HW8 Solution CS6220-Data Mining

```
#Load the data
load('realEstate.RData')
ls()
```

```
## [1] "realEstate"
```

```
summary(realEstate)
```

```
##        Id             Sales            SqFeet          Bed       
##  Min.   :  1.0   Min.   : 84000   Min.   : 980   Min.   :0.000  
##  1st Qu.:131.2   1st Qu.:180000   1st Qu.:1701   1st Qu.:3.000  
##  Median :261.5   Median :229900   Median :2061   Median :3.000  
##  Mean   :261.5   Mean   :277894   Mean   :2261   Mean   :3.471  
##  3rd Qu.:391.8   3rd Qu.:335000   3rd Qu.:2636   3rd Qu.:4.000  
##  Max.   :522.0   Max.   :920000   Max.   :5032   Max.   :7.000  
##       Bath            HVAC            Garage          Pool       
##  Min.   :0.000   Min.   :0.0000   Min.   :0.0   Min.   :0.00000  
##  1st Qu.:2.000   1st Qu.:1.0000   1st Qu.:2.0   1st Qu.:0.00000  
##  Median :3.000   Median :1.0000   Median :2.0   Median :0.00000  
##  Mean   :2.642   Mean   :0.8314   Mean   :2.1   Mean   :0.06897  
##  3rd Qu.:3.000   3rd Qu.:1.0000   3rd Qu.:2.0   3rd Qu.:0.00000  
##  Max.   :7.000   Max.   :1.0000   Max.   :7.0   Max.   :1.00000  
##       Year         Quality          Style           LotSize      
##  Min.   :1885   Min.   :1.000   Min.   : 1.000   Min.   : 4560  
##  1st Qu.:1956   1st Qu.:2.000   1st Qu.: 1.000   1st Qu.:17205  
##  Median :1966   Median :2.000   Median : 2.000   Median :22200  
##  Mean   :1967   Mean   :2.184   Mean   : 3.345   Mean   :24370  
##  3rd Qu.:1981   3rd Qu.:3.000   3rd Qu.: 7.000   3rd Qu.:26787  
##  Max.   :1998   Max.   :3.000   Max.   :11.000   Max.   :86830  
##     Highway      
##  Min.   :0.00000  
##  1st Qu.:0.00000  
##  Median :0.00000  
##  Mean   :0.02107  
##  3rd Qu.:0.00000  
##  Max.   :1.00000
```

```
# Can possibly transform the categorical variables into factors. However since they are binary this is 
# realEstate$HVAC <- as.factor(realEstate$HVAC)
# realEstate$Pool <- as.factor(realEstate$Pool)
# realEstate$Highway <- as.factor(realEstate$Highway)

realEstate$Y <- 0
realEstate$Y[realEstate[10]== 1] <- 1
realEstate$Y <- as.factor(realEstate$Y)

data = data.frame(realEstate[, -c(1, 10)])
summary(data)
```

```
##      Sales            SqFeet          Bed             Bath
##  Min.   : 84000   Min.   : 980   Min.   :0.000   Min.   :0.000
##  1st Qu.:180000   1st Qu.:1701   1st Qu.:3.000   1st Qu.:2.000
##  Median :229900   Median :2061   Median :3.000   Median :3.000
##  Mean   :277894   Mean   :2261   Mean   :3.471   Mean   :2.642
##  3rd Qu.:335000   3rd Qu.:2636   3rd Qu.:4.000   3rd Qu.:3.000
##  Max.   :920000   Max.   :5032   Max.   :7.000   Max.   :7.000
##       HVAC            Garage         Pool             Year
##  Min.   :0.0000   Min.   :0.0   Min.   :0.00000   Min.   :1885
##  1st Qu.:1.0000   1st Qu.:2.0   1st Qu.:0.00000   1st Qu.:1956
##  Median :1.0000   Median :2.0   Median :0.00000   Median :1966
##  Mean   :0.8314   Mean   :2.1   Mean   :0.06897   Mean   :1967
##  3rd Qu.:1.0000   3rd Qu.:2.0   3rd Qu.:0.00000   3rd Qu.:1981
##  Max.   :1.0000   Max.   :7.0   Max.   :1.00000   Max.   :1998
##      Style           LotSize         Highway        Y
##  Min.   : 1.000   Min.   : 4560   Min.   :0.00000   0:454
##  1st Qu.: 1.000   1st Qu.:17205   1st Qu.:0.00000   1: 68
##  Median : 2.000   Median :22200   Median :0.00000
##  Mean   : 3.345   Mean   :24370   Mean   :0.02107
##  3rd Qu.: 7.000   3rd Qu.:26787   3rd Qu.:0.00000
##  Max.   :11.000   Max.   :86830   Max.   :1.00000
```

Problem 1: JWHT Problem 9, page 334

(a)

```
set.seed(123)
train <- sample(nrow(data),350,replace=F)
data.train <- data[train,]
data.test <- data[-train,]
```

(b)

```
library(tree)
mytree <- tree(Y ~ ., data=data.train)
summary(mytree)
```

```
##
## Classification tree:
## tree(formula = Y ~ ., data = data.train)
## Variables actually used in tree construction:
## [1] "Sales"  "Year"   "SqFeet"
## Number of terminal nodes:  12
## Residual mean deviance:  0.1369 = 46.27 / 338
## Misclassification error rate: 0.02857 = 10 / 350
```

```
set.seed (3)
```

Training error rate: 0.02857

Number of terminal nodes: 12

(c)

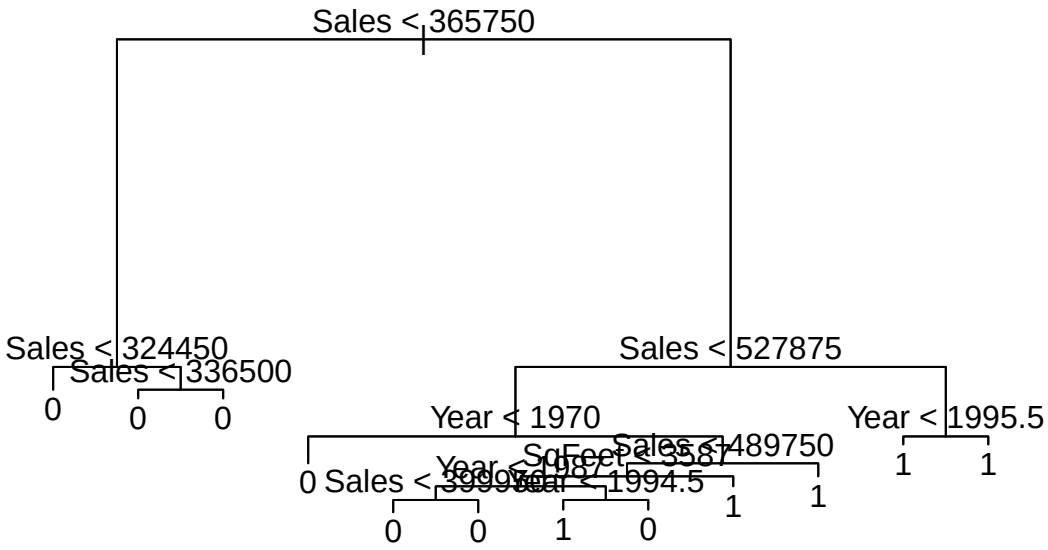
```
mytree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 350 272.400 0 ( 0.868571 0.131429 )
##     2) Sales < 365750 272  23.640 0 ( 0.992647 0.007353 )
##       4) Sales < 324450 248   0.000 0 ( 1.000000 0.000000 ) *
##       5) Sales > 324450 24  13.770 0 ( 0.916667 0.083333 )
##        10) Sales < 336500 10  10.010 0 ( 0.800000 0.200000 ) *
##        11) Sales > 336500 14   0.000 0 ( 1.000000 0.000000 ) *
##     3) Sales > 365750 78 106.800 1 ( 0.435897 0.564103 )
##       6) Sales < 527875 52  68.270 0 ( 0.634615 0.365385 )
##        12) Year < 1970 11   0.000 0 ( 1.000000 0.000000 ) *
##        13) Year > 1970 41  56.620 0 ( 0.536585 0.463415 )
##          26) Sales < 489750 34  45.230 0 ( 0.617647 0.382353 )
##            52) SqFeet < 3587 29  35.920 0 ( 0.689655 0.310345 )
##             104) Year < 1987 16  12.060 0 ( 0.875000 0.125000 )
##               208) Sales < 399950 5   6.730 0 ( 0.600000 0.400000 ) *
##               209) Sales > 399950 11   0.000 0 ( 1.000000 0.000000 ) *
##             105) Year > 1987 13  17.940 1 ( 0.461538 0.538462 )
##               210) Year < 1994.5 6   5.407 1 ( 0.166667 0.833333 ) *
##               211) Year > 1994.5 7   8.376 0 ( 0.714286 0.285714 ) *
##            53) SqFeet > 3587 5   5.004 1 ( 0.200000 0.800000 ) *
##          27) Sales > 489750 7   5.742 1 ( 0.142857 0.857143 ) *
##       7) Sales > 527875 26   8.477 1 ( 0.038462 0.961538 )
##        14) Year < 1995.5 21   0.000 1 ( 0.000000 1.000000 ) *
##        15) Year > 1995.5 5   5.004 1 ( 0.200000 0.800000 ) *
```

Node 210: For the home with sales price between 365750$ and 489750$, with SqFeet less than 3587 and built after 1987 and before 1994.5 has high quality with 0.83 probability. The prediction is based on 6 record in training dataset and the prediction devidance is: 5.407

(d)

```
plot(mytree)
text(mytree)
```



The tree shows that the high quality houses are the ones with:

1) sales price more than 527875 or
2) sales price between 489750 and 527875 and built after 1970 or
3) sales price between 365750 and 489750 and sqfeet more than 3587 or
4) sales price between 365750 and 489750 and sqfeet less than 3587 and built between 1987 and 1994.5

(e)

```
mytree.predict = predict(mytree, data.test, type="class")
table(mytree.predict,data.test$Y)
```

```
##
## mytree.predict   0   1
##              0 147   4
##              1   3  18
```

```
#test.error.rate:
(3+4)/(3+4+18+147)
```

```
## [1] 0.04069767
```

The test error rate is 4%.

(f)

```
mycv = cv.tree(mytree, FUN=prune.misclass)
mycv
```

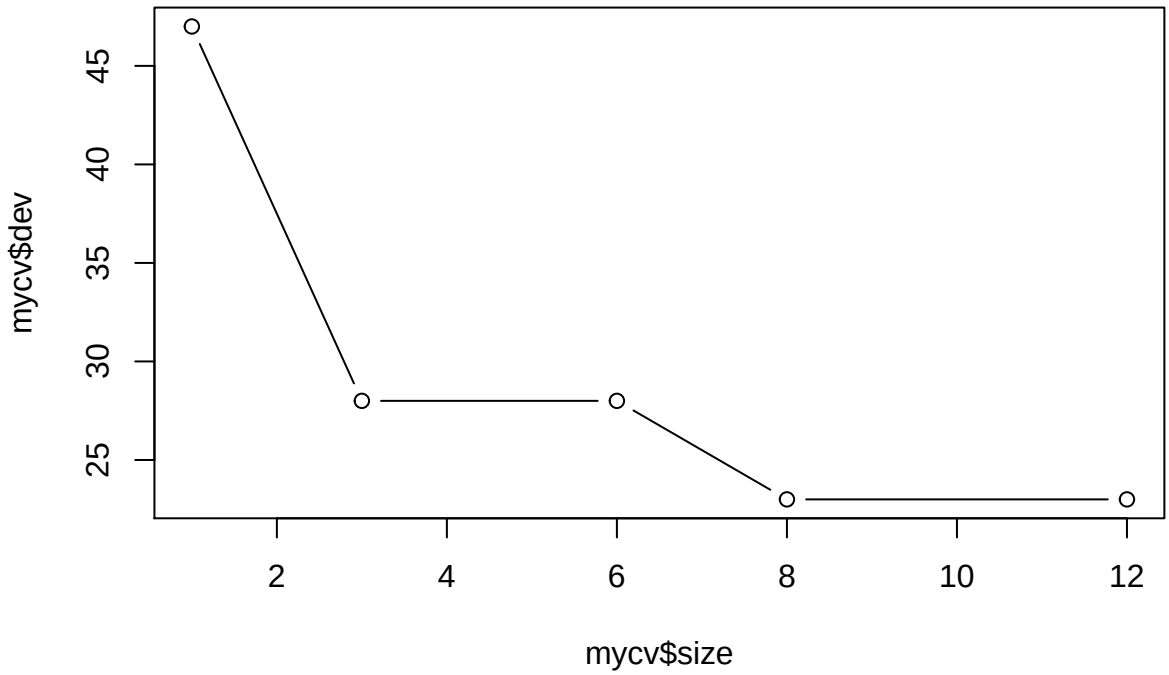
```
## $size
## [1] 12  8  6  3  1
##
```

```
## $dev
## [1] 23 23 28 28 47
##
## $k
## [1]      -Inf  0.000000  2.000000  2.666667 12.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

dev shows the cross-validation error. So, both trees with size 8 and 12 terminal nodes have the lowest cross-validation error of 23.

(g)

```
plot(mycv$size, mycv$dev ,type="b")
```



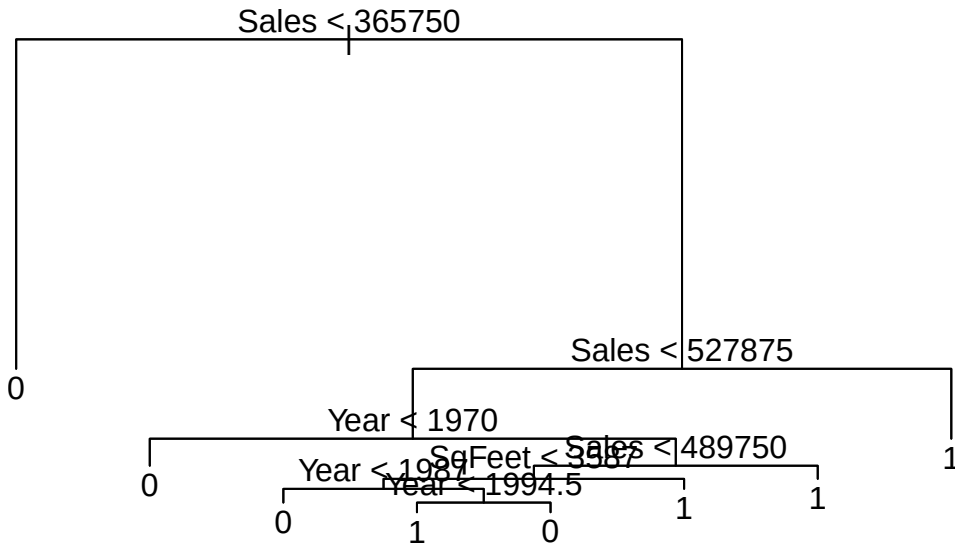
The plot shows the lowst cross-validation error rate for trees with size 8 and 12 terminal nodes.

(h)

Trees with size 8 and 12 terminal nodes. We pursure with the size 8 for smaller and simpler tree.

(i)

```
mytree.prune = prune.misclass (mytree, best = 8)
plot(mytree.prune )
text(mytree.prune, pretty =0)
```



(j)

```
summary(mytree.prune)
```

```
##
## Classification tree:
## snip.tree(tree = mytree, nodes = c(2L, 7L, 104L))
## Variables actually used in tree construction:
## [1] "Sales"  "Year"   "SqFeet"
## Number of terminal nodes:  8
## Residual mean deviance:  0.2009 = 68.7 / 342
## Misclassification error rate: 0.02857 = 10 / 350
```

In this example, the error rate of the pruned and unpruned trees are the same.

(k)

```
mytree.prune.predict = predict(mytree.prune, data.test, type="class")
table(mytree.prune.predict, data.test$Y)
```

```
##
## mytree.prune.predict   0   1
##                    0 147   4
##                    1   3  18
```

```
#test.error.rate:
(3+4)/(3+4+18+147)
```

```
## [1] 0.04069767
```

The errore rate is the same in this example as cv.tree() showed before.

Problem 2

```r
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

```r
set.seed(23)
mybagging=randomForest(Y~.,data=data.train, mtry=11,importance=TRUE)
mybagging
```

```
##
## Call:
##  randomForest(formula = Y ~ ., data = data.train, mtry = 11, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##         OOB estimate of  error rate: 6.57%
## Confusion matrix:
##     0  1 class.error
## 0 296  8  0.02631579
## 1  15 31  0.32608696
```

```r
#train.error.rate:
(8+15)/(8+15+31+296)
```

```
## [1] 0.06571429
```

```r
mybagging.predict = predict(mybagging, newdata=data.test)
table(mybagging.predict, data.test$Y)
```

```
##
## mybagging.predict   0   1
##                 0 147   5
##                 1   3  17
```

```r
#test.error.rate:
(3+5)/(3+5+17+147)
```

```
## [1] 0.04651163
```

Both of the train and the test error rates are more than a single-tree classification.

Problem 3

```r
set.seed(23)
myforest=randomForest(Y ~ ., data=data.train,mtry=3,importance=TRUE)
myforest
```

```
##
## Call:
##  randomForest(formula = Y ~ ., data = data.train, mtry = 3, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 6.86%
## Confusion matrix:
##     0  1 class.error
## 0 296  8  0.02631579
## 1  16 30  0.34782609
```

```
#train.error.rate:
(8+16)/(8+16+30+296)
```

```
## [1] 0.06857143
```

```
myforest.predict = predict(myforest,newdata=data.test)
table(myforest.predict, data.test$Y)
```

```
##
## myforest.predict   0   1
##                0 148   6
##                1   2  16
```

```
#test.error.rate:
(2+6)/(2+6+16+148)
```

```
## [1] 0.04651163
```

Again, both of the train and the test error rates are more than a single-tree classification.

Problem 4

```
library(gbm)
```

```
## Loading required package: survival
## Loading required package: lattice
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.1
```

```
#For gbm bernoulli, Y should be numeric {0,1}
data.train$Y = ifelse(data.train$Y == '0', 0, 1)
data.test$Y = ifelse(data.test$Y == '0', 0, 1)

#lambda = seq(length = 10, from = 0.001,  by = 0.05)
lambda = c(0.001, 0.005, 0.01, 0.015, 0.04, 0.06, 0.08, 0.1, 0.3, 0.5)
```

(a)

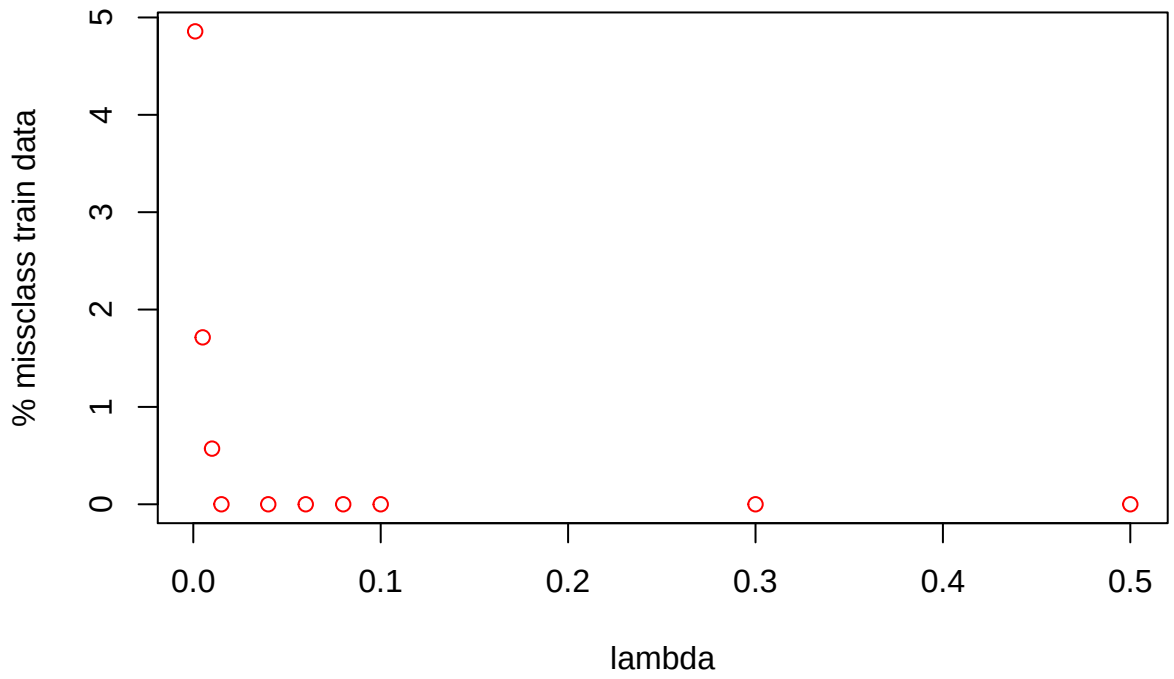
```
boost.list = list()
i = 1
for (l in lambda) {
boost.list[[i]] = gbm(Y ~ ., data = data.train, distribution= "bernoulli",
n.trees = 1000, interaction.depth = 4, shrinkage = l, verbose= F)
i = i+1
}

error.list = list()
i = 1
for (boost in boost.list) {
  boost.prediction = predict(boost, newdata=data.train, n.trees=1000, type='response')
  boost.prediction <- ifelse(boost.prediction <= 0.5, 0, 1)
  confusion = table(boost.prediction, data.train$Y)
  error.list[[i]] <- (confusion[1,2] + confusion[2,1] )* 100/sum(confusion)
  i = i+1
}

print(unlist(error.list))
```

```
##  [1] 4.8571429 1.7142857 0.5714286 0.0000000 0.0000000 0.0000000 0.0000000
##  [8] 0.0000000 0.0000000 0.0000000
```

```
plot(unlist(error.list) ~ lambda, xlab = "lambda", ylab = "% missclass train data", col = "red")
```



(b)

```
boost.list = list()
i = 1
for (l in lambda) {
boost.list[[i]] = gbm(Y ~ ., data = data.test, distribution= "bernoulli",
```

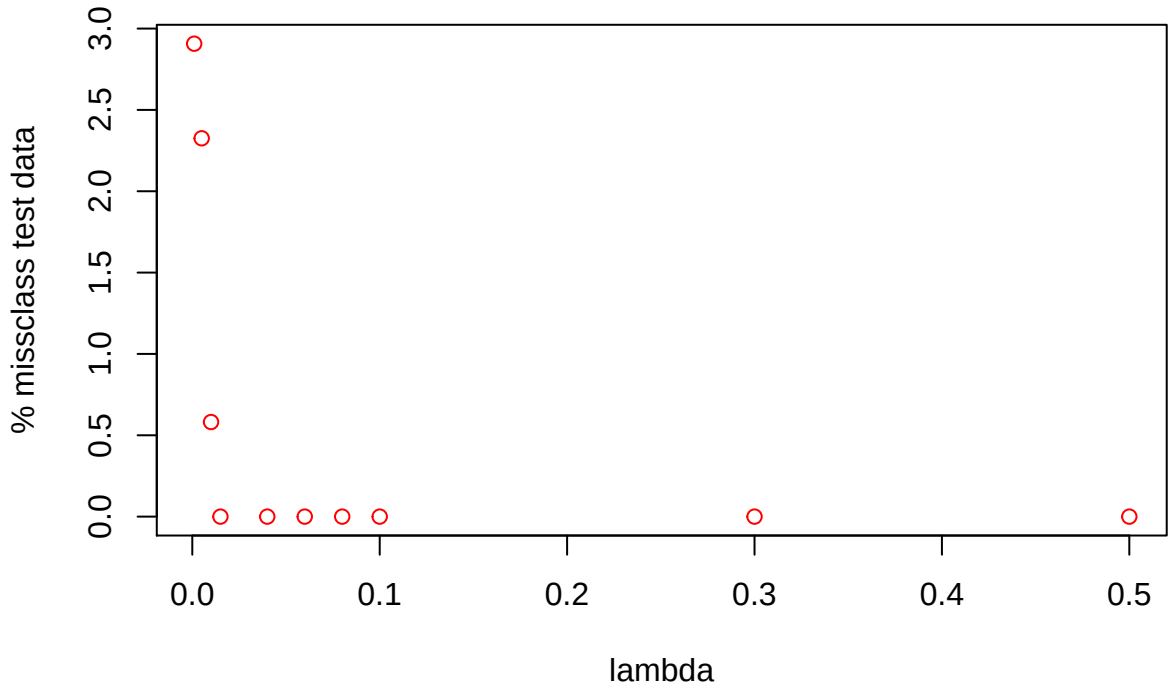
```
n.trees = 1000, interaction.depth = 4, shrinkage = l, verbose= F)
i = i+1
}

error.list = list()
i = 1
for (boost in boost.list) {
  boost.prediction = predict(boost, newdata=data.test, n.trees=1000, type='response')
  boost.prediction <- ifelse(boost.prediction <= 0.5, 0, 1)
  confusion = table(boost.prediction, data.test$Y)
  error.list[[i]] <- (confusion[1,2] + confusion[2,1] ) * 100 /sum(confusion)
  i = i+1
}

print(unlist(error.list))
```

```
##  [1] 2.9069767 2.3255814 0.5813953 0.0000000 0.0000000 0.0000000 0.0000000
##  [8] 0.0000000 0.0000000 0.0000000
```

```
plot(unlist(error.list) ~ lambda, xlab = "lambda", ylab = "% missclass test data", col = "red")
```

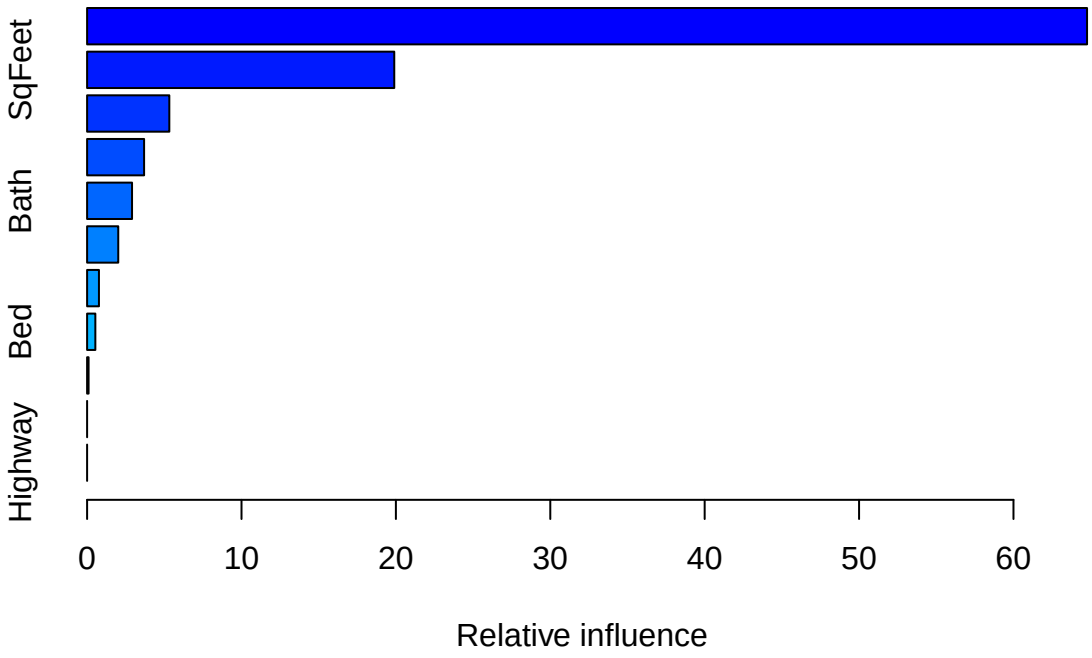


(c)

The miss classification error rate in Boosting for both the train and test data set are less than all other approaches in this HW.

(d)

```
summary(boost.list[[5]])
```




```
##              var     rel.inf
## Sales     Sales 64.76601246
## SqFeet   SqFeet 19.89351919
## Garage   Garage  5.32447565
## Year       Year  3.69077345
## Bath       Bath  2.90957351
## LotSize LotSize  2.01797931
## Style     Style  0.76763520
## Bed         Bed  0.53452599
## Pool       Pool  0.09550524
## HVAC       HVAC  0.00000000
## Highway Highway  0.00000000
```

Sales, sqFeet and year were used in other methods too but, boosting used other predictors not used in other metods too: Garage, Bath, lotsize, etc.