

# STA6703 SML HW8

Christopher Marais

## Question 3

$$E = 1 - \max_k(\hat{p}_{mk})$$

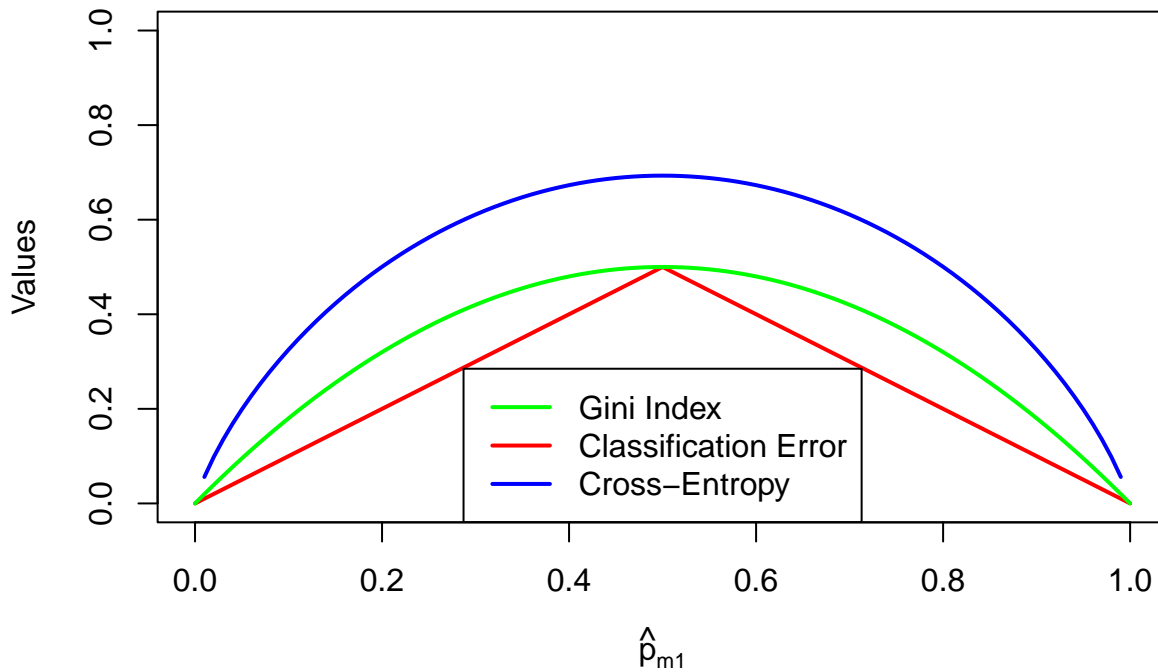
$$G = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk})$$

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk})$$

Figure 1: Classification Error (E), Gini Index (G), and Entropy (D)

```
# plot Gini Index, Classification Error and Cross-Entropy
p <- seq(0, 1, 0.01)
gini.index <- 2 * p * (1 - p)
class.error <- 1 - pmax(p, 1 - p)
cross.entropy <- - (p * log(p) + (1 - p) * log(1 - p))
df = data.frame(cbind(p, gini.index, class.error, cross.entropy))
{plot(1, type="n", main="Gini Index, Classification Error and Cross-Entropy",
      xlab=expression(hat(p)[m1]), ylab="Values", xlim=seq(0,1), ylim=c(0, 1))
lines(df$p,df$cross.entropy,col="blue",lwd=2)
lines(df$p,df$class.error,col="red",lwd=2)
lines(df$p,df$gini.index,col="green",lwd=2)
legend("bottom",
      legend=c("Gini Index", "Classification Error", "Cross-Entropy"),
      col=c("green", "red", "blue"),
      lty=c(1,1,1),
      lwd=c(2,2,2))}
```

## Gini Index, Classification Error and Cross-Entropy



### Question 8

a.)

```
# import data
library(ISLR)
library(caTools)
df_data = Carseats

# split data into training and testing sets
set.seed(0)
train_test_filter = sample.split(df_data$Sales, SplitRatio = 0.75)
df_train = subset(df_data, train_test_filter==TRUE)
df_test = subset(df_data, train_test_filter==FALSE)
```

b.)

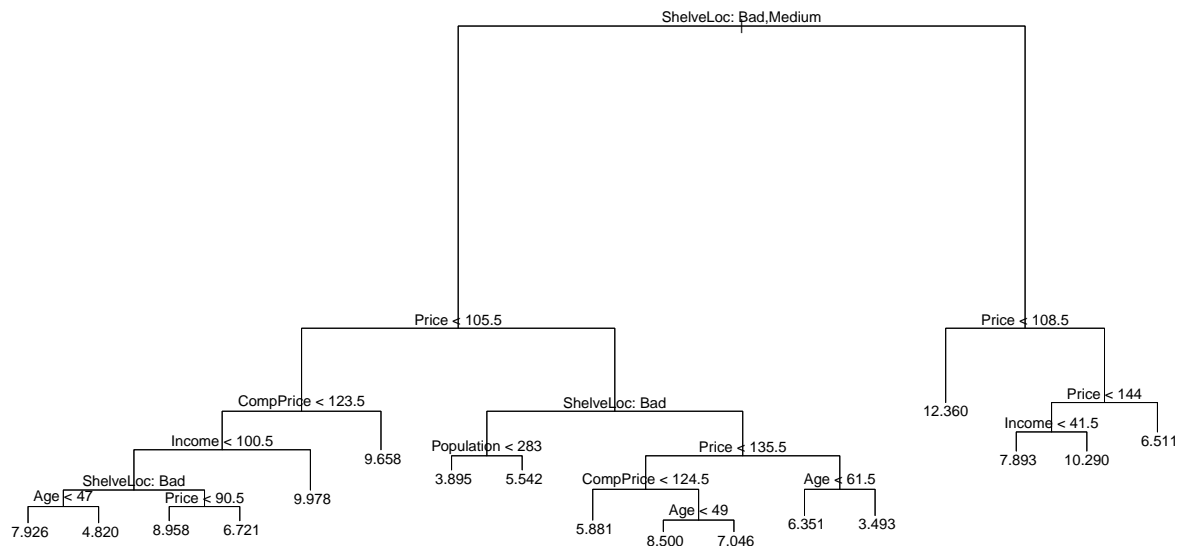
```
# import tree library
library(tree)

# create regression tree model
```

```
reg_tree_model = tree(Sales~.,data=df_train)
summary(reg_tree_model)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = df_train)
## Variables actually used in tree construction:
## [1] "ShelveLoc" "Price" "CompPrice" "Income" "Age"
## [6] "Population"
## Number of terminal nodes: 17
## Residual mean deviance: 2.351 = 665.5 / 283
## Distribution of residuals:
## Min. 1st Qu. Median Mean 3rd Qu. Max.
## -4.12200 -1.13100 -0.06688 0.00000 1.04400 4.59800
```

```
# plot tree
{plot(reg_tree_model)
text(reg_tree_model,pretty=0)}
```



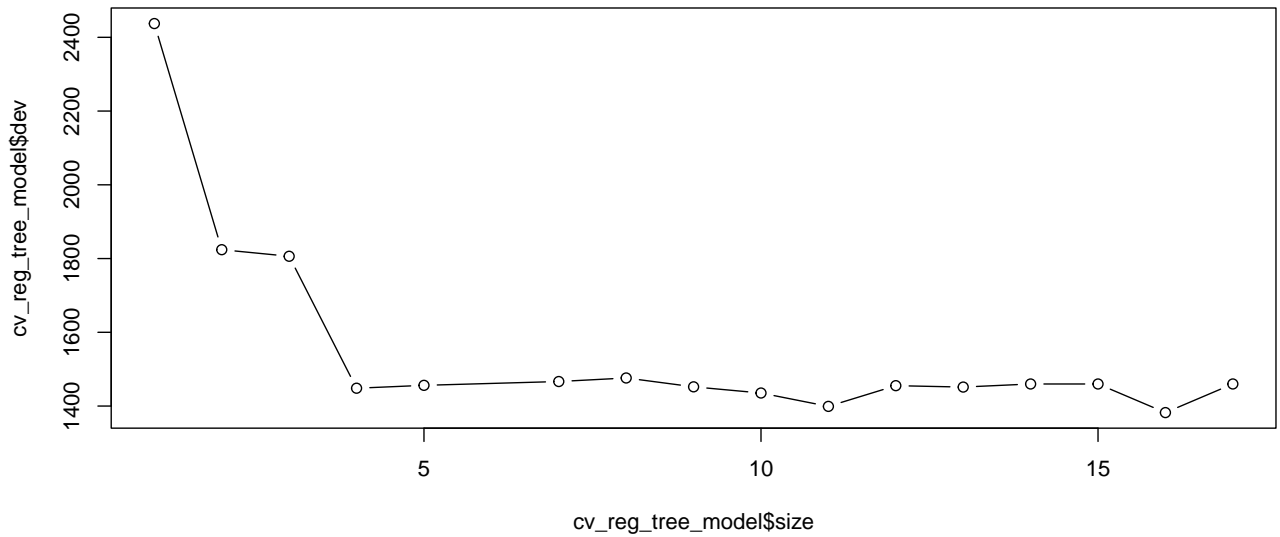
```
# calculate test MSE
yhat= predict(reg_tree_model ,newdata=df_test)
mse = mean((yhat-df_test$Sales)^2)

print(paste("The test MSE is: ", mse))
```

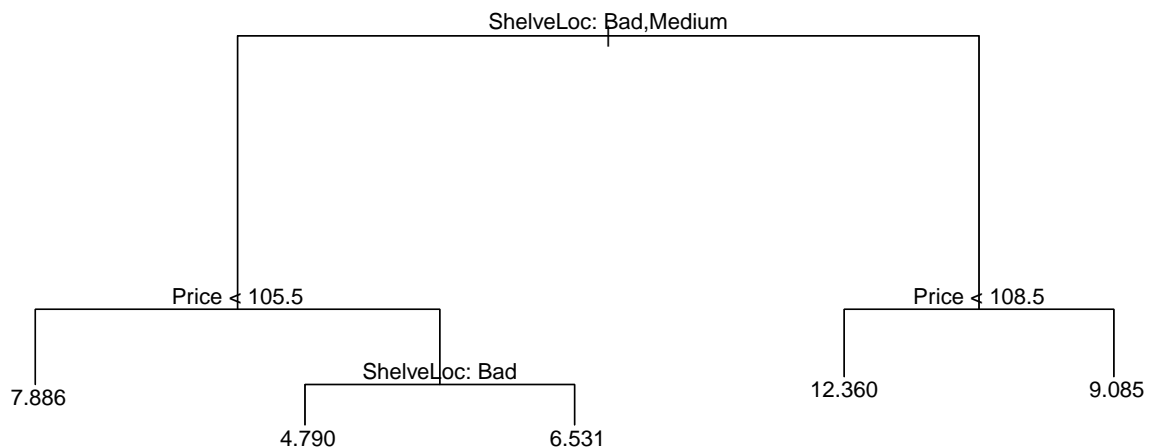
```
## [1] "The test MSE is: 4.70515261090458"
```

c.)

```
# apply cross validation on tree complexity
set.seed(0)
cv_reg_tree_model = cv.tree(reg_tree_model)
plot(cv_reg_tree_model$size ,cv_reg_tree_model$dev ,type="b")
```



```
# prune tree to best size from CV
set.seed(0)
prune_reg_tree_model=prune.tree(reg_tree_model ,best=5)
plot(prune_reg_tree_model )
text(prune_reg_tree_model ,pretty =0)
```



```

# calculate test MSE on pruned tree
yhat= predict(prune_reg_tree_model ,newdata=df_test)
mse = mean((yhat-df_test$Sales)^2)

print(paste("The test MSE after pruning is: ", mse))

```

```
## [1] "The test MSE after pruning is: 4.62268983079291"
```

Pruning the decision tree only slightly improves the MSE.

d.)

```

# import random forest package
library(randomForest)

```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```

# create bagging model
set.seed(0)
bag_tree_model = randomForest(Sales~.,
                              data=df_train,
                              mtry=ncol(df_train)-1,
                              importance =TRUE)

```

```

# calculate test MSE on pruned tree

```

```
yhat = predict(bag_tree_model ,newdata=df_test)
mse = mean((yhat-df_test$Sales)^2)

print(paste("The test MSE for the bagging model is: ", mse))
```

```
## [1] "The test MSE for the bagging model is: 2.5585149484648"
```

```
importance(bag_tree_model)
```

```
##           %IncMSE IncNodePurity
## CompPrice 30.2792313    229.93406
## Income   12.3444510    149.82951
## Advertising 19.5025809    155.85323
## Population -1.1909030     81.69699
## Price     68.1588497    630.89837
## ShelfLoc  74.1374927    824.04818
## Age       19.4934903    203.09335
## Education  3.2020555     57.17521
## Urban     -0.3537144     13.31629
## US        3.7656699     14.11128
```

The bagging approach substantially decreases the MSE. The most important predictors are ShelfLoc and Price.

e.)

```
# create random forest model
set.seed(0)
rf_tree_model = randomForest(Sales~.,
                             data=df_train,
                             importance =TRUE)

# calculate test MSE on pruned tree
yhat = predict(rf_tree_model ,newdata=df_test)
mse = mean((yhat-df_test$Sales)^2)

print(paste("The test MSE for the RF model is: ", mse))
```

```
## [1] "The test MSE for the RF model is: 2.91624482305896"
```

```
importance(rf_tree_model)
```

```
##           %IncMSE IncNodePurity
## CompPrice 16.9065928    213.73663
## Income    6.0447745    182.26679
```

```
## Advertising 13.8972554    190.11656
## Population  -1.8545867    137.77206
## Price       42.1641730    525.36243
## ShelfLoc    50.7665440    590.08970
## Age         15.8340355    265.95503
## Education   0.8776803     99.64608
## Urban       -1.8020997     23.94132
## US          5.7207788     35.92824
```

The MSE is reduced substantially for the random forest model. The most important predictors are ShelfLoc and Price. By reducing  $m$  the size increases the error rate slightly.

## Question 9

a.)

```
# import data
set.seed(0)
df_data = OJ

# split data
train_test_filter = sample.split(df_data$Purchase,
                                SplitRatio = 800/nrow(df_data))
df_train = subset(df_data, train_test_filter==TRUE)
df_test = subset(df_data, train_test_filter==FALSE)
```

b.)

```
# create tree classifier model
set.seed(0)
class_tree_model = tree(Purchase~., data=df_train)
summary(class_tree_model)

##
## Classification tree:
## tree(formula = Purchase ~ ., data = df_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"      "PriceDiff"    "SpecialCH"    "ListPriceDiff"
## [5] "PctDiscMM"
## Number of terminal nodes: 9
## Residual mean deviance: 0.7189 = 568.7 / 791
## Misclassification error rate: 0.1575 = 126 / 800
```

The training error rate of the model is: 0.1575 The number of terminal nodes on the tree is 9

c.)

```
class_tree_model
```

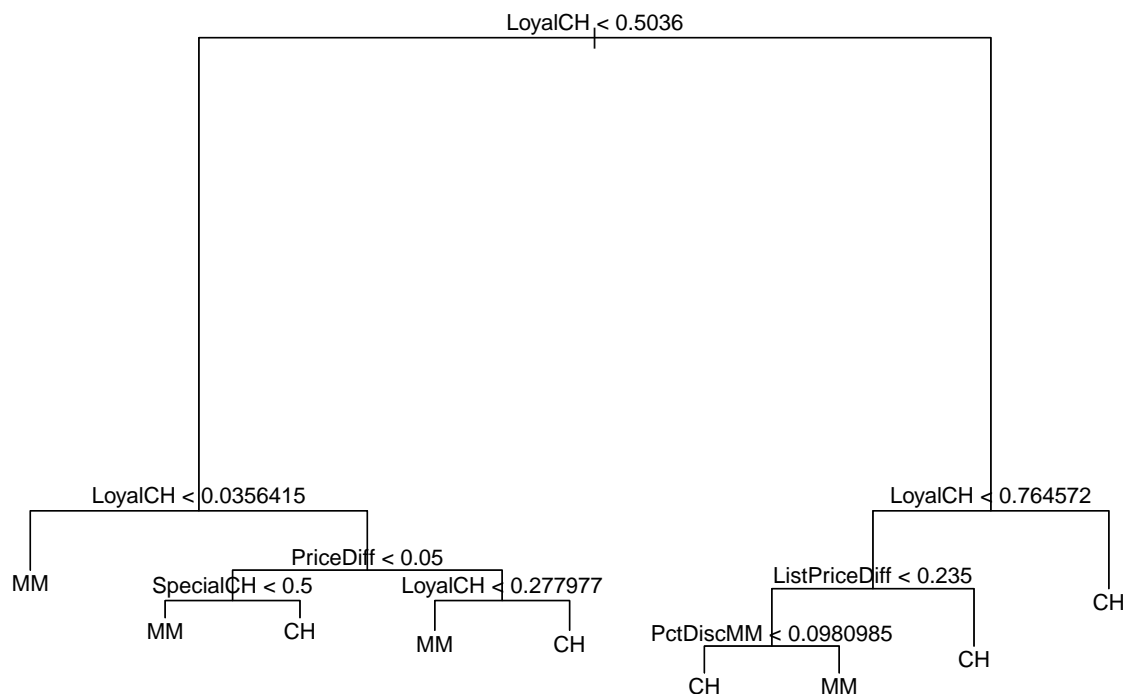
```
## node), split, n, deviance, yval, (yprob)
##      * denotes terminal node
##
## 1) root 800 1070.00 CH ( 0.61000 0.39000 )
##    2) LoyalCH < 0.5036 355 422.10 MM ( 0.28169 0.71831 )
##      4) LoyalCH < 0.0356415 53 0.00 MM ( 0.00000 1.00000 ) *
##      5) LoyalCH > 0.0356415 302 383.50 MM ( 0.33113 0.66887 )
##        10) PriceDiff < 0.05 132 131.00 MM ( 0.19697 0.80303 )
##          20) SpecialCH < 0.5 115 96.35 MM ( 0.14783 0.85217 ) *
##          21) SpecialCH > 0.5 17 23.51 CH ( 0.52941 0.47059 ) *
##        11) PriceDiff > 0.05 170 232.80 MM ( 0.43529 0.56471 )
##          22) LoyalCH < 0.277977 63 66.74 MM ( 0.22222 0.77778 ) *
##          23) LoyalCH > 0.277977 107 146.80 CH ( 0.56075 0.43925 ) *
##    3) LoyalCH > 0.5036 445 340.60 CH ( 0.87191 0.12809 )
##      6) LoyalCH < 0.764572 185 211.80 CH ( 0.74054 0.25946 )
##        12) ListPriceDiff < 0.235 78 108.10 CH ( 0.51282 0.48718 )
##          24) PctDiscMM < 0.0980985 45 54.10 CH ( 0.71111 0.28889 ) *
##          25) PctDiscMM > 0.0980985 33 36.55 MM ( 0.24242 0.75758 ) *
##        13) ListPriceDiff > 0.235 107 66.44 CH ( 0.90654 0.09346 ) *
##    7) LoyalCH > 0.764572 260 78.23 CH ( 0.96538 0.03462 ) *
```

- 4) LoyalCH < 0.0356415 53 0.00 MM ( 0.00000 1.00000 ) \* is a terminal node. This shows us that whenever predictor LoyalCH is smaller than 0.0356415 that the classification of Purchase is equal to MM, otherwise then it means that the next nested node should be considered.

d.)

```
{plot(class_tree_model)
text(class_tree_model,pretty=0)}
```





From this tree we can see how nested the predictors are in describing the classifications.

e.)

```

# predict test data
pred_class_tree_model = predict(class_tree_model,
                                newdata = df_test,
                                type = "class")

# produce confusion matrix
conf_mat = table(pred_class_tree_model, df_test$Purchase)
conf_mat

```

```

##
## pred_class_tree_model  CH  MM
##                CH 152  37
##                MM  13  68

```

```

# calculate error rate
test_error_rate = sum(conf_mat[1,2], conf_mat[2,1])/sum(conf_mat)
print(paste("The test error rate is: ", test_error_rate))

```

```

## [1] "The test error rate is:  0.185185185185185"

```

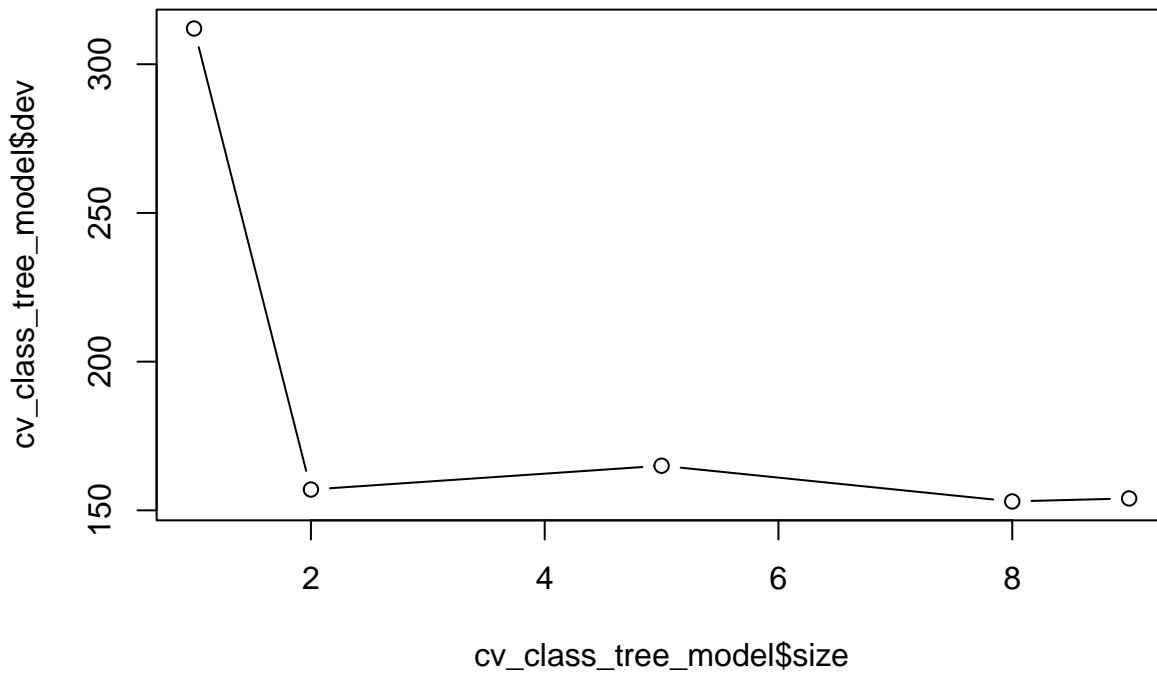
f.)

```
# cross validation on tree model complexity
set.seed(0)
cv_class_tree_model = cv.tree(class_tree_model, FUN=prune.misclass)
cv_class_tree_model
```

```
## $size
## [1] 9 8 5 2 1
##
## $dev
## [1] 154 153 165 157 312
##
## $k
## [1]      -Inf    1.000000    4.333333    5.666667 155.000000
##
## $method
## [1] "misclass"
##
## attr("class")
## [1] "prune"      "tree.sequence"
```

g.)

```
# plot deviance against tree size
plot(cv_class_tree_model$size ,cv_class_tree_model$dev ,type="b")
```



It

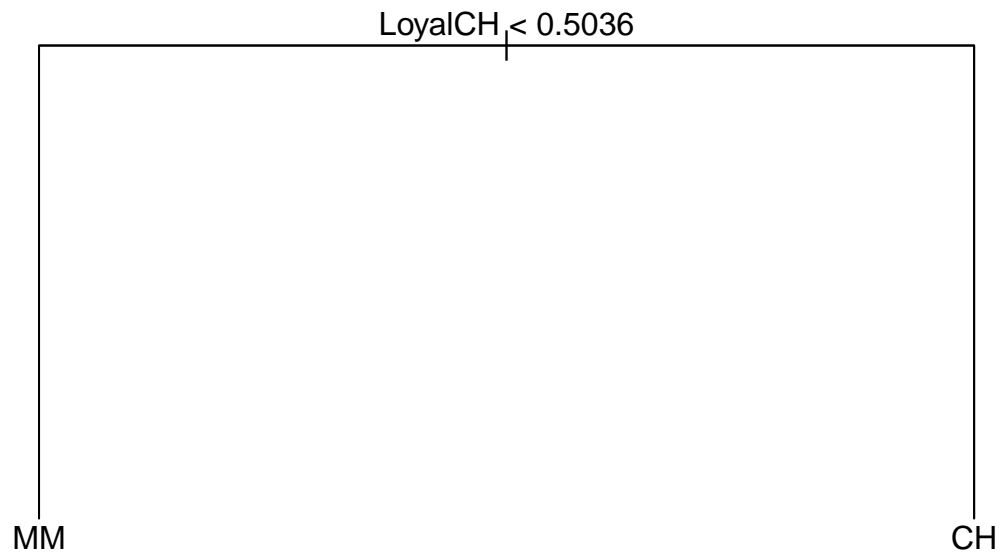
seems that the ideal tree size might be 2.

h.)

Even though a size of 8 (153) gives the lowest cross-validated classification error rate, it is not too much different from that given by a size of 2 (157). Therefore 2 might be the better option when considering Occam's razor.

i.)

```
# prune tree
prune_class_tree_model <- prune.misclass(class_tree_model, best = 2)
plot(prune_class_tree_model)
text(prune_class_tree_model, pretty = 0)
```



j.)

```
# compare pruned tree to original unpruned tree
summary(class_tree_model)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = df_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"      "PriceDiff"    "SpecialCH"    "ListPriceDiff"
## [5] "PctDiscMM"
## Number of terminal nodes: 9
## Residual mean deviance: 0.7189 = 568.7 / 791
## Misclassification error rate: 0.1575 = 126 / 800
```

```
summary(prune_class_tree_model)
```

```
##
## Classification tree:
## snip.tree(tree = class_tree_model, nodes = 2:3)
## Variables actually used in tree construction:
## [1] "LoyalCH"
```

```
## Number of terminal nodes: 2
## Residual mean deviance: 0.9558 = 762.8 / 798
## Misclassification error rate: 0.1962 = 157 / 800
```

The unpruned tree has a slightly lower training error rate. However, the pruned tree is much less complex.

k.)

```
# predict test data
pred_class_tree_model = predict(prune_class_tree_model,
                                newdata = df_test,
                                type = "class")

# produce confusion matrix
conf_mat = table(pred_class_tree_model, df_test$Purchase)
conf_mat
```

```
##
## pred_class_tree_model CH MM
##                CH 132 24
##                MM 33 81
```

```
# calculate error rate
test_error_rate = sum(conf_mat[1,2], conf_mat[2,1])/sum(conf_mat)
print(paste("The test error rate is: ", test_error_rate))
```

```
## [1] "The test error rate is: 0.211111111111111"
```

The testing error rate is also slightly higher for the pruned tree.

## Question 10

a.)

```
# remove NA for salaries
df_data = Hitters
df_data = df_data[-which(is.na(df_data$Salary)), ]
sum(is.na(df_data$Salary))
```

```
## [1] 0
```

```
# log transform salary data
df_data$Salary = log(df_data$Salary)
```

b.)

```
# split data
df_train = df_data[1:200, ]
df_test = df_data[-(1:200), ]
```

c.)

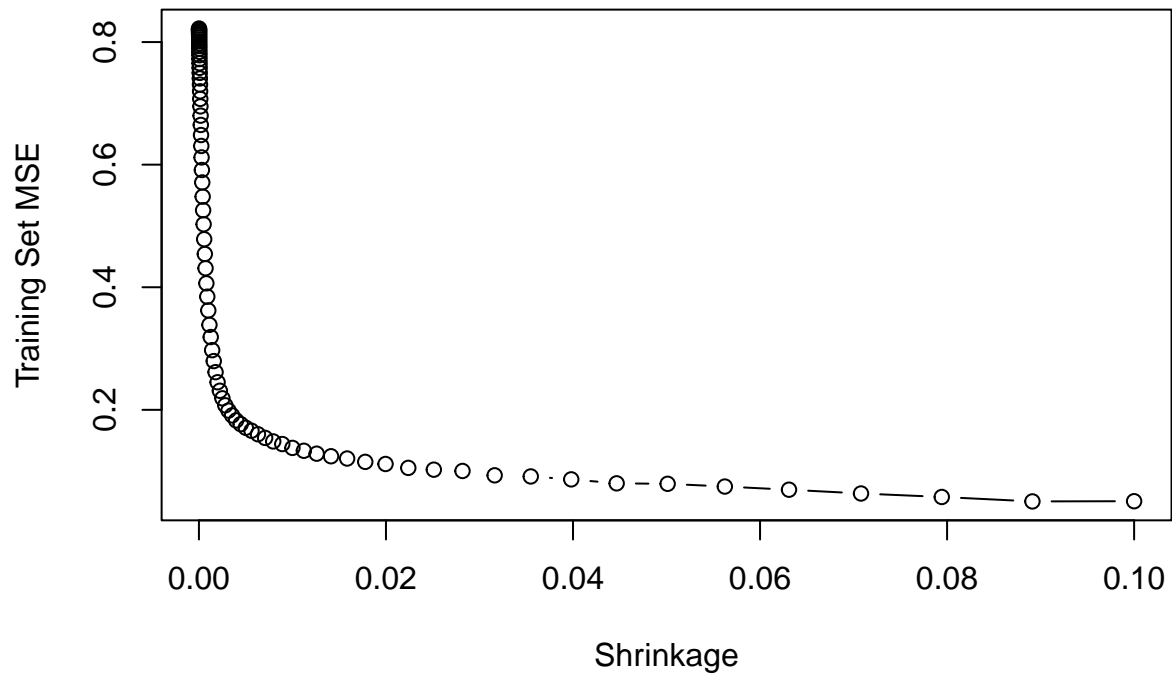
```
# import libraries
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
# boosting
set.seed(0)
lambdas = 10^seq(-5, -1, by = 0.05)
train_mse = rep(NA, length(lambdas))
test_mse = rep(NA, length(lambdas))
for (i in 1:length(lambdas)) {
  boost= gbm(Salary ~ ., data = df_train, distribution = "gaussian",
             n.trees = 1000, shrinkage = lambdas[i])
  train_pred = predict(boost, df_train, n.trees = 1000)
  test_pred = predict(boost, df_test, n.trees = 1000)
  train_mse[i] = mean((df_train$Salary - train_pred)^2)
  test_mse[i] = mean((df_test$Salary - test_pred)^2)
}

# plot results
plot(lambdas,
     train_mse,
     type = "b",
     main = 'Training Set MSE vs Shrinkage',
     xlab = "Shrinkage",
     ylab = "Training Set MSE")
```

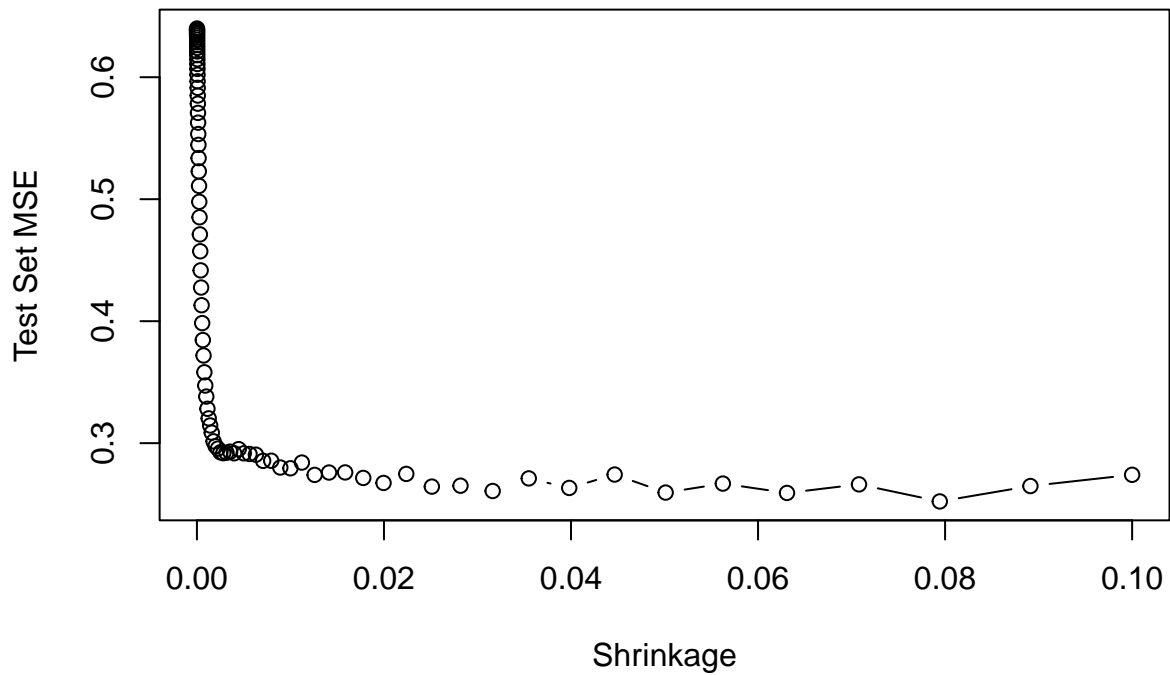
**Training Set MSE vs Shrinkage**



d.)

```
# plot MSe vs Shrinkage
plot(lambdas,
     test_mse,
     type = "b",
     main = 'Test Set MSE vs Shrinkage',
     xlab = "Shrinkage",
     ylab = "Test Set MSE")
```

## Test Set MSE vs Shrinkage



```
min(test_mse)
```

```
## [1] 0.252252
```

```
lambdas[which.min(test_mse)]
```

```
## [1] 0.07943282
```

e.)

```
# import library  
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

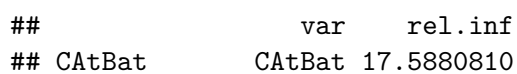
```
# Regression model  
set.seed(0)  
lm = lm(Salary ~ ., data = df_train)  
lm_pred = predict(lm, df_test)  
mean((df_test$Salary - lm_pred)^2)
```



```
x_train = model.matrix(Salary ~ ., data = df_train)
x_test = model.matrix(Salary ~ ., data = df_test)
lasso = glmnet(x_train, df_train$Salary, alpha = 1)
lasso_pred = predict(lasso, s = 0.01, newx = x_test)
mean((df_test$Salary - lasso_pred)^2)
```

The test MSE of boosting model is lower than linear and Lasso models.

```
set.seed(0)
best = gbm(Salary ~ ., data = df_train, distribution = "gaussian",
           n.trees = 1000, shrinkage = lambdas[which.min(test_mse)])
summary(best)
```



```
## CRuns          CRuns 12.1657282
## PutOuts        PutOuts 9.0297031
## CRBI           CRBI 7.6523867
## CHmRun         CHmRun 7.0986463
## Walks          Walks 6.8327535
## CWalks         CWalks 6.0398111
## Years          Years 5.6228571
## RBI            RBI 4.9330850
## Hits           Hits 4.1969825
## Assists        Assists 3.9439999
## AtBat          AtBat 3.8472490
## HmRun          HmRun 3.2829758
## Runs           Runs 2.6624755
## Errors         Errors 2.3828680
## CHits          CHits 1.3986994
## Division       Division 0.6689345
## NewLeague      NewLeague 0.4376038
## League         League 0.2151596
```

CAtBat, CRuns and PutOuts are most important predictors in the boosted model

g.)

```
set.seed(0)
rf = randomForest(Salary ~ ., data = df_train, ntree = 500, mtry = 19)
rf_pred = predict(rf, df_test)
mean((df_test$Salary - rf_pred)^2)
```

```
## [1] 0.2304067
```

The test MSE of bagging is lower than boosting.