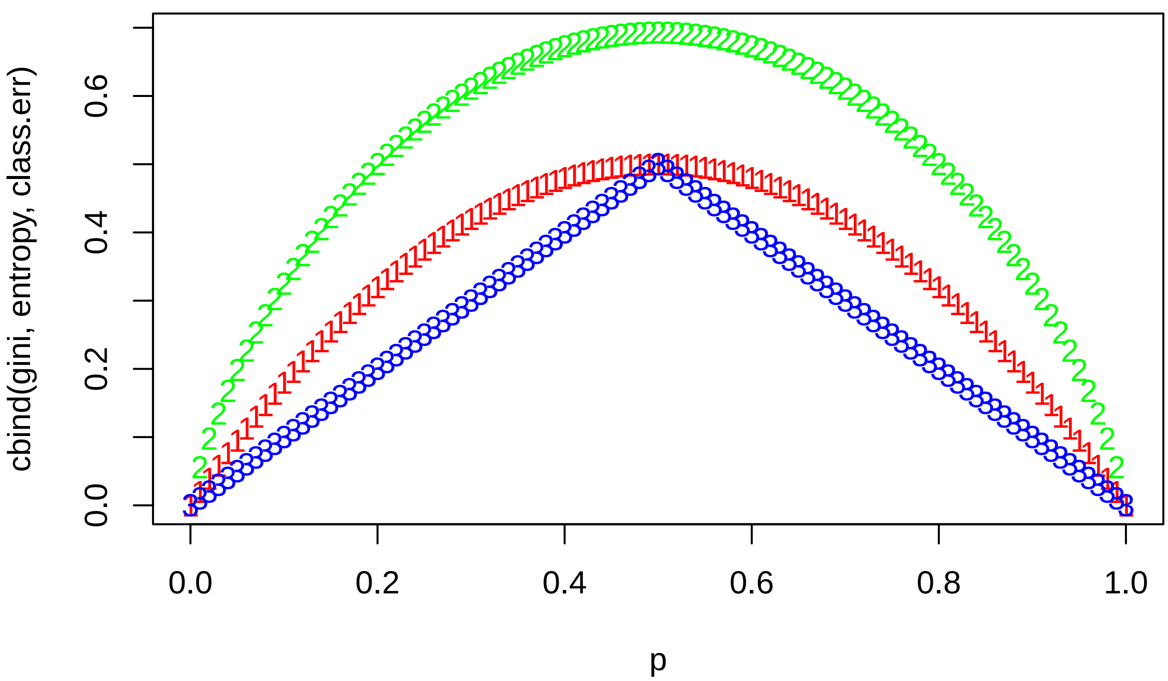
HW9 solutions

Homework 9

Chapter 8, Exercise 3

Consider the Gini index, classification error, and entropy in a simple classification setting with two classes. Create a single plot that displays each of these quantities as a function of $\hat{p}_{m1}$. The x-axis should display $\hat{p}_{m1}$, ranging from 0 to 1, and the y-axis should display the value of the Gini index, classification error, and entropy.

```
p = seq(0, 1, .01)
gini = p * (1-p) * 2
entropy = - (p * log(p) + (1-p) * log(1-p))
class.err = 1 - pmax(p, 1-p)
matplot(p, cbind(gini, entropy, class.err), col=c("red", "green", "blue"))
```



Chapter 8, Exercise 8

In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

```
library(ISLR)
attach(Carseats)
set.seed(1)

train = sample(dim(Carseats)[1], dim(Carseats)[1] / 2)
Carseats.train = Carseats[train, ]
Carseats.test = Carseats[-train, ]
```

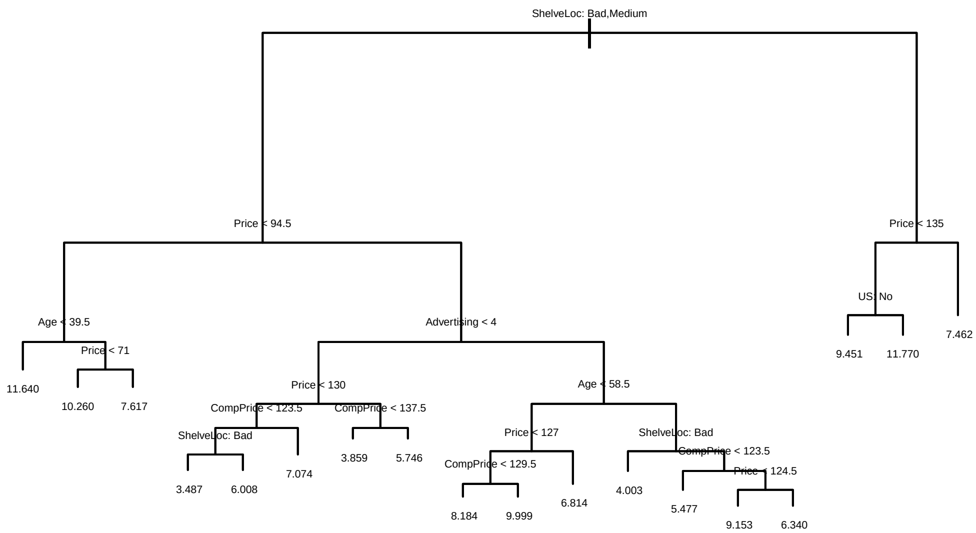
(a) Split the data set into a training set and a test set.

```
library(tree)
tree.carseats = tree(Sales~., data=Carseats.train)
summary(tree.carseats)
```

(b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats.train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Advertising" "CompPrice"
## [6] "US"
## Number of terminal nodes:  18
## Residual mean deviance:  2.167 = 394.3 / 182
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.88200 -0.88200 -0.08712  0.00000  0.89590  4.09900
```

```
plot(tree.carseats)
text(tree.carseats, pretty=0, , cex=.3)
```



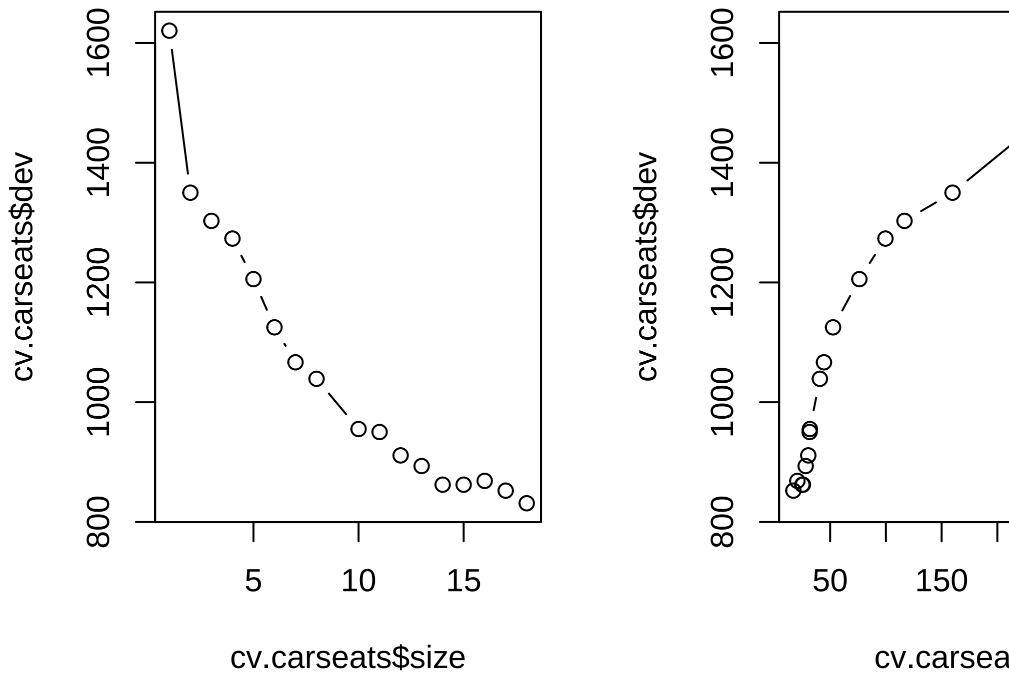

```r
pred.carseats = predict(tree.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.carseats)^2)
```

```
## [1] 4.922039
```

The test MSE is about 4.9.

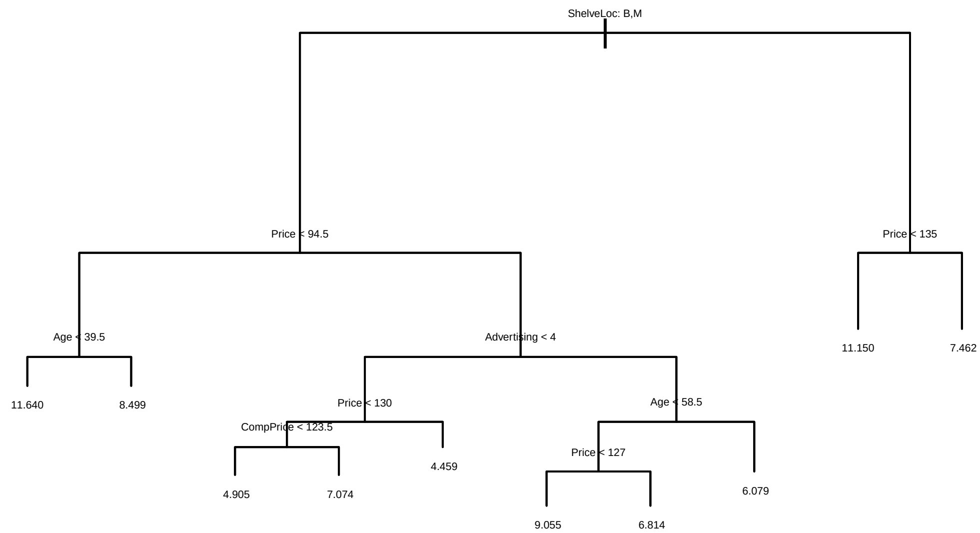
```r
cv.carseats = cv.tree(tree.carseats, FUN=prune.tree)
par(mfrow=c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type="b")
plot(cv.carseats$k, cv.carseats$dev, type="b")
```

(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning



the tree improve the test MSE?

```
# Best size = 9
pruned.carseats = prune.tree(tree.carseats, best=9)
par(mfrow=c(1, 1))
plot(pruned.carseats)
text(pruned.carseats, pretty=1, cex = .3)
```




```
pred.pruned = predict(pruned.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.pruned)^2)
```

```
## [1] 4.918134
```

Pruning the tree in this case results in a small change to the test MSE.

```
library(randomForest)
```

(d) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important.

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

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

```
bag.carseats = randomForest(Sales~., data=Carseats.train, mtry=10, ntree=500, importance=T)
bag.pred = predict(bag.carseats, Carseats.test)
mean((Carseats.test$Sales - bag.pred)^2)
```

```
## [1] 2.657296
```

```
importance(bag.carseats)
```

```
##                %IncMSE IncNodePurity
## CompPrice   23.07909904    171.185734
## Income       2.82081527     94.079825
## Advertising 11.43295625     99.098941
## Population  -3.92119532     59.818905
```

```
## Price        54.24314632    505.887016
## ShelveLoc    46.26912996    361.962753
## Age          14.24992212    159.740422
## Education    -0.07662320     46.738585
## Urban         0.08530119      8.453749
## US            4.34349223     15.157608
```

Bagging improves the test MSE to 2.6. We also see that `Price`, `ShelveLoc` and `Age` are three most important predictors of `Sale`.

```
rf.carseats = randomForest(Sales~., data=Carseats.train, mtry=5, ntree=500, importance=T)
rf.pred = predict(rf.carseats, Carseats.test)
mean((Carseats.test$Sales - rf.pred)^2)
```

(e) Use random forests to analyze this data. What test MSE do you obtain? 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.

```
## [1] 2.701665
```

```
importance(rf.carseats)
```

```
##                %IncMSE IncNodePurity
## CompPrice   19.8160444     162.73603
## Income       2.8940268     106.96093
## Advertising 11.6799573     106.30923
## Population  -1.6998805      79.04937
## Price       46.3454015     448.33554
## ShelveLoc   40.4412189     334.33610
## Age         12.5440659     169.06125
## Education    1.0762096      55.87510
## Urban        0.5703583      13.21963
## US           5.8799999      25.59797
```

In this case, random forest changes the MSE a little bit Changing m varies test MSE between 2.5 to 3. We again see that `Price`, `ShelveLoc` and `Age` are three most important predictors of `Sale`.

(f) Now analyze the data using BART, and report your results. We use the ‘`BART`‘ package, and within it the ‘`gbart()`‘ function, to fit a Bayesian additive regression tree model. The ‘`gbart()`‘ function is designed for quantitative outcome variables. (For binary outcomes, ‘`lgbart()`‘ and ‘`pgbart()`‘ are available.)

To run the ‘`gbart()`‘ function, we must first create matrices of predictors for the training and test data. We run `BART` with default settings.

```
dim(Carseats)
```

```
## [1] 400  11
```

```
names(Carseats)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

```
#install.packages("BART")
library(BART)
```

```
## Loading required package: nlme
```

```
## Loading required package: nnet
## Loading required package: survival
```

```
x <- Carseats[, 2:11]
y <- Carseats[, "Sales"]
xtrain <- x[train, ]
ytrain <- y[train]
xtest <- x[-train, ]
ytest <- y[-train]
set.seed(1)
bartfit <- gbart(xtrain, ytrain, x.test = xtest)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 200, 14, 200
## y1,yn: 2.781850, 1.091850
## x1,x[n*p]: 107.000000, 1.000000
## xp1,xp[np*p]: 111.000000, 1.000000
## *****Number of Trees: 200
## *****Number of Cut Points: 63 ... 1
## *****burn,nd,thin: 100,1000,1
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.273474,3,0.23074,7.57815
## *****sigma: 1.088371
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,14,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 3s
## trcnt,tecnt: 1000,1000
```

Next we compute the test error.

```
yhat.bart <- bartfit$yhat.test.mean
mean((ytest - yhat.bart)^2)
```

```
## [1] 1.450842
```

On this data set, the test error of BART is lower than the test error of random forests and boosting.

Now we can check how many times each variable appeared in the collection of trees.

```
ord <- order(bartfit$varcount.mean, decreasing = T)
bartfit$varcount.mean[ord]
```

```
##       Price   CompPrice  ShelveLoc2         US2  ShelveLoc1         US1
```

```
##      24.396      18.427      18.323      17.580      17.471      17.233
##   Education         Age      Urban1      Urban2      Income  Population
##      16.524      16.503      16.331      15.945      15.693      15.518
##  ShelveLoc3 Advertising
##      15.440      13.818
```