.46296 ) *
##    3) LoyalCH > 0.48285 495  424.50 CH ( 0.84646 0.15354 )
##      6) LoyalCH < 0.764572 245  285.50 CH ( 0.73061 0.26939 )
##       12) PriceDiff < 0.085 87  120.50 MM ( 0.48276 0.51724 )
##         24) ListPriceDiff < 0.235 57   71.10 MM ( 0.31579 0.68421 ) *
##         25) ListPriceDiff > 0.235 30   30.02 CH ( 0.80000 0.20000 ) *
##       13) PriceDiff > 0.085 158  123.80 CH ( 0.86709 0.13291 ) *
##      7) LoyalCH > 0.764572 250   83.97 CH ( 0.96000 0.04000 ) *
```

The terminal node to be interpreted here is node 4. The split criterion is `LoyalCH < 0.051325`. There are 68 observations in this branch with a deviation of 18.05. $\approx 3\%$ of the observations are classified as `MM` whereas the other $\approx 97\%$ are classified as `CH`.

(d) Create a plot of the tree and interpret the results.

```r
plot(model4)
text(model4)
```



The topmost important criterion is if `LoyalCH` is less than 0.48285. Further classification is aided by additional segmentation of `LoyalCH` followed by `PriceDiff`.

(e) Predict the response on the test data and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```r
predictions = predict(model4, test, type = "class")
table(predictions, test$Purchase)
```

```
##
## predictions  CH  MM
##          CH 137  30
##          MM  25  78
```

According to this model, a good number of correct predictions are made. The test error rate is

```r
(nrow(test) - sum(diag(table(predictions, test$Purchase)))) / nrow(test)
```

```
## [1] 0.2037037
```

This is somewhat high; it can be improved.

(f) Apply the `cv.tree()` function to the training set in order to determine the optimal tree size.

```r
model4_cv = cv.tree(model4)
model4_cv
```

```
## $size
## [1] 7 6 5 4 3 2 1
##
## $dev
## [1]  724.7805  699.9392  696.8692  720.8456  733.4632  768.1672 1067.6540
##
## $k
## [1]      -Inf  18.39377  19.38334  27.65907  41.15852  55.03483 309.39449
##
## $method
```

```
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

The number of tree sizes considered are from 1 to 7.

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
model4_cv_df = data.frame(size = model4_cv$size,
                          dev = model4_cv$dev)
ggplot(model4_cv_df, aes(x = size, y = dev)) + geom_point() +
  ggtitle("Cross Validation Error as a Function of Tree Size") +
  theme_minimal()
```

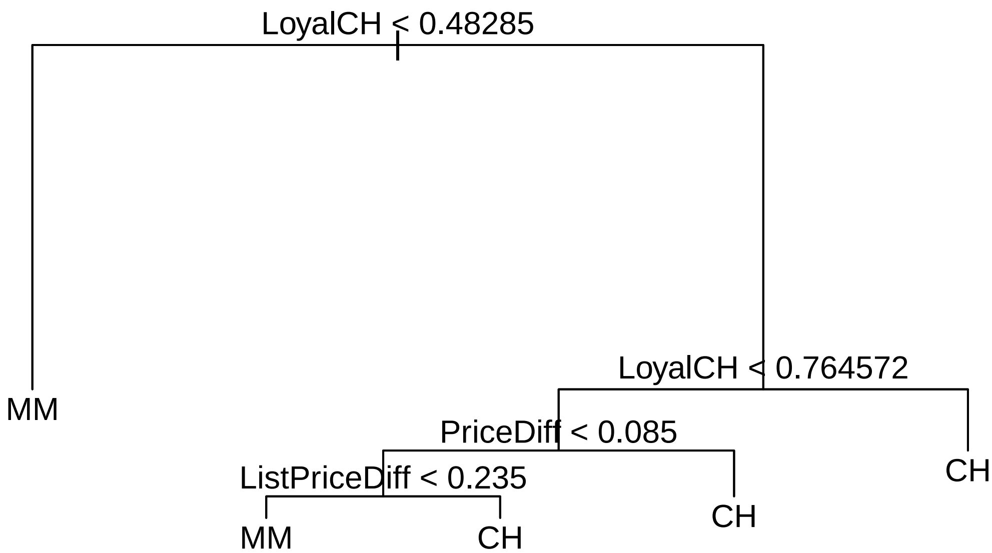
Misclassification error rate decreases rapidly from a size of 1 to a size of 2 and continue to do so until 5 trees are used and then increases.

(h) Which tree size corresponses to the lowest cross-validated classification error rate?

The optimal tree size is 5 which had the lowest misclassification rate.

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
model4_pruned = prune.misclass(model4, best = 5)
plot(model4_pruned)
text(model4_pruned)
```



(j) Compare the training error rates between the pruned and unpruned trees. Which is higher?

```r
summary(model4)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7519 = 596.2 / 793
## Misclassification error rate: 0.1538 = 123 / 800
```

```r
summary(model4_pruned)
```

```
##
## Classification tree:
## snip.tree(tree = model4, nodes = 2L)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff"
## Number of terminal nodes:  5
## Residual mean deviance:  0.8079 = 642.3 / 795
## Misclassification error rate: 0.1588 = 127 / 800
```

The training error rate is slightly higher for the pruned tree.

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

```r
predictions2 = predict(model4, test, type = "class")
(nrow(test) - sum(diag(table(predictions2, test$Purchase)))) / nrow(test)
```

```
## [1] 0.2037037
```

The test error made using the pruned tree is 0.203 whereas before pruning it is also 0.203. It can be seen that pruning the tree made no difference.

Question 10: Now using boosting to predict `Salary` in the `Hitters` dataset.

(a) Remove the observations for whom the salary information is unknown and then log-transform the salaries.

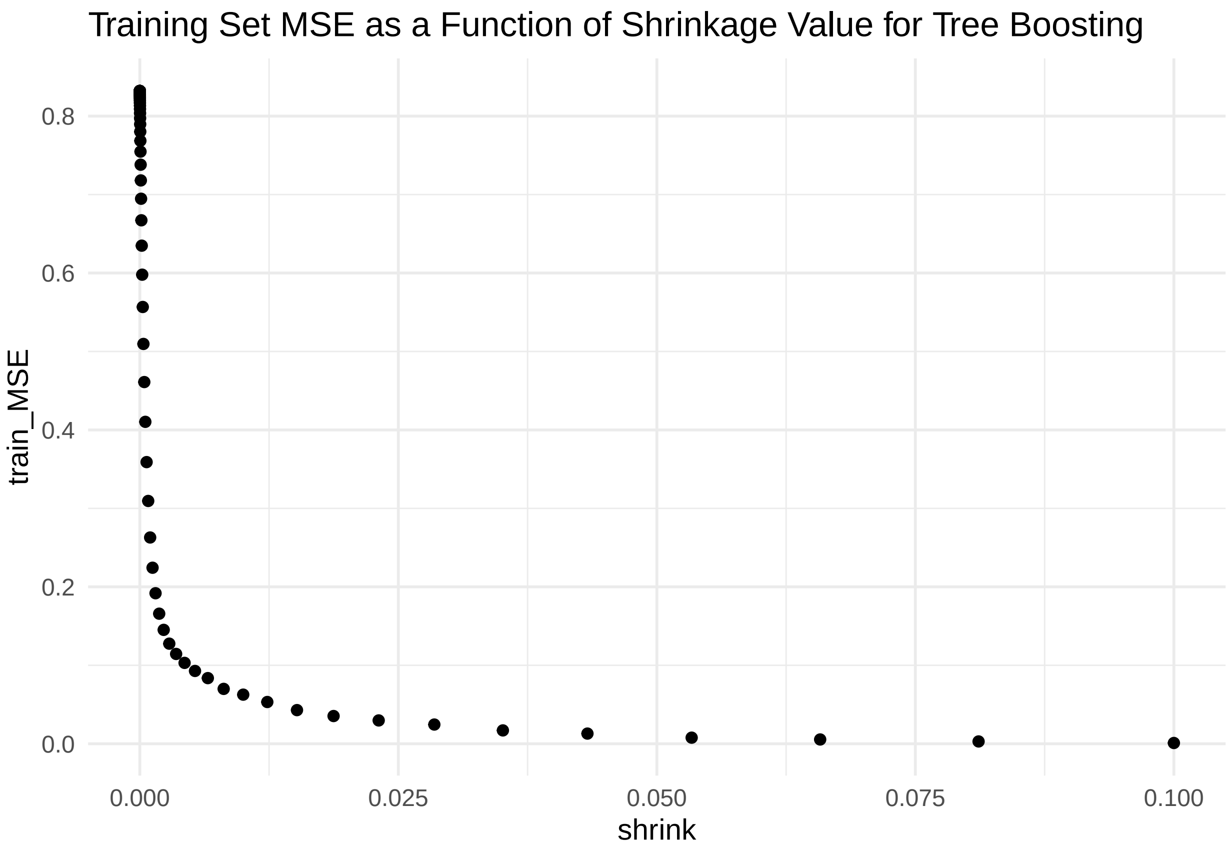
```
df = Hitters
df = df[!is.na(df$Salary),]
df$Salary = log(df$Salary)
```

(b) Create a training set consiting of the first 200 observations and a test set consisting of the remaining observations.

```
indices = 1:200
train = df[indices,]
test = df[-indices,]
```

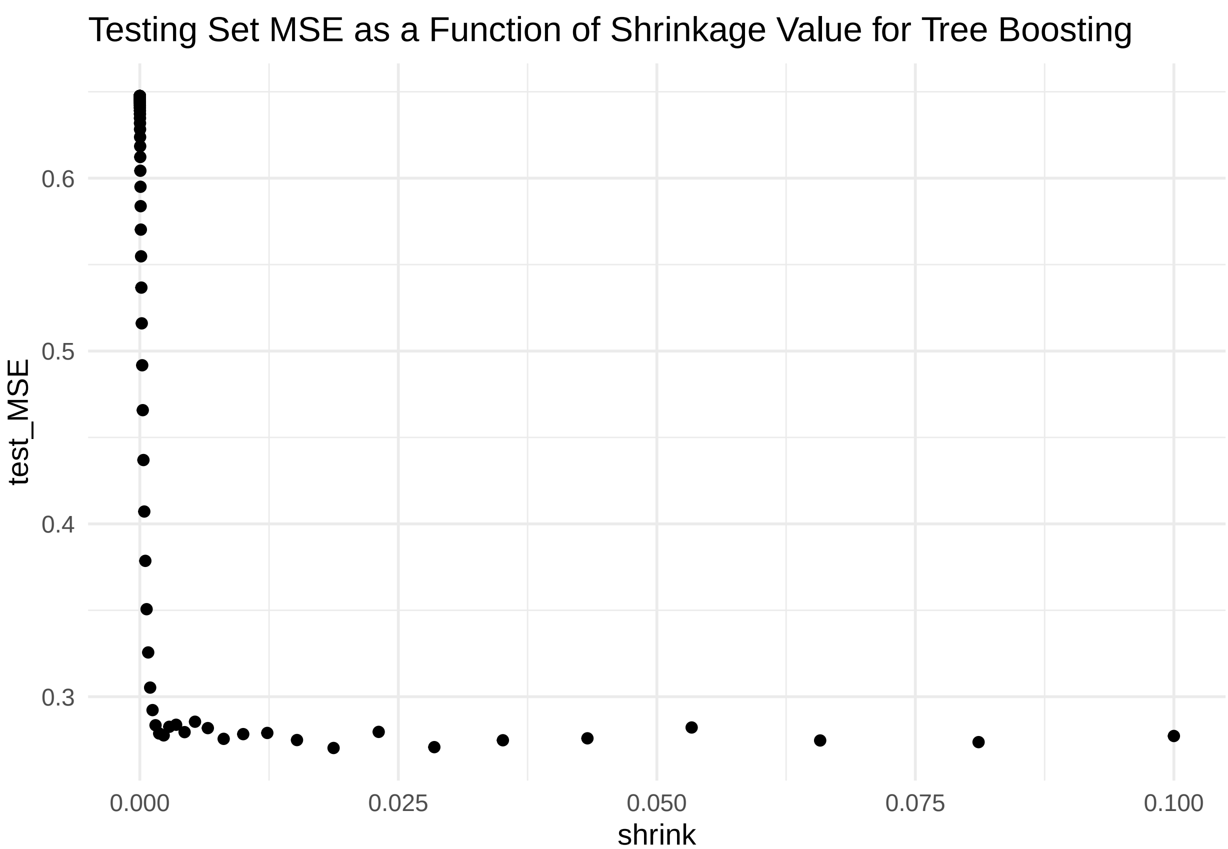
(c) Perform boosting on the training set with $1,000$ trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
set.seed(1000)
shrink = 10^seq(-10, -1, length = 100)
train_MSE = c()
for(i in shrink){
  model5 = gbm(Salary ~ ., data = train,
               distribution = "gaussian", n.trees = 1000,
               interaction.depth = 4, shrinkage = i)
  train_MSE = c(train_MSE,
                mean((predict(model5, train, n.tree = 1000) - train$Salary)^2))
}
train_MSE_df = data.frame(shrink, train_MSE)
ggplot(train_MSE_df, aes(x = shrink, y = train_MSE)) + geom_point() +
  ggtitle("Training Set MSE as a Function of Shrinkage Value for Tree Boosting") +
  theme_minimal()
```



(d) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```
set.seed(1000)
shrink = 10^seq(-10, -1, length = 100)
test_MSE = c()
for(i in shrink){
  model5 = gbm(Salary ~ ., data = train,
                distribution = "gaussian", n.trees = 1000,
                interaction.depth = 4, shrinkage = i)
  test_MSE = c(test_MSE,
                mean((predict(model5, test, n.tree = 1000) - test$Salary)^2))
}
test_MSE_df = data.frame(shrink, test_MSE)
ggplot(test_MSE_df, aes(x = shrink, y = test_MSE)) + geom_point() +
  ggtitle("Testing Set MSE as a Function of Shrinkage Value for Tree Boosting") +
  theme_minimal()
```



Unlike the training set MSE, the test set MSE is not continuously decreasing as shrinking increases. In fact, the lowest test MSE occurs when shrinkage is

```
shrink[which.min(test_MSE)]
```

```
## [1] 0.01873817
```

and the test set MSE is

```
min(test_MSE)
```

```
## [1] 0.2703327
```

(e) Compare the test MSE of boosting to the test MSE that results from applying two different regression approaches.

```
train_matrix = model.matrix(Salary~., data = train)
test_matrix = model.matrix(Salary~., data = train)

model5ridge = glmnet(train_matrix, train$Salary,
                     alpha = 0,
                     lambda = 10^seq(10, -2, length = 1000),
                     thresh = 1e-12)
model5_ridge_cv = cv.glmnet(train_matrix, train$Salary,
                            alpha = 0,
                            lambda = 10^seq(10, -2, length = 1000),
                            thresh = 1e-12)
ridge_mse = mean((predict(model5ridge, s = model5_ridge_cv$lambda.min,
                          newx = test_matrix) - test$Salary)^2)
```

```
model5lasso = glmnet(train_matrix, train$Salary,
                     alpha = 1,
                     lambda = 10^seq(10, -2, length = 1000),
                     thresh = 1e-12)
model5_lasso_cv = cv.glmnet(train_matrix, train$Salary,
                            alpha = 1,
                            lambda = 10^seq(10, -2, length = 1000),
                            thresh = 1e-12)
lasso_mse = mean((predict(model5lasso, s = model5_lasso_cv$lambda.min,
                          newx = test_matrix) - test$Salary)^2)
```

```
paste("Ridge MSE", ridge_mse)
```

```
## [1] "Ridge MSE 1.06192052833569"
```

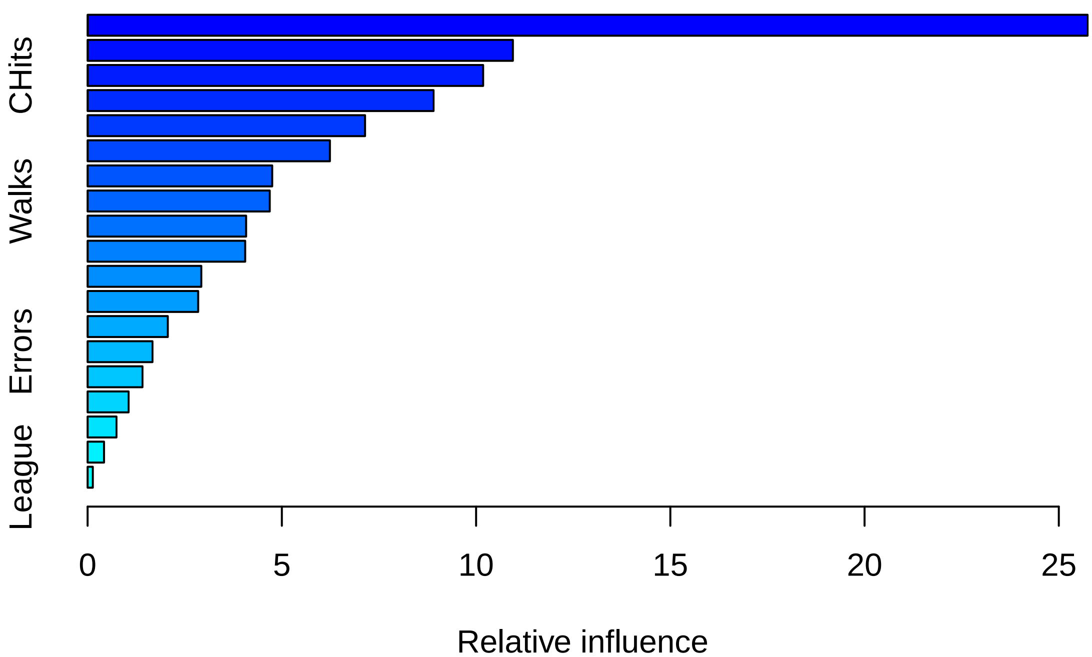
```
paste("Lasso MSE", lasso_mse)
```

```
## [1] "Lasso MSE 1.11324439396726"
```

The test set of boosting is lower than the ones found using ridge and lasso regression.

(f) Which variables appear to be the most important predictors in the boosted model?

```
set.seed(10)
model5best = gbm(Salary ~ ., data = train, distribution = "gaussian",
                 n.trees = 1000, shrinkage = shrink[which.min(test_MSE)])
summary(model5best)
```




```
##                 var    rel.inf
## CAtBat       CAtBat 25.7398127
## CWalks       CWalks 10.9470933
## CHits         CHits 10.1811801
## CRBI           CRBI  8.9044855
## Years         Years  7.1417590
## CHmRun       CHmRun  6.2373358
```

```
## Hits           Hits  4.7515929
## Walks         Walks  4.6881372
## PutOuts     PutOuts  4.0804835
## CRuns         CRuns  4.0560154
## RBI             RBI  2.9262507
## AtBat         AtBat  2.8463555
## HmRun         HmRun  2.0639382
## Errors       Errors  1.6695639
## Assists     Assists  1.4118036
## Runs           Runs  1.0547873
## Division   Division  0.7427251
## NewLeague NewLeague  0.4229875
## League       League  0.1336928
```

The variables that appear to be the important predictors are `CAtBat`, `CWalks` and `CHits`.

(g) Now apply bagging to the training set. What is the test set MSE for this approach?

```r
set.seed(10)
model5bagged = randomForest(Salary ~ ., data = train,
                            mtry = 15, ntree = 100)
predictions = predict(model5bagged, test)
mean((predictions - test$Salary)^2)
```

```
## [1] 0.2233214
```

When bagging, the test set MSE is 0.223; this is lower than when boosting is used instead.

Question 11: This question uses the `Caravan` dataset.

(a) Create a training set consisting of the first 1,000 observations and a test set consisting of the remaining observations.

```r
set.seed(11)
df = Caravan
df$Purchase = ifelse(df$Purchase == "Yes", 1, 0)
indices = 1:1000
train = df[indices,]
test = df[-indices,]
```

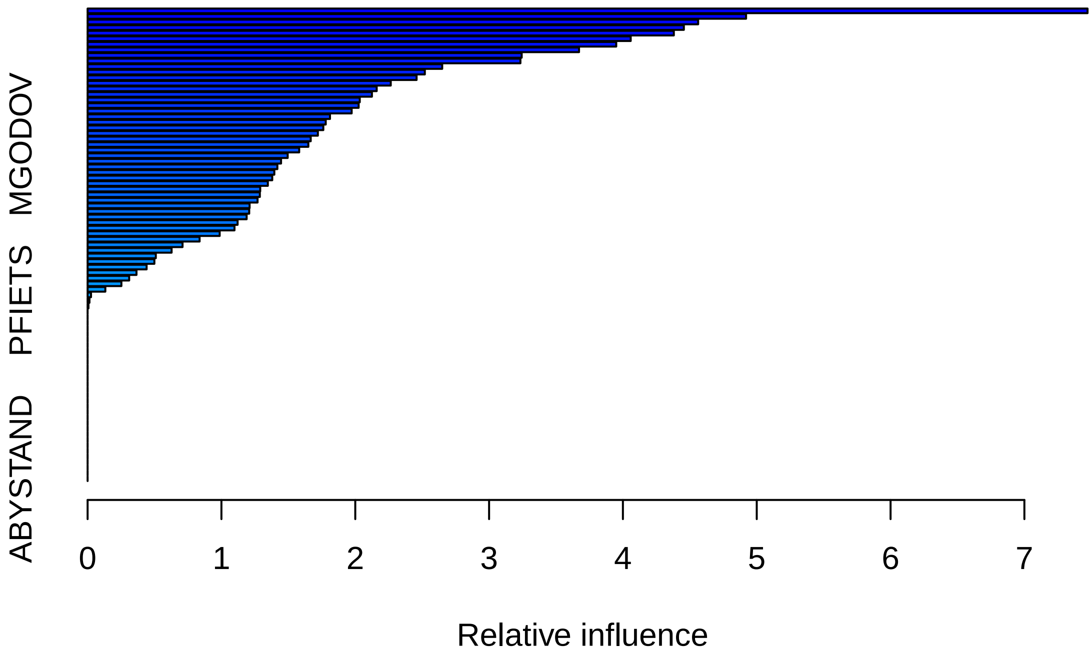
(b) Fit a boosting model to the training set with `Purchase` as the response and the other variables as predictors. Use 1,000 trees and a shrinkage of 0.01. Which predictors appear to be the most important?

```r
set.seed(11)
model6 = gbm(Purchase ~ ., data = train, distribution = "bernoulli",
             n.trees = 1000, interaction.depth = 4, shrinkage = 0.01)
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 50: PVRAAUT has no variation.
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 71: AVRAAUT has no variation.
```

```r
summary(model6)
```




```
##               var     rel.inf
## PPERSAUT PPERSAUT 7.472017283
## MGODGE     MGODGE 4.920268524
## PBRAND     PBRAND 4.562337369
## MKOOPKLA MKOOPKLA 4.455474976
## MOPLHOOG MOPLHOOG 4.380527334
## MOSTYPE   MOSTYPE 4.058244281
## MINK3045 MINK3045 3.950103516
## MGODPR     MGODPR 3.672167279
## MBERMIDD MBERMIDD 3.243664884
## MAUT2       MAUT2 3.234222424
## MBERARBG MBERARBG 2.649691655
## MSKB1       MSKB1 2.519943184
## MSKC         MSKC 2.458311336
## MOPLMIDD MOPLMIDD 2.265387824
## MSKA         MSKA 2.160331133
## PWAPART   PWAPART 2.124682559
## MBERARBO MBERARBO 2.033834610
## MINKM30   MINKM30 2.025470388
## MFWEKIND MFWEKIND 1.973111208
## MRELOV     MRELOV 1.811226920
## MBERHOOG MBERHOOG 1.779508314
## MAUT1       MAUT1 1.761319430
## MRELSA     MRELSA 1.720751942
## MRELGE     MRELGE 1.667203409
## MGODOV     MGODOV 1.649518856
## MINK7512 MINK7512 1.580946746
## MINKGEM   MINKGEM 1.495104835
## MFALLEEN MFALLEEN 1.445414183
## MSKB2       MSKB2 1.417953954
## MZFONDS   MZFONDS 1.395215715
## MAUT0       MAUT0 1.379730933
## MGODRK     MGODRK 1.346956354
```

```
## MINK4575 MINK4575 1.289878285
## MFGEKIND MFGEKIND 1.287102544
## ABRAND     ABRAND 1.269977976
## MGEMLEEF MGEMLEEF 1.210600254
## MHHUUR     MHHUUR 1.207518555
## MGEMOMV   MGEMOMV 1.188485795
## MSKD         MSKD 1.120963005
## MHKOOP     MHKOOP 1.097480654
## MZPART     MZPART 0.987005458
## APERSAUT APERSAUT 0.836917534
## MOSHOOFD MOSHOOFD 0.709339641
## MOPLLAAG MOPLLAAG 0.627775856
## MBERZELF MBERZELF 0.509784887
## PMOTSCO   PMOTSCO 0.498942450
## MBERBOER MBERBOER 0.440908428
## PLEVEN     PLEVEN 0.365185344
## PBYSTAND PBYSTAND 0.310737284
## MINK123M MINK123M 0.252407865
## MAANTHUI MAANTHUI 0.132457819
## ALEVEN     ALEVEN 0.025485709
## PFIETS     PFIETS 0.013999110
## PAANHANG PAANHANG 0.006402191
## PWABEDR   PWABEDR 0.000000000
## PWALAND   PWALAND 0.000000000
## PBESAUT   PBESAUT 0.000000000
## PVRAAUT   PVRAAUT 0.000000000
## PTRACTOR PTRACTOR 0.000000000
## PWERKT     PWERKT 0.000000000
## PBROM       PBROM 0.000000000
## PPERSONG PPERSONG 0.000000000
## PGEZONG   PGEZONG 0.000000000
## PWAOREG   PWAOREG 0.000000000
## PZEILPL   PZEILPL 0.000000000
## PPLEZIER PPLEZIER 0.000000000
## PINBOED   PINBOED 0.000000000
## AWAPART   AWAPART 0.000000000
## AWABEDR   AWABEDR 0.000000000
## AWALAND   AWALAND 0.000000000
## ABESAUT   ABESAUT 0.000000000
## AMOTSCO   AMOTSCO 0.000000000
## AVRAAUT   AVRAAUT 0.000000000
## AAANHANG AAANHANG 0.000000000
## ATRACTOR ATRACTOR 0.000000000
## AWERKT     AWERKT 0.000000000
## ABROM       ABROM 0.000000000
## APERSONG APERSONG 0.000000000
## AGEZONG   AGEZONG 0.000000000
## AWAOREG   AWAOREG 0.000000000
## AZEILPL   AZEILPL 0.000000000
## APLEZIER APLEZIER 0.000000000
## AFIETS     AFIETS 0.000000000
## AINBOED   AINBOED 0.000000000
## ABYSTAND ABYSTAND 0.000000000
```

The predictors that are most important are: `PPERSAUT`, `MGODGE`, `PBRAND`, `MKOOPKLA` , `MOPLHOOG` and `MOSTYPE`.

(c) Use the boosting model to predict the response on the test data. Predict that a person will make a purchase if the estimated probability is greater than 20%. Form a confusion matrix. What fraction of the people predicted to make a purchase do in fact make one? How does this compare with the results obtained from applying KNN or logistic regression to this dataset?

```
probs = predict(model6, test, n.tree = 1000, type = "response")
predictions = ifelse(probs > 0.2, 1, 0)
table(predictions, test$Purchase)
```

```
##
## predictions    0    1
##           0 4346  254
##           1  187   35
```

The fraction of the people predicted to make a purchase that do in fact make one is

```
4346 / nrow(test)
```

```
## [1] 0.9012858
```

Not bad. When using KNN,

```
set.seed(11)
model6knn = knn(train, test, cl = train$Purchase, k = 5)
table(model6knn, test$Purchase)
```

```
##
## model6knn    0    1
##         0 4504  285
##         1   29    4
```

The fraction of the people predicted to make a purchase goes up when using a KNN model. When using logistic regression,

```
set.seed(11)
model6log = glm(Purchase ~ ., data = train, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
probs = predict(model6log, test, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
predictions = ifelse(probs > 0.2, 1, 0)
table(predictions, test$Purchase)
```

```
##
## predictions    0    1
##           0 4183  231
##           1  350   58
```

The fraction of the people predicted to make a purchase goes down when using a logistic regression model.

The best predictions are made using the KNN model.

Question 12: Apply boosting, bagging and random forests to any dataset. Be sure to fit the models on a training set and to evaluate their performance on a test set. How accurate are the results compared to simple methods like linear or logistic regression? Which of these appraoches yields the best performance?

```
df = biopsy
df = na.omit(df)
set.seed(12)
df = df %>% subset(select = -ID)
df$class = ifelse(df$class == "malignant", 1, 0)
indices = sample(1:nrow(df), size = 0.7*nrow(df))
train = df[indices,]
test = df[-indices,]
```

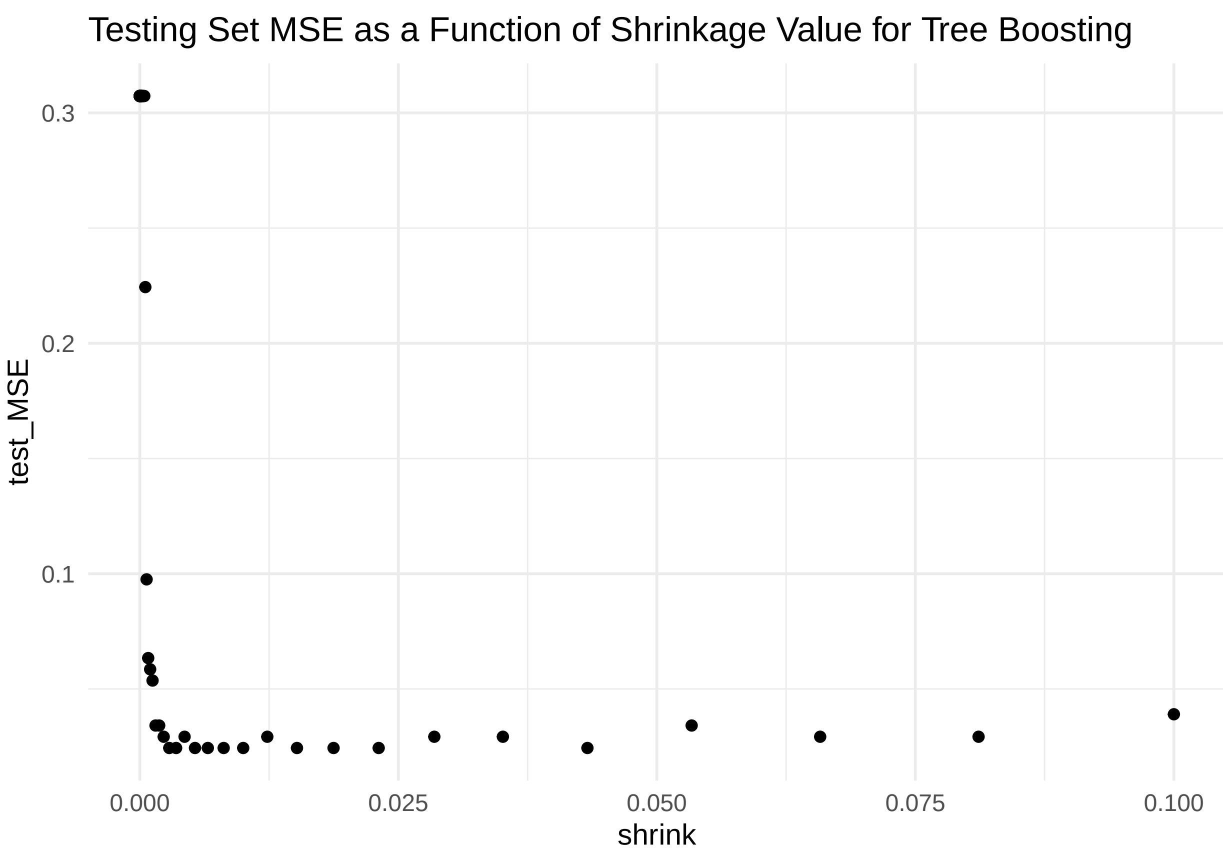
Base Tree Model:

```
set.seed(12)
model_base = tree(class ~ ., train)
predictions = predict(model_base, test, class = "response")
probs = ifelse(predictions > 0.5, 1, 0)
mse_base = (nrow(test) - sum(diag(table(probs, test$class)))) / nrow(test)
mse_base
```

```
## [1] 0.04390244
```

Boosting:

```
set.seed(12)
shrink = 10^seq(-10, -1, length = 100)
test_MSE = c()
for(i in shrink){
  model_boost = gbm(class ~ ., data = train, distribution = "bernoulli",
                    n.trees = 1000, interaction.depth = 4, shrinkage = i)
  predictions = predict(model_boost, test, n.trees = 1000)
  probs = ifelse(predictions > 0.5, 1, 0)
  score = (nrow(test) - sum(diag(table(probs, test$class)))) / nrow(test)
  test_MSE = c(test_MSE, score)
}
test_MSE_df = data.frame(shrink, test_MSE)
ggplot(test_MSE_df, aes(x = shrink, y = test_MSE)) + geom_point() +
  ggtitle("Testing Set MSE as a Function of Shrinkage Value for Tree Boosting") +
  theme_minimal()
```



Using the best shrinkage value, the test set MSE for a boosting model is

```
mse_boost = min(test_MSE)
mse_boost
```

```
## [1] 0.02439024
```

Bagging:

```
set.seed(12)
train$class = as.factor(train$class)
test$class = as.factor(test$class)
model_bag = randomForest(class ~., data = train,
                          mtry = ncol(df)-1, importance = TRUE)
predictions = predict(model_bag, test,
                       n.trees = 1000, type = "response")
probs = ifelse(predictions == 1, 1, 0)
mse_bag = (nrow(test) - sum(diag(table(probs, test$class)))) / nrow(test)
mse_bag
```

```
## [1] 0.02926829
```

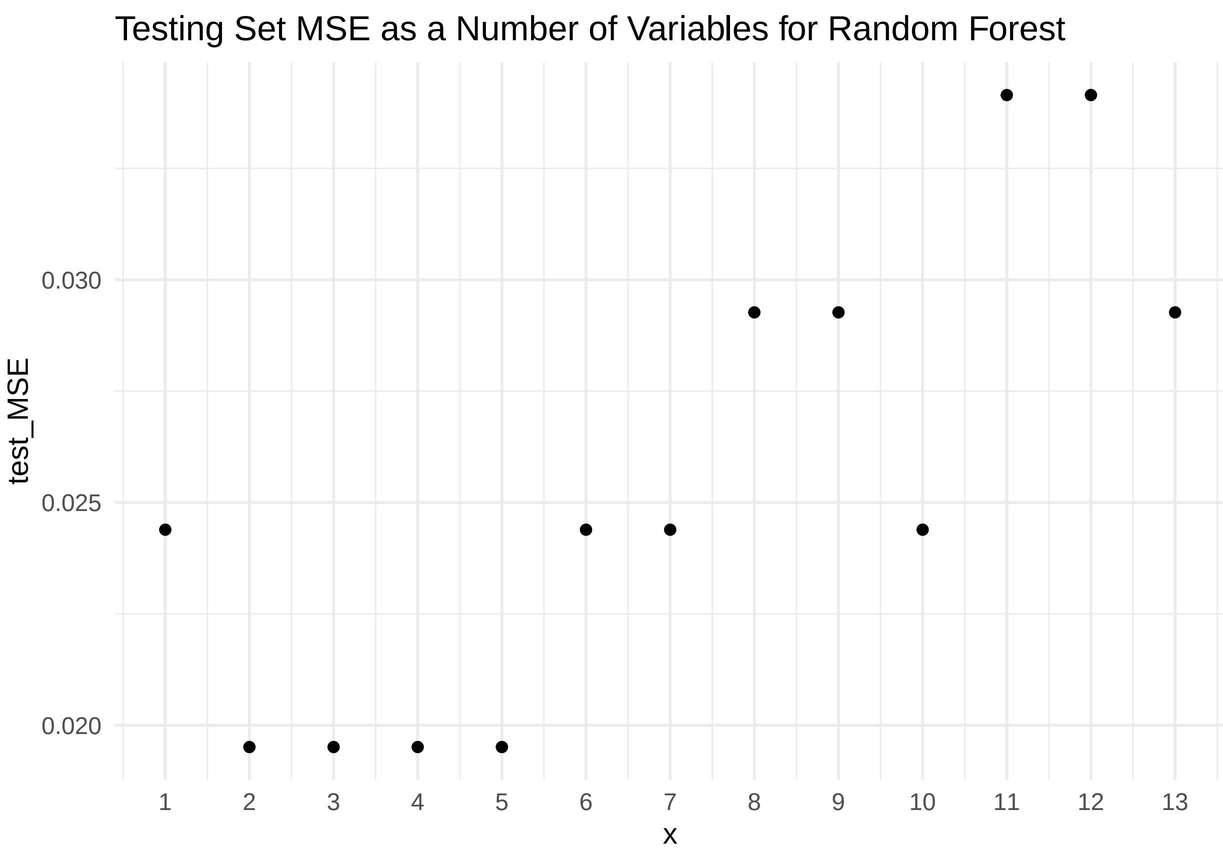
Random Forest:

```
set.seed(12)
test_MSE = c()
for(i in 1:13){
  model_rf = randomForest(class ~., data = train,
                           mtry = i, importance = TRUE)
  predictions = predict(model_rf, test,
```

```r
                          n.trees = 1000, type = "response")
  probs = ifelse(predictions == 1, 1, 0)
  score = (nrow(test) - sum(diag(table(probs, test$class)))) / nrow(test)
  test_MSE = c(test_MSE, score)
}
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within
## valid range

## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within
## valid range

## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within
## valid range

## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within
## valid range
```

```r
test_MSE_df = data.frame(x = 1:13, test_MSE)
ggplot(test_MSE_df, aes(x = x, y = test_MSE)) + geom_point() +
  scale_x_continuous(breaks = 1:13) +
  ggtitle("Testing Set MSE as a Number of Variables for Random Forest") +
  theme_minimal()
```



The best number of variables to use is

```r
which.min(test_MSE)
```

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

and the respective test set MSE is

```
mse_rf = min(test_MSE)
mse_rf
```

```
## [1] 0.0195122
```

Alternative, when using KNN, the test set MSE is

```
set.seed(12)
model_knn = knn(train, test, cl = train$class, k = 5)
mse_knn = (nrow(test) - sum(diag(table(probs, test$class)))) / nrow(test)
mse_knn
```

```
## [1] 0.02926829
```

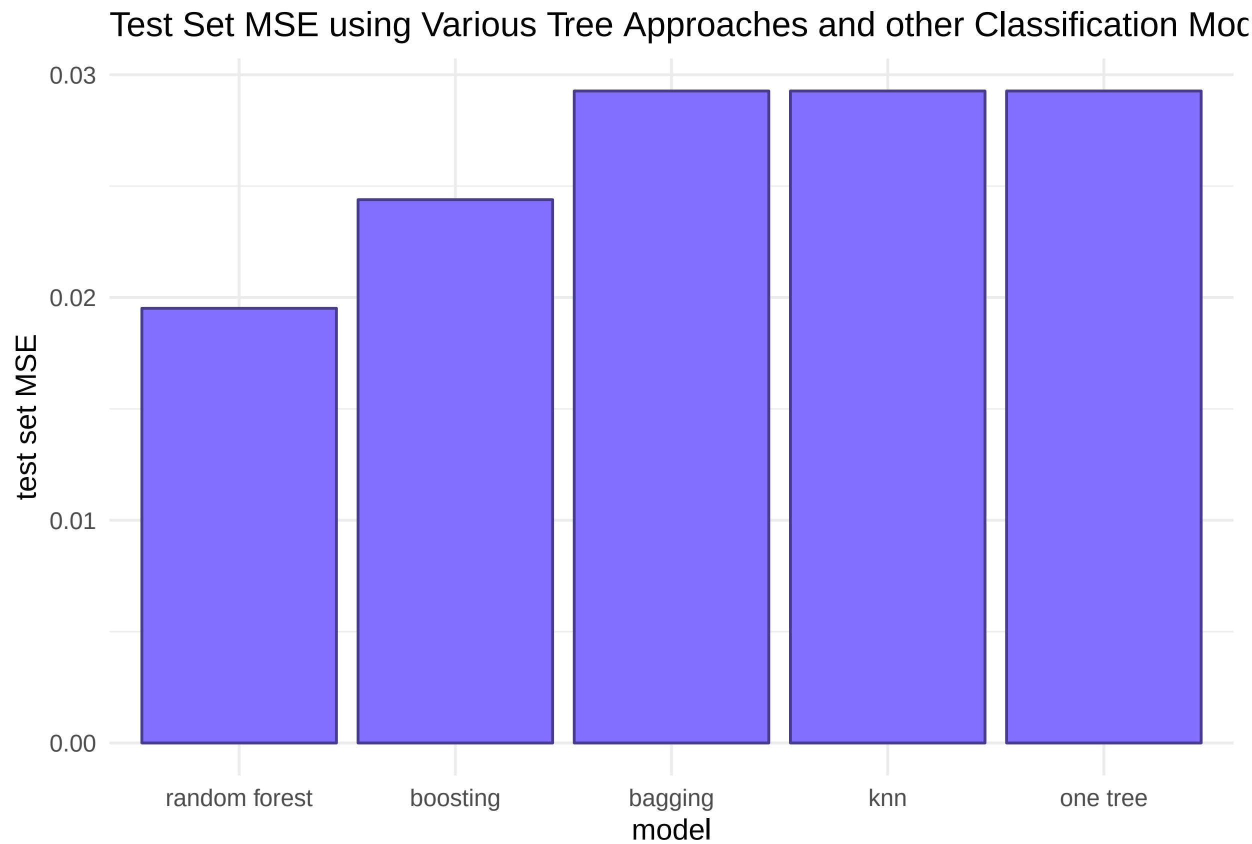
and when performing logistic regression, the test set MSE is

```
set.seed(12)
model_log = glm(class ~., data = train, family = "binomial")
probs = predict(model_log, test, type = "response")
predictions = ifelse(probs == 1, 1, 0)
mse_log = (nrow(test) - sum(diag(table(predictions, test$class)))) / nrow(test)
mse_log
```

```
## [1] 0.3073171
```

All the test set MSEs are plotted below. Logistic regression is omitted since its test set MSE is far above the rest.

```
mse_df = data.frame("model" = c("one tree", "boosting",
                                "bagging", "random forest", "knn"),
                    "mse" = c(mse_bag, mse_boost, mse_bag, mse_rf, mse_knn))
ggplot(mse_df, aes(x = reorder(model, mse), y = mse)) +
  geom_col(color = "slateblue4", fill = "slateblue1") +
  labs(x = "model", y = "test set MSE") +
  ggtitle("Test Set MSE using Various Tree Approaches and other Classification Models") +
  theme_minimal()
```



It is apparent that the best approach for classification in this problem is the random forest implementation. The KNN and one tree model performed similarly and logistic regression performed the worst (not shown).

All of the practice applied exercises in this document are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.