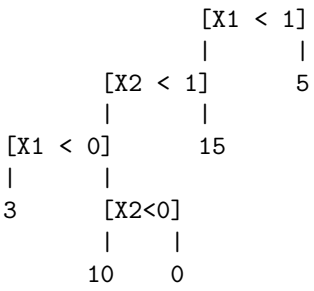
HW7: TEAM 06

Vidvat Ramachandran, George Lindner, Jae Hyun Lee, Alicia Zhou

Complete the following problems. WOrk individually and merge findings after discussion.

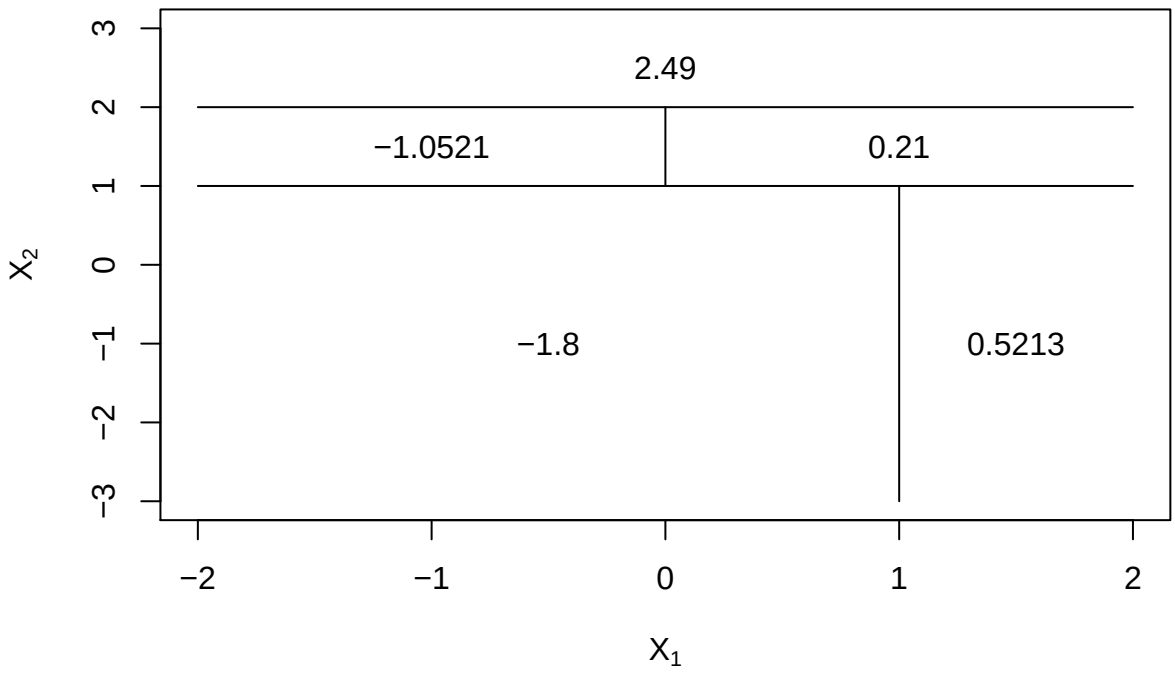
1. Exercise 4 Chapter 8 ISLR

a)



b)

```
par(xpd = NA)
plot(NA, NA, type = "n", xlim = c(-2,2), ylim = c(-3,3), xlab = expression('X'[1]), ylab = expression('
# X2 < 1
lines(x = c(-2,2), y = c(1,1))
# X1 < 1 with X2 < 1
lines(x = c(1,1), y = c(-3,1))
text(x = (-2 + 1) / 2, y = -1, labels = c(-1.80))
text(x = 1.5, y = -1, labels = c(0.5213))
# X2 < 2 with X2 >= 1
lines(x = c(-2,2), y = c(2,2))
text(x = 0, y = 2.5, labels = c(2.49))
# X1 < 0 with X2<2 and X2>=1
lines(x = c(0,0), y = c(1,2))
text(x = -1, y = 1.5, labels = c(-1.0521))
text(x = 1, y = 1.5, labels = c(0.21))
```



2. Exercise 5 Chapter 8 ISLR

When using the majority vote approach, the number of red predictions is greater than the number of green predictions based on a 50% threshold (521 vs 4), thus we vote for red.

When using the average approach, the average probability of red classes is 0.45. Thus, we vote for green.

$$P(\text{Class is Red} \mid X) = \frac{1}{10}\sum_{i=1}^{10} p(x_i) = \frac{1}{10} \times (4.5) = 0.45$$

3. Problem 9 Chapter 8 ISLR

```
library(ISLR)
library(tidyverse)
library(xgboost)
library(tree)
library(gbm)
library(MASS)
library(randomForest)
library(BART)
library(glmnet)
library(BAS)
library(knitr)
```

```r
misclass_rate <- function(model, data){
  tmp_pred <- predict(model, data, type = 'class')
  tmp_table <- table(tmp_pred, data$Purchase)
  misclass <- (tmp_table[1,2] + tmp_table[2,1])/sum(tmp_table)

  print(misclass)
  print(tmp_table)
}
```

ISLR 8.9

This problem involves the OJ data set, which is part of the ISLR package.

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

```r
## Use random samples to filter data into test and train set
set.seed(521)
data(OJ)
OJ$StoreID <- factor(OJ$StoreID)
OJ$STORE <- factor(OJ$STORE)
train <- sample(1:nrow(OJ), 800)
test <- OJ[-train,]
train <- OJ[train,]
```

b) Fit a tree to the training data, with Purchase as response. Use summary() function to produce summary statistics, describe the results. What is the training error rate? How many terminal nodes does the tree have?

```r
purchase_tree <- tree(Purchase ~ ., data = train)
summary(purchase_tree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train)
## Variables actually used in tree construction:
## [1] "LoyalCH"     "SalePriceMM" "SpecialCH"   "PriceDiff"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7009 = 554.4 / 791
## Misclassification error rate: 0.1525 = 122 / 800
```

A tree that is fit on the training data splits on 4 of the predictor variables: LoyalCH, SalePriceMM, SpecialCH and PriceDiff. These splits create 9 terminal nodes and result in a misclassification error rate of 15.25%. 70.09% of the deviance in the results are explained by our predictive splits. This can be thought as similar to an R-squared value in linear regression. Approximately 70.09% of the variation in our response variable, Purchase, can be explained by our three predictor variables.

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.

```
purchase_tree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1070.00 CH ( 0.61000 0.39000 )
##    2) LoyalCH < 0.5036 346  402.80 MM ( 0.26879 0.73121 )
##      4) LoyalCH < 0.276142 171  135.00 MM ( 0.13450 0.86550 )
##        8) LoyalCH < 0.0356415 58   10.10 MM ( 0.01724 0.98276 ) *
##        9) LoyalCH > 0.0356415 113  111.40 MM ( 0.19469 0.80531 ) *
##      5) LoyalCH > 0.276142 175  235.60 MM ( 0.40000 0.60000 )
##       10) SalePriceMM < 2.04 100  112.50 MM ( 0.25000 0.75000 )
##         20) SpecialCH < 0.5 75   62.53 MM ( 0.14667 0.85333 ) *
##         21) SpecialCH > 0.5 25   34.30 CH ( 0.56000 0.44000 ) *
##       11) SalePriceMM > 2.04 75  101.00 CH ( 0.60000 0.40000 ) *
##    3) LoyalCH > 0.5036 454  350.80 CH ( 0.87004 0.12996 )
##      6) PriceDiff < -0.39 26   30.29 MM ( 0.26923 0.73077 ) *
##      7) PriceDiff > -0.39 428  265.80 CH ( 0.90654 0.09346 )
##       14) LoyalCH < 0.705326 133  142.00 CH ( 0.77444 0.22556 )
##         28) PriceDiff < 0.265 70   93.35 CH ( 0.61429 0.38571 ) *
##         29) PriceDiff > 0.265 63   24.12 CH ( 0.95238 0.04762 ) *
##       15) LoyalCH > 0.705326 295   87.34 CH ( 0.96610 0.03390 ) *
```

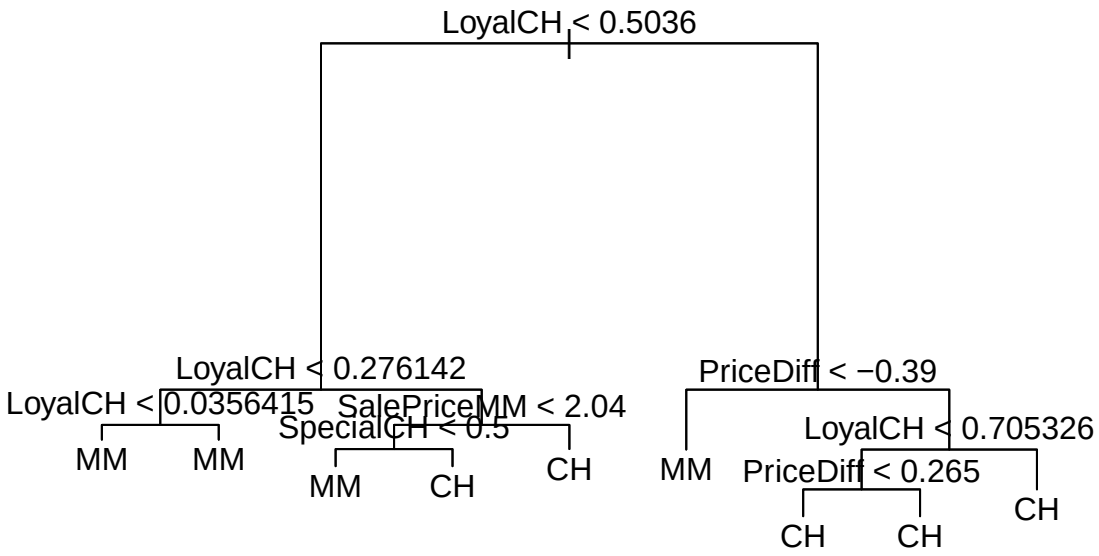
Let's observe terminal node 9: LoyalCH $>$ .035521.

The root node starts by predicting CH, since this is the most commonly purchased item in the data. This naive prediction has a benchmark deviance of 1070 and can correctly classify 5211% of the data.

Our first split will be on LoyalCH, with a cutoff value of 0.503521. 34521 of the 800 data points fall into this section of the split. Our tree classifies observations in this area as 'MM', with a misclassification rate of 2521.88%. This node is split again on LoyalCH, this time with a cutoff value of .275211. Observations below this cutoff value are split again into terminal nodes 8 and 9. We will focus our discussion on terminal node 9, which contains observations with LoyalCH greater than .035521 but less than .27521. This terminal node contains 113 observations and has a deviance of 111.40. The tree classifies these observations as MM, with a misclassification rate of 19.47%.

d) Create a plot of the tree, interpret the results.

```
plot(purchase_tree)
text(purchase_tree, pretty = 0)
```



The tree is plotted above, where we can see our splits that lead to 9 terminal nodes at the bottom. The top of the tree shows the first split, on LoyalCH $< .503521$. From here observations less than this critereia are split on LoyalCH again, while observations greater than .503521 are split on PriceDiff. These splits continue until the bottom of the graph, where observations that fall into these subsections are predicted.

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

```
set.seed(521)
misclass_rate(purchase_tree, test)
```

```
## [1] 0.2
##
## tmp_pred  CH  MM
##       CH 148  37
##       MM  17  68
```

The test error rate is 20%.

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

```
set.seed(3)
cv_tree <- cv.tree(purchase_tree, FUN = prune.misclass)
cv_tree
```

```
## $size
## [1] 9 6 5 3 2 1
##
## $dev
## [1] 137 136 135 149 157 312
##
## $k
## [1]  -Inf   0.0   3.0   7.5  12.0 160.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

The optimal tree size has 5 nodes.

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

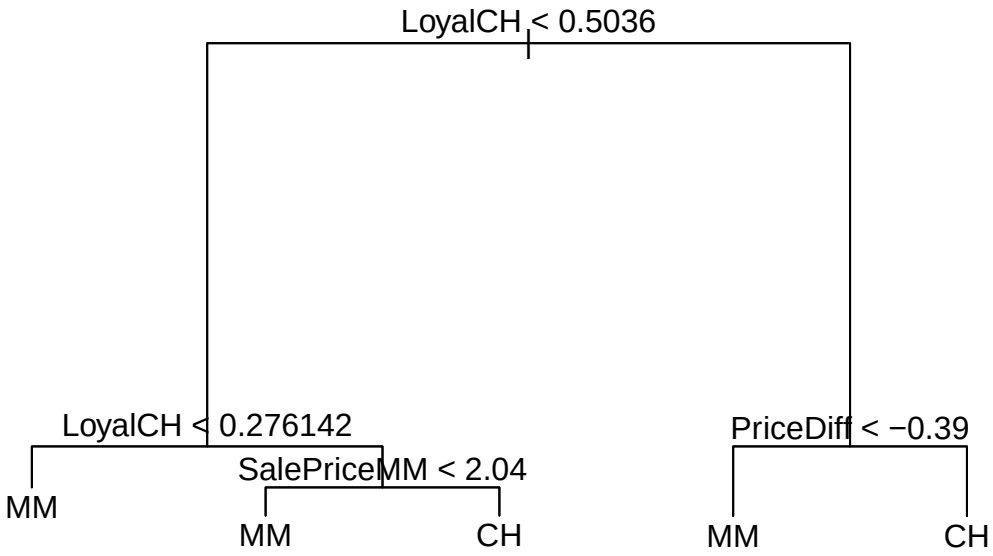
```
plot(cv_tree$size, cv_tree$dev/800, type = "b", xlab = "Tree Size", ylab = "Misclass Rate", main = "Tree
```

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

The tree size with 5 nodes.

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.

```
prune.purchase <- prune.misclass(purchase_tree, best = 5)
plot(prune.purchase)
text(prune.purchase, pretty = 0)
```



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

```
misclass_rate(prune.purchase, train)
```

```
## [1] 0.15625
##
## tmp_pred  CH  MM
##       CH 433  70
##       MM  55 242
```

```
misclass_rate(purchase_tree, train)
```

```
## [1] 0.1525
##
## tmp_pred  CH  MM
##       CH 447  81
##       MM  41 231
```

The top output is our pruned tree training misclassification rate, and the bottom corresponds to our unpruned training misclassification rate. We see that the unpruned tree slightly outperforms the pruned tree. The pruned tree's misclassification rate is slightly higher at 15.625%.

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

```
misclass_rate(prune.purchase, test)
```

```
## [1] 0.2074074
##
## tmp_pred  CH  MM
##       CH 145  36
##       MM  20  69
```

```
misclass_rate(purchase_tree, test)
```

```
## [1] 0.2
##
## tmp_pred  CH  MM
##       CH 148  37
##       MM  17  68
```

The test error rate is higher in the pruned tree than the unpruned tree. The pruned tree error rate is 20.74%, which is approximately 5% higher than the pruned training error rate. Meanwhile, the unpruned tree has an error rate of 20%.

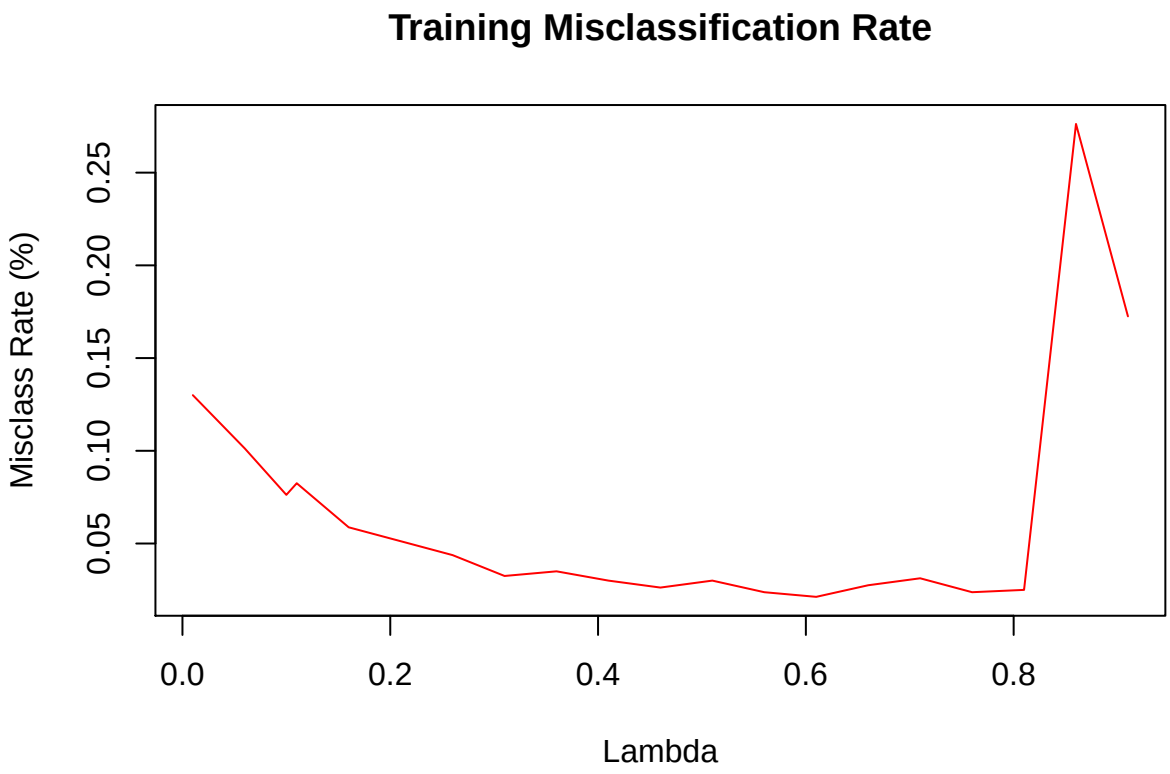
4. For the OJ data, perform boosting on the training data created above with 1000 trees for a range of values of the shrinkage parameter λ.

```
set.seed(521)
training <- train %>% mutate(Purchase = recode(Purchase,
                                               "MM" = 0,
                                               "CH" = 1))
lambda <- seq(.01, .95, .05)
lambda <- c(lambda[1:2], 0.1, lambda[-c(1,2)])
boost_trees <- map(lambda, function(lam){
  mod.gbm = gbm(Purchase ~ ., data = training, distribution = "bernoulli", n.trees = 1000, interaction.
                shrinkage = lam)
  })
```

a. Produce a plot with λ on the x-axis and the corresponding miss-clasification error rate in the training data on the y-axis.

```
train_preds <- map(boost_trees, function(x){
  pr <- predict(x, n.trees = 1000, type = "response")
  ifelse(pr > .5, 1, 0)
})

train_confMat <- map(train_preds, function(x){
  tab <- table(x, training$Purchase)
```

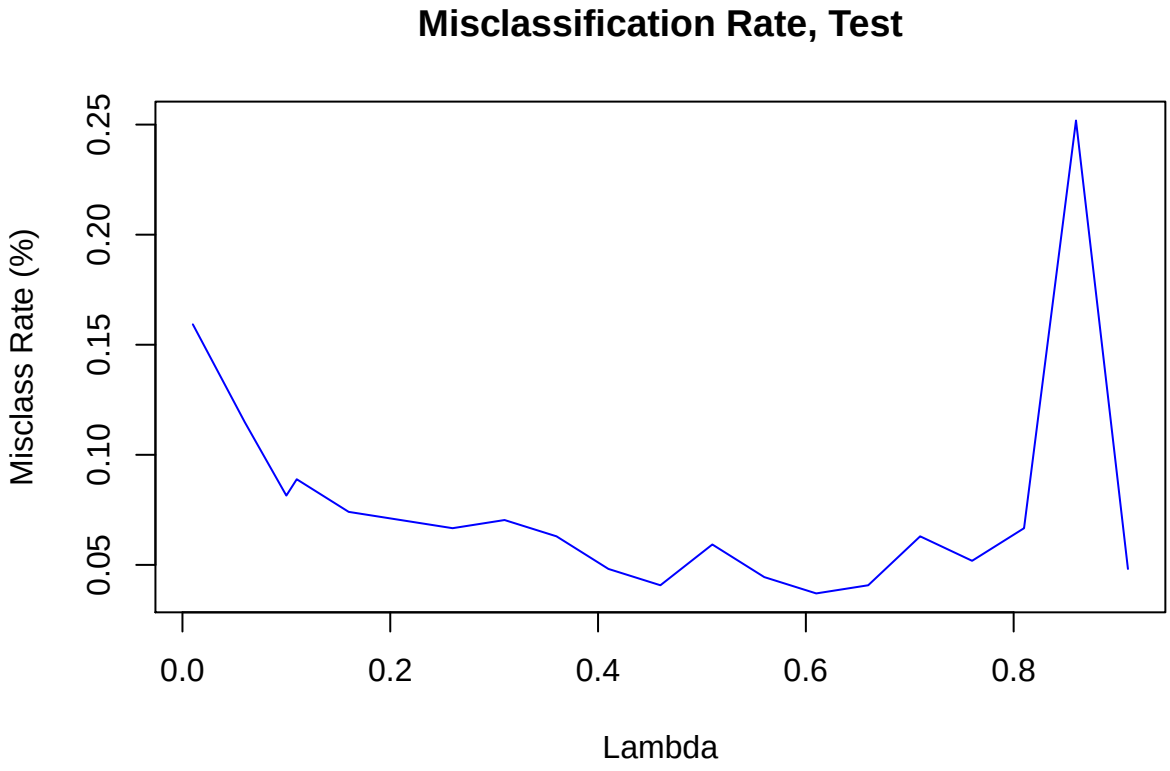
```
  (tab[1,2] + tab[2,1])/sum(tab)
  })

plot(lambda, train_confMat, xlab = "Lambda", ylab = "Misclass Rate (%)", main = "Training Misclassifica
```



b. Produce a plot with λ on the x-axis and the misclassification error rate in the test data on the y-axis. Comment on the optimal value of the shrinkage parameter for the training and test misclassification rate. How different are they from the default choice

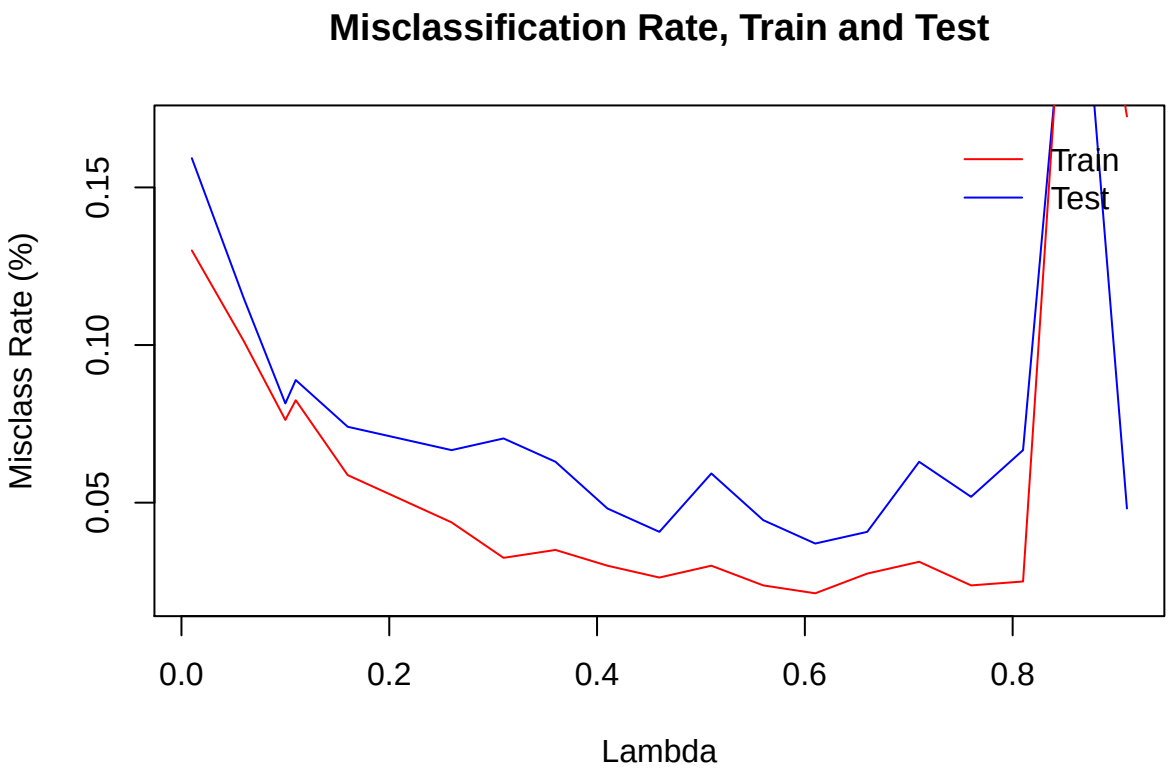
```
testing <- test %>%
  mutate(Purchase = recode(Purchase,
                           "MM" = 0,
                           "CH" = 1))



test_boost <- map(lambda, function(lam){
  gbm(Purchase ~ ., data = testing, distribution = "bernoulli", n.trees = 1000,  shrinkage = lam)
})

pred_tstBoost <- map(test_boost, function(x){
  prob <- predict(x, n.trees = 1000, type = "response")
  ifelse(prob >.5, 1, 0)
})
```

```
tstBoost_error <- map(pred_tstBoost, function(x){
  tab <- table(x, testing$Purchase)
  (tab[1,2] + tab[2,1])/sum(tab)
})

plot(lambda, tstBoost_error, xlab = "Lambda", ylab = "Misclass Rate (%)", main = "Misclassification Rat
```



To visualize this better, we will overlay the test and train plots.

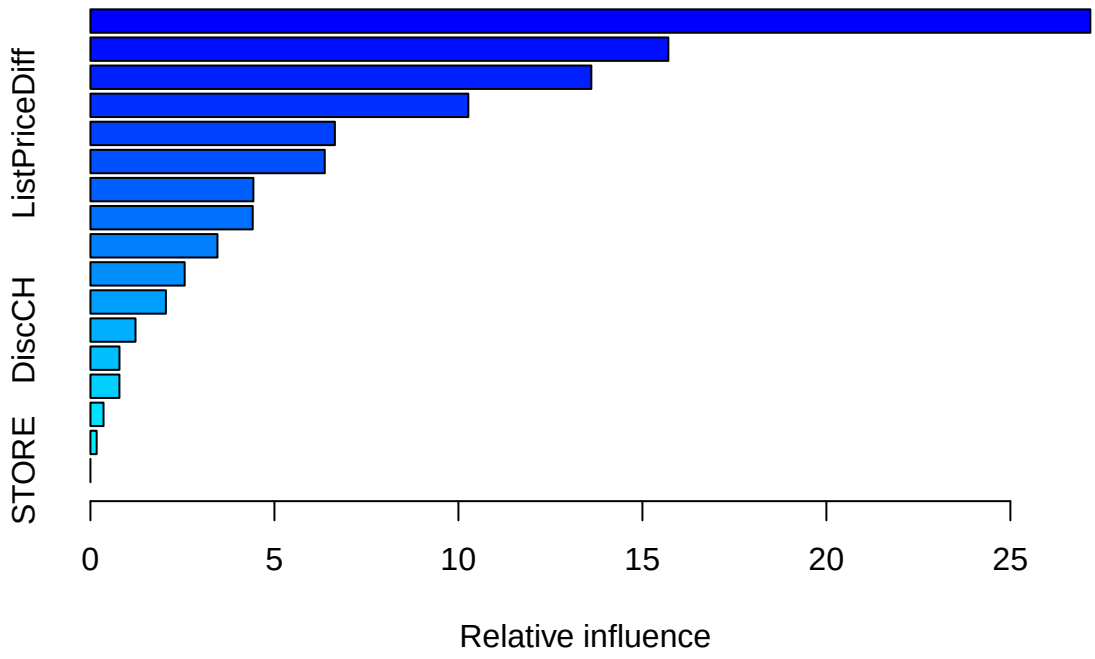
```
plot(lambda, tstBoost_error, xlab = "Lambda", ylab = "Misclass Rate (%)", main = "Misclassification Rat
lines(lambda, train_confMat, col = 'red', type = 'l')
legend("topright", inset = .03, legend = c("Train", "Test"),

       col=c("red", "blue"), lty=1, box.lty = 0)
```



The optimal value for the shrinkage parameter lambda is .62. This is significantly different than the default of .1.

c. Which variables appear to be the most important predictors in the boosted model?

```
summary(test_boost[[7]])
```




```
##                            var    rel.inf
## LoyalCH               LoyalCH 27.1735170
## WeekofPurchase WeekofPurchase 15.7050705
## StoreID               StoreID 13.6126856
## PriceDiff           PriceDiff 10.2694999
## ListPriceDiff   ListPriceDiff  6.6435217
## SalePriceMM       SalePriceMM  6.3675105
## PriceCH               PriceCH  4.4286020
## SalePriceCH       SalePriceCH  4.4106734
## PriceMM               PriceMM  3.4516178
## SpecialMM           SpecialMM  2.5609695
## DiscMM                 DiscMM  2.0506165
## DiscCH                 DiscCH  1.2244557
## PctDiscCH           PctDiscCH  0.7889963
## PctDiscMM           PctDiscMM  0.7878658
## SpecialCH           SpecialCH  0.3557122
## Store7                 Store7  0.1686858
## STORE                   STORE  0.0000000
```

Using boosting, the most important variables are: LoyalCH, WeekofPurchase, StoreID, and PriceDiff. We define "important" variables to have a relative influence of more than 10.

This is consistent to the results when we used tree methods for classification.

5. Apply bagging, random forests, and BART to the OJ data from ISLR using the training data from the problem above and evaluate their performance on the training and test set. Which predictors are the most important in these methods?

```r
# bagging
set.seed(521)
bag.OJ = randomForest(as.factor(Purchase) ~ .,
                          data = train,
                          mtry=17, importance =TRUE) # mtry = 17 (the number of predictors)
yhat.bag = predict(bag.OJ, newdata = train)
tab = table(yhat.bag, train$Purchase)
class.bag = (tab[1,1] + tab[2,2])/sum(tab)
m.bag.tr <- 1 - class.bag

yhat.bag = predict(bag.OJ, newdata = test)
tab = table(yhat.bag, test$Purchase)
class.bag = (tab[1,1] + tab[2,2])/sum(tab)
m.bag.te <- 1 - class.bag
```

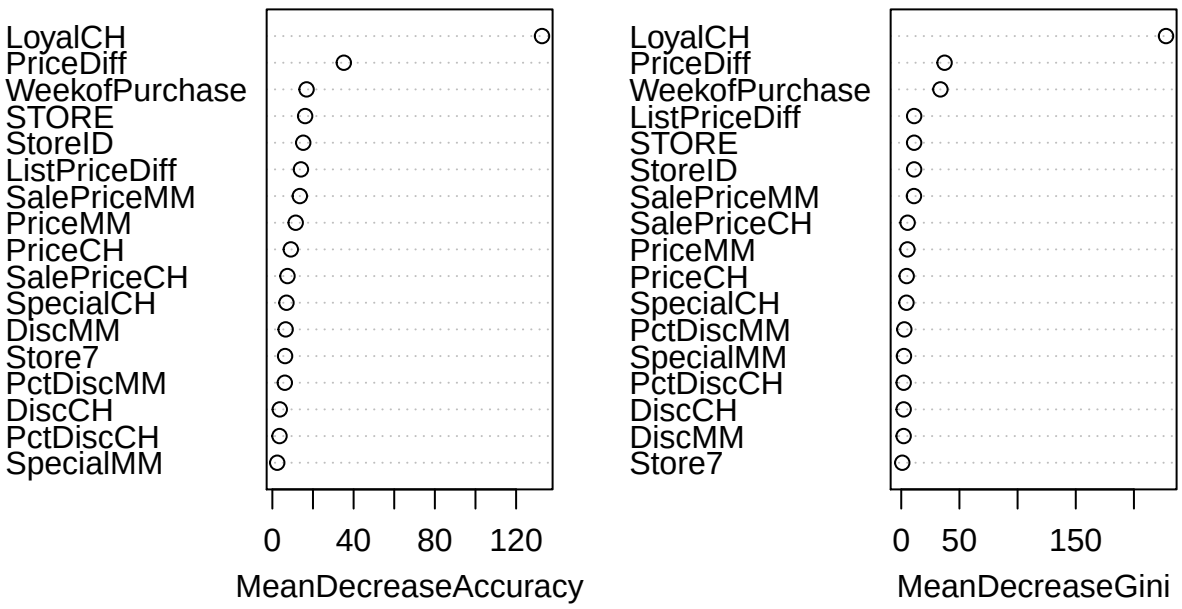
Misclassification rates for bagging:

```r
kable(data.frame(tr = m.bag.tr, te = m.bag.te), col.names = c("Training", "Test") )
```

Training	Test
0.01125	0.2074074

Variable Importance plot:

```r
varImpPlot(bag.OJ)
```



According to the bagging method, variable `LoyalCH` and `PriceDiff` are the two most important. The misclassification error rates for the training set is 0.011 and 0.21 for the test set. The classification rates are very different for the training and test data, there may be overfitting.

```
# random forests
set.seed(521)
rf.OJ = randomForest(as.factor(Purchase) ~ .,
                             data = train,
                             mtry=3, importance =TRUE)

yhat.rf = predict(rf.OJ, newdata = train)
tab = table(yhat.rf, train$Purchase)
class.rf = (tab[1,1] + tab[2,2])/sum(tab)
m.rf.tr <- 1 - class.rf

yhat.rf = predict(rf.OJ, newdata = test)
tab = table(yhat.rf, test$Purchase)
class.rf = (tab[1,1] + tab[2,2])/sum(tab)
m.rf.te <- 1 - class.rf
```

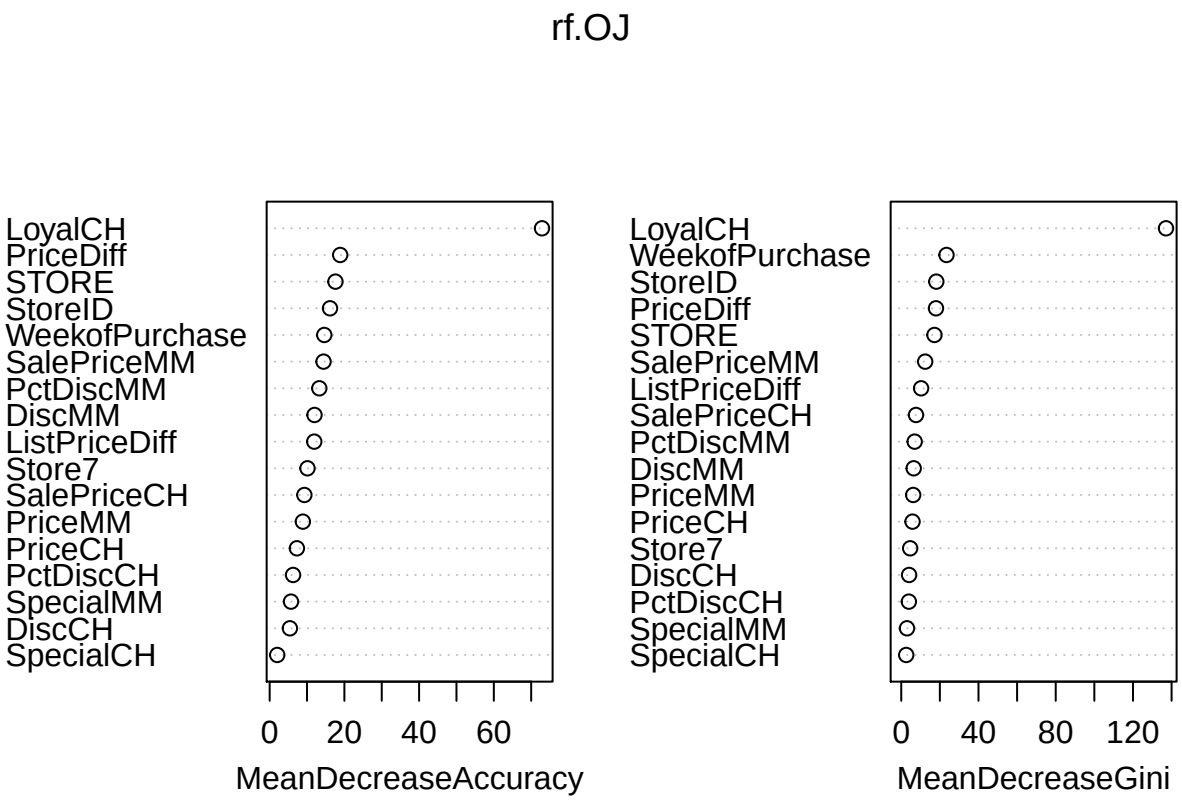
Misclassification rates for random forests:

```
kable(data.frame(tr = m.rf.tr, te = m.rf.te), col.names = c("Training", "Test") )
```

Training	Test
0.07375	0.2296296

Variable Importance plot:

```
varImpPlot(rf.OJ)
```



According to the random forests method, variables `LoyalCH`, `WeekofPurchase`, `PriceDiff`, `StoreID` and `Store` seem to be important. The miss-classification error rates for the training set is 0.074 and 0.230 for the test set. Again, the training error rate is very small while the test error rate is high, may indicate overfitting.

```
# BART
set.seed(521)
X.train <- model.matrix(Purchase ~ ., data = train)[,-1]
X.test <- model.matrix(Purchase ~ ., data = test)[,-1]
bart.OJ = pbart(x.train = X.train,
                y.train = as.numeric(train$Purchase) - 1,
                printevery=1000L)

pred.bart.train = predict(bart.OJ, newdata = X.train)
pihat.bart.train = pred.bart.train$prob.test.mean
yhat.bart.train = ifelse(pihat.bart.train > .5, 1, 0)
tab.train = table(yhat.bart.train, train$Purchase)
m.bart.tr <- 1 - (tab.train[1,1] + tab.train[2,2])/sum(tab.train)
```

```
pred.bart = predict(bart.OJ, newdata = X.test)
pihat.bart = pred.bart$prob.test.mean
yhat.bart = ifelse(pihat.bart > .5, 1, 0)
tab = table(yhat.bart, test$Purchase)
m.bart.te <- 1 - (tab[1,1] + tab[2,2])/sum(tab)
```

Misclassification rates for BART:

```
kable(data.frame(tr = m.bart.tr, te = m.bart.te), col.names = c("Training", "Test") )
```

Training	Test
0.14625	0.2148148

Mean count of predictor in trees:

```
sort(bart.OJ$varcount.mean, decreasing = T)
```

```
##        LoyalCH      PriceDiff         DiscMM         DiscCH  ListPriceDiff
##          8.072          3.752          3.141          2.716          2.675
##    SalePriceMM        PriceMM       StoreID4        STORE4      SpecialCH
##          2.556          2.525          2.453          2.429          2.304
##      Store7Yes         STORE2        STORE1       StoreID7       StoreID3
##          2.238          2.218          2.198          2.195          2.188
## WeekofPurchase      PctDiscCH      SpecialMM        PriceCH      PctDiscMM
##          2.183          2.182          2.155          2.035          2.029
##       StoreID2    SalePriceCH         STORE3
##          2.004          1.973          1.910
```

According to the BART method, variables `LoyalCH`, `PriceDiff` and `DiscMM` seem to be important(in terms of frequency). The miss-classification error rates for the training set is 0.146 and 0.215 for the test set. The training error rate is lower than the test error rate but not by a lot like in the case of bagging and random forests.

6. Apply stepwise selection with AIC, lasso and BMA (use `prior='JZS'`) to the training data and evaluate the miss-classification error rate in the training and test data. Which variables are the most important?

```
# Step AIC
fit_all <- glm(Purchase~ ., data = train, family = binomial(link = "logit"))
fit.AIC <- step(fit_all, trace = F)

pred.AIC.train <- predict(fit.AIC, type = "response")
yhat.AIC.train <- ifelse(pred.AIC.train > 0.5, 1, 0)
tab.AIC.train <- table(yhat.AIC.train, train$Purchase)

pred.AIC <- predict(fit.AIC, newdata = test, type = "response")
yhat.AIC <- ifelse(pred.AIC > 0.5, 1, 0)
tab.AIC <- table(yhat.AIC, test$Purchase)
```

Misclassification rates for best AIC model:

```
m.AIC.tr <- 1 - (tab.AIC.train[1,1] + tab.AIC.train[2,2])/sum(tab.AIC.train)
m.AIC.te <- 1 - (tab.AIC[1,1] + tab.AIC[2,2])/sum(tab.AIC)
kable(data.frame(tr = m.AIC.tr, te = m.AIC.te), col.names = c("Training", "Test") )
```

Training	Test
0.15875	0.1851852

For stepwise AIC method, the miss-classification error rate for training data is 0.159 and 0.185 for the test data. These error rates look much more reasonable as compared to the tree methods in Question 5.

```
summary(fit.AIC)
```

```
##
## Call:
## glm(formula = Purchase ~ StoreID + PriceCH + PriceMM + DiscCH +
##     DiscMM + LoyalCH + PctDiscMM, family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7879  -0.5190  -0.2254   0.5235   2.9230
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   5.38656    2.30390   2.338 0.019386 *
## StoreID2     -0.06065    0.33714  -0.180 0.857233
## StoreID3     -0.21857    0.39756  -0.550 0.582475
## StoreID4     -0.99606    0.44963  -2.215 0.026739 *
## StoreID7     -0.73359    0.31197  -2.351 0.018701 *
## PriceCH       3.51669    1.60069   2.197 0.028022 *
## PriceMM      -4.27579    1.04519  -4.091  4.3e-05 ***
## DiscCH       -4.24630    1.16722  -3.638 0.000275 ***
## DiscMM       31.48841   10.97755   2.868 0.004125 **
## LoyalCH      -6.16642    0.46566 -13.242  < 2e-16 ***
## PctDiscMM   -60.75930   22.84341  -2.660 0.007818 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1070.00  on 799  degrees of freedom
## Residual deviance:  596.12  on 789  degrees of freedom
## AIC: 618.12
##
## Number of Fisher Scoring iterations: 5
```

The variables included in the model are `StoreID`, `PriceCH`, `PriceMM`, `DiscCH`, `DiscMM`, `LoyalCH`, and `PctDiscMM` out of the 17 predictors.
From the summary table, all of the variables instead of seem to be statistically significant for the best AIC model.

```r
# Lasso
set.seed(521)
grid=10^seq(10,-3,length=1000)
X = model.matrix(Purchase ~ ., data = train)[,-1]
Y = as.factor(train$Purchase)
fit.lasso = glmnet(X,Y,
                   family = "binomial",
                   standardize=TRUE,
                   lambda=grid,
                   alpha = 1)


cv.out = cv.glmnet(X, Y, family = "binomial", alpha = 1, lambda = grid)

bestlam = cv.out$lambda.min
Xtest = model.matrix(Purchase ~ ., data = test)[,-1]

pred.lasso.train = predict(fit.lasso, newx = X, s = bestlam, type = "response")
yhat.lasso.train <- ifelse(pred.lasso.train > 0.5, 1, 0)
tab.lasso.train <- table(yhat.lasso.train, train$Purchase)

pred.lasso = predict(fit.lasso, newx = Xtest, type = "response", s = bestlam)
yhat.lasso <- ifelse(pred.lasso > 0.5, 1, 0)
tab.lasso <- table(yhat.lasso, test$Purchase)
```

```r
1 - (tab.lasso.train[1,1] + tab.lasso.train[2,2])/sum(tab.lasso.train)
```

```
## [1] 0.15625
```

Misclassification rates for Lasso:

```r
m.lasso.tr <- 1 - (tab.lasso.train[1,1] + tab.lasso.train[2,2])/sum(tab.lasso.train)
m.lasso.te <- 1 - (tab.lasso[1,1] + tab.lasso[2,2])/sum(tab.lasso)
kable(data.frame(tr = m.lasso.tr, te = m.lasso.te), col.names = c("Training", "Test") )
```

Training	Test
0.15625	0.1777778

```r
1 - (tab.lasso[1,1] + tab.lasso[2,2])/sum(tab.lasso)
```

```
## [1] 0.1777778
```

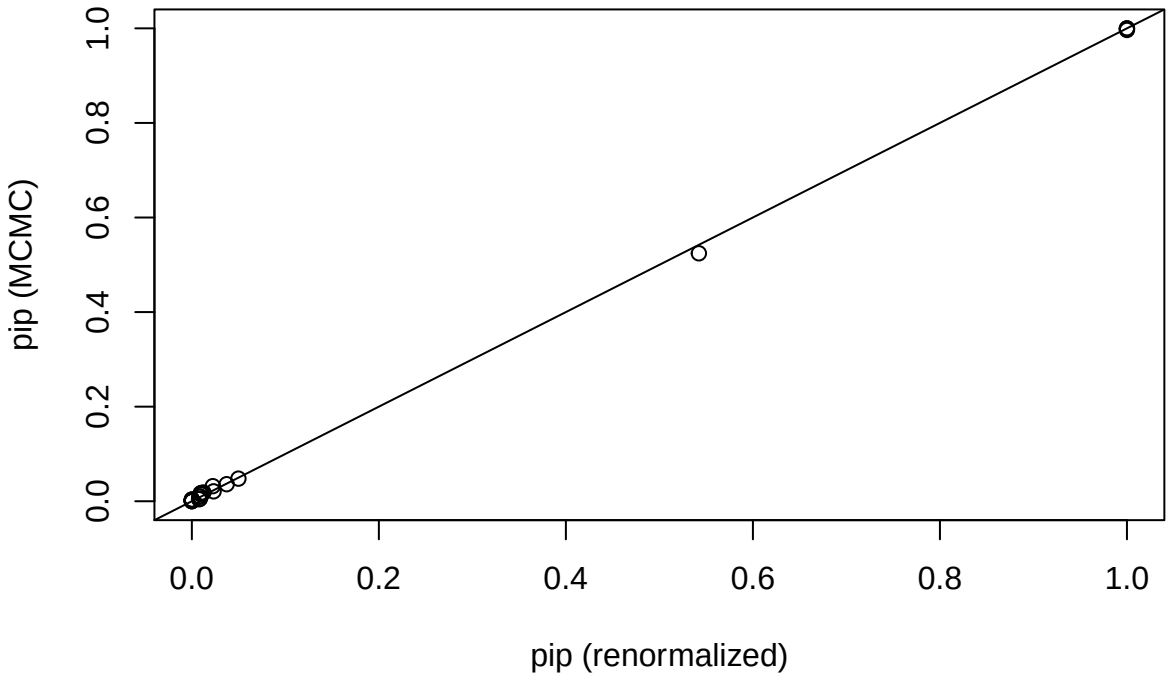
For Lasso method, the miss-classification error rate for training data is 0.156 and 0.178 for the test data. The error rates look reasonable here as well, and the test error rate is the small.

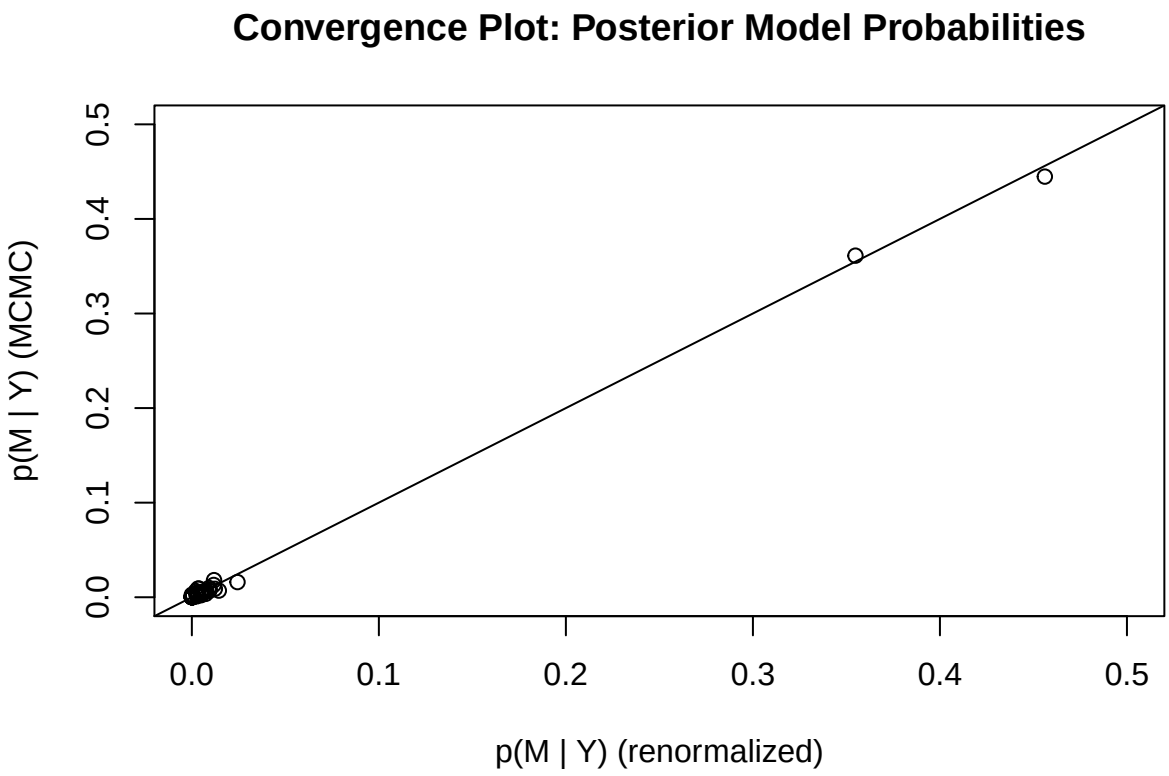
```r
coef.lasso <- predict(fit.lasso, type="coefficients", s=bestlam)[1:24,]
kable(coef.lasso[which(coef.lasso!=0)], col.names = "Coefficient Estimate")
```

	Coefficient Estimate
(Intercept)	3.0831843
StoreID2	0.1546709
StoreID4	-0.4211410
StoreID7	-0.5366488
PriceMM	-0.0102715
DiscCH	-0.6936987
LoyalCH	-5.5900446
PriceDiff	-2.2528870
Store7Yes	-0.0009451
ListPriceDiff	-0.3242000
STORE2	0.0000479
STORE4	-0.0458518

From the above we can see that variables `StoreID`, `PriceMM`, `DiscCH`, `LoyalCH`, `PriceDiff`, `Store7`, `ListPriceDiff` and `STORE2` and `STORE4`(2 values of the factor predictor) are being selected using Lasso and should be important.

```
# BMA
set.seed(521)
fit.bma = bas.glm(Purchase ~ ., data = train,
                  family = binomial(link = "logit"),
                  method='MCMC',
                  n.models=1000,
                  MCMC.iterations = 10000,
                  force.heredity = TRUE)
diagnostics(fit.bma)
```


Convergence Plot: Posterior Inclusion Probabilities
pip (MCMC)
0.0
0.2
0.4
0.6
0.8
1.0
pip (renormalized)
0.0
0.2
0.4
0.6
0.8
1.0




```
pred.bma.train = predict(fit.bma, estimator = "BMA", type = "response")$fit
yhat.bma.train <- ifelse(pred.bma.train > 0.5, 1, 0)
tab.bma.train <- table(yhat.bma.train, train$Purchase)

pred.bma = predict(fit.bma, newdata = test, estimator = "BMA", type = "response")$fit
yhat.bma <- ifelse(pred.bma > 0.5, 1, 0)
tab.bma <- table(yhat.bma, test$Purchase)
```

The convergence plots look fine.

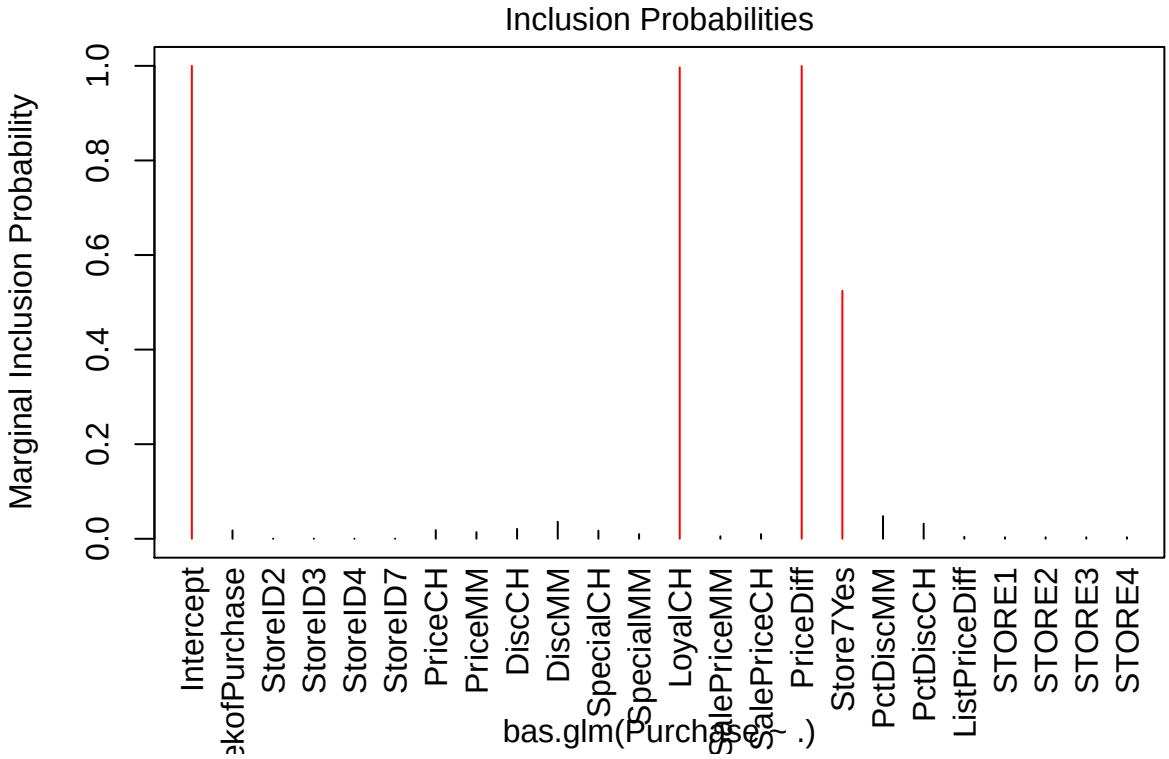
Misclassification rates for BMA:

```
m.bma.tr <- 1 - (tab.bma.train[1,1] + tab.bma.train[2,2])/sum(tab.bma.train)
m.bma.te <- 1 - (tab.bma[1,1] + tab.bma[2,2])/sum(tab.bma)
kable(data.frame(tr = m.bma.tr, te = m.bma.te), col.names = c("Training", "Test") )
```

Training	Test
0.16875	0.1666667

For BMA method, the miss-classification error rate for training data is 0.169 and 0.167 for the test data. Here, the training and test error rates are almost the same(Training error rate is very slighly higher)..

```
plot(fit.bma, which = 4)
```



For BMA, most of the predictors have a very small inclusion probability, the most important predictors are `Store7`, `LoyalCH` and `PriceDiff`.

7. Write up a summary of your findings commenting on performance in-sample and out of sample and findings regarding important variables. Are there any methods where the in-sample rates are comparable to the out of sample error rates? (For boosting report results under the default shrinkage value). Comment on how the different methods handle factors.

In this assignment, we were asked to solve a binary classification problem given a set of predictor variables. We tested multiple approaches to solving this problem, including a tree, a pruned tree using cross validation, boosting on 1000 trees, bagging, random forests, Bayesian Additive Regression Trees, and step-wise selection with AIC, LASSO, and Bayesian Model Averaging.

Using boosting, we saw similar results between the training and testing misclassification error rates. Furthermore, the test misclassification rate for boosting was significantly lower than under other methods. It had a test error rate of 8.148%, whereas the next best method had a test error rate of 16.67%. The most important variables in prediction using boosting were LoyalCH, WeekofPurchase, StoreID, and PriceDiff.

When we applied bagging, we saw an example of a model overfitting the data. The training error rate was 1.125%. However, the out-of-sample error on the testing data was 20.74%. This gap in performance rates suggests overfitting. The most important variables were similar, but not the same as boosting. They were: LoyalCH, PriceDiff, WeekofPurchase, STORE, and StoreID.

Random Forests also overfit the data. The test error rate was 22.96% in comparison to the training error rate of 7.375%. This method had the worst performance on our testing data. The most important variables

were the same as bagging, permutated in a different order: LoyalCH, PriceDiff, STORE, StoreID, and WeekofPurchase.

Using Bayesian Additive Regression Trees, we saw similar performance on the test data, with slightly worse training results. There was a 14.625% error rate on the training set, and 21.48% misclassification rate on the testing data. The most important variables were LoyalCH, PriceDiff, DiscMM, and DiscCH.

We tried a step-wise selection using AIC as criteria, next. The in-sample error rate was similar to the out-of-sample rate, 15.875% and 18.518% respectively. Variables that we deemed significant include LoyalCH, PriceMM, PriceCH, DiscCH, DiscMM, and PctDiscMM.

LASSO penalized regression also has similar error rates between the training and the test data, slightly outperforming the AIC model in both categories. It had a training error rate of 15.625% and an test error rate of 17.78%. The most important variables were LoyalCH, PriceDiff, StoreID, DiscCH, and ListPriceDiff.

Finally, we observed a Bayesian Model Averaging approach. This was the only approach where the testing rate slightly outperformed the trainig rate. While this would not be expected to happen repeatedly under BMA, these results do suggest a goodness of fit for the model and no over-fitting. The most important variables were LoyalCH, PriceDiff, and StoreID.

In summary, the boosting had the best out-of-sample performance. Bagging and Random Forests, on the other hand, overfit our data. Generally speaking, there were similar variables deemed to be important across all of the models. While all the models agreed LoyalCH and PriceDiff were important, there were slight inconsistencies on what the remaining important variables were. We would suggest using the Boosting model to solve this binary classification problem.

```
df <- data.frame(tr = c(train_confMat[[3]], m.bag.tr, m.rf.tr, m.bart.tr, m.AIC.tr, m.lasso.tr, m.bma.t
                 te = c(tstBoost_error[[3]], m.bag.te, m.rf.te, m.bart.te, m.AIC.te, m.lasso.te, m.bma.
                 row.names = c("Boosting", "Bagging", "Random Forests", "BART",
                               "AIC", "Lasso", "BMA"))
kable(df, col.names = c("Training misclassification rate", "Test misclassification rate"))
```

	Training misclassification rate	Test misclassification rate
Boosting	0.07625	0.0814815
Bagging	0.01125	0.2074074
Random Forests	0.07375	0.2296296
BART	0.14625	0.2148148
AIC	0.15875	0.1851852
Lasso	0.15625	0.1777778
BMA	0.16875	0.1666667

All models except lasso and BART take each factor predictor as a single variable. Lasso and BART seem to take each value of the factor predictors as separate predictors(after absorbing one value in the intercept).