STAT4198 HW8 Min Yang

### 1.

Carseats <-read.csv("~/desktop/Carseats.csv",head=T)  
library(tree)

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

### 2.

High=ifelse(Carseats$Sales <=8," No"," Yes ")

Carseats =data.frame(Carseats[,-1] ,High)  
summary(Carseats)

## CompPrice Income Advertising Population   
## Min. : 77 Min. : 21.00 Min. : 0.000 Min. : 10.0   
## 1st Qu.:115 1st Qu.: 42.75 1st Qu.: 0.000 1st Qu.:139.0   
## Median :125 Median : 69.00 Median : 5.000 Median :272.0   
## Mean :125 Mean : 68.66 Mean : 6.635 Mean :264.8   
## 3rd Qu.:135 3rd Qu.: 91.00 3rd Qu.:12.000 3rd Qu.:398.5   
## Max. :175 Max. :120.00 Max. :29.000 Max. :509.0   
## Price ShelveLoc Age Education Urban   
## Min. : 24.0 Bad : 96 Min. :25.00 Min. :10.0 No :118   
## 1st Qu.:100.0 Good : 85 1st Qu.:39.75 1st Qu.:12.0 Yes:282   
## Median :117.0 Medium:219 Median :54.50 Median :14.0   
## Mean :115.8 Mean :53.32 Mean :13.9   
## 3rd Qu.:131.0 3rd Qu.:66.00 3rd Qu.:16.0   
## Max. :191.0 Max. :80.00 Max. :18.0   
## US High   
## No :142 No :236   
## Yes:258 Yes :164   
##   
##   
##   
##

### 3.

tree.carseats =tree(High~.,data=Carseats )

### Q4

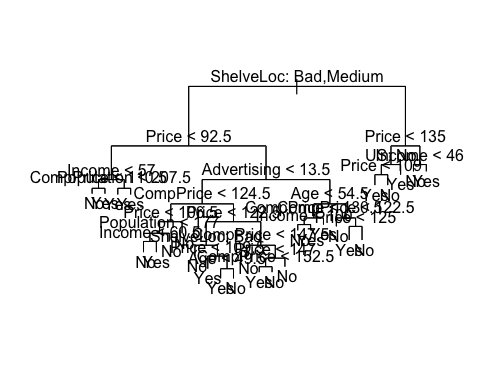
summary (tree.carseats )

##   
## Classification tree:  
## tree(formula = High ~ ., data = Carseats)  
## Variables actually used in tree construction:  
## [1] "ShelveLoc" "Price" "Income" "CompPrice" "Population"   
## [6] "Advertising" "Age" "US"   
## Number of terminal nodes: 27   
## Residual mean deviance: 0.4575 = 170.7 / 373   
## Misclassification error rate: 0.09 = 36 / 400

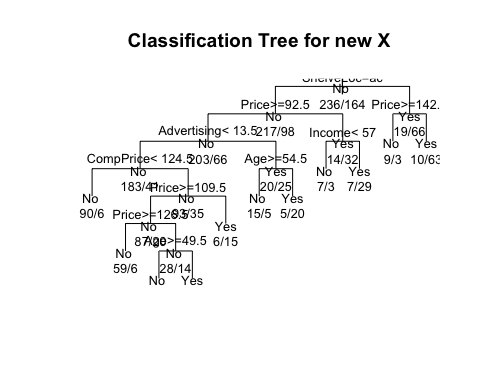
The variables used in the trees are ShelveLoc, Price, Income, CompPrice, Population, Advertising, and Age. The number of terminal nodes is 27, residual mean diviance is 0.4575, and misclassification error rate is 0.09.

### Q5

plot(tree.carseats )  
 text(tree.carseats ,pretty =0)



library(rpart)  
fit <-rpart(High~.,data = Carseats)  
plot(fit, uniform=TRUE,   
 main="Classification Tree for new X")  
text(fit, use.n=TRUE, all=TRUE, cex=.8)



new X: ShelveLoc=Good ->no ->Price=131 ->no -> no, outcome for new X is No.

tree.carseats

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 400 541.500 No ( 0.59000 0.41000 )   
## 2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )   
## 4) Price < 92.5 46 56.530 Yes ( 0.30435 0.69565 )   
## 8) Income < 57 10 12.220 No ( 0.70000 0.30000 )   
## 16) CompPrice < 110.5 5 0.000 No ( 1.00000 0.00000 ) \*  
## 17) CompPrice > 110.5 5 6.730 Yes ( 0.40000 0.60000 ) \*  
## 9) Income > 57 36 35.470 Yes ( 0.19444 0.80556 )   
## 18) Population < 207.5 16 21.170 Yes ( 0.37500 0.62500 ) \*  
## 19) Population > 207.5 20 7.941 Yes ( 0.05000 0.95000 ) \*  
## 5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )   
## 10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )   
## 20) CompPrice < 124.5 96 44.890 No ( 0.93750 0.06250 )   
## 40) Price < 106.5 38 33.150 No ( 0.84211 0.15789 )   
## 80) Population < 177 12 16.300 No ( 0.58333 0.41667 )   
## 160) Income < 60.5 6 0.000 No ( 1.00000 0.00000 ) \*  
## 161) Income > 60.5 6 5.407 Yes ( 0.16667 0.83333 ) \*  
## 81) Population > 177 26 8.477 No ( 0.96154 0.03846 ) \*  
## 41) Price > 106.5 58 0.000 No ( 1.00000 0.00000 ) \*  
## 21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )   
## 42) Price < 122.5 51 70.680 Yes ( 0.49020 0.50980 )   
## 84) ShelveLoc: Bad 11 6.702 No ( 0.90909 0.09091 ) \*  
## 85) ShelveLoc: Medium 40 52.930 Yes ( 0.37500 0.62500 )   
## 170) Price < 109.5 16 7.481 Yes ( 0.06250 0.93750 ) \*  
## 171) Price > 109.5 24 32.600 No ( 0.58333 0.41667 )   
## 342) Age < 49.5 13 16.050 Yes ( 0.30769 0.69231 ) \*  
## 343) Age > 49.5 11 6.702 No ( 0.90909 0.09091 ) \*  
## 43) Price > 122.5 77 55.540 No ( 0.88312 0.11688 )   
## 86) CompPrice < 147.5 58 17.400 No ( 0.96552 0.03448 ) \*  
## 87) CompPrice > 147.5 19 25.010 No ( 0.63158 0.36842 )   
## 174) Price < 147 12 16.300 Yes ( 0.41667 0.58333 )   
## 348) CompPrice < 152.5 7 5.742 Yes ( 0.14286 0.85714 ) \*  
## 349) CompPrice > 152.5 5 5.004 No ( 0.80000 0.20000 ) \*  
## 175) Price > 147 7 0.000 No ( 1.00000 0.00000 ) \*  
## 11) Advertising > 13.5 45 61.830 Yes ( 0.44444 0.55556 )   
## 22) Age < 54.5 25 25.020 Yes ( 0.20000 0.80000 )   
## 44) CompPrice < 130.5 14 18.250 Yes ( 0.35714 0.64286 )   
## 88) Income < 100 9 12.370 No ( 0.55556 0.44444 ) \*  
## 89) Income > 100 5 0.000 Yes ( 0.00000 1.00000 ) \*  
## 45) CompPrice > 130.5 11 0.000 Yes ( 0.00000 1.00000 ) \*  
## 23) Age > 54.5 20 22.490 No ( 0.75000 0.25000 )   
## 46) CompPrice < 122.5 10 0.000 No ( 1.00000 0.00000 ) \*  
## 47) CompPrice > 122.5 10 13.860 No ( 0.50000 0.50000 )   
## 94) Price < 125 5 0.000 Yes ( 0.00000 1.00000 ) \*  
## 95) Price > 125 5 0.000 No ( 1.00000 0.00000 ) \*  
## 3) ShelveLoc: Good 85 90.330 Yes ( 0.22353 0.77647 )   
## 6) Price < 135 68 49.260 Yes ( 0.11765 0.88235 )   
## 12) US: No 17 22.070 Yes ( 0.35294 0.64706 )   
## 24) Price < 109 8 0.000 Yes ( 0.00000 1.00000 ) \*  
## 25) Price > 109 9 11.460 No ( 0.66667 0.33333 ) \*  
## 13) US: Yes 51 16.880 Yes ( 0.03922 0.96078 ) \*  
## 7) Price > 135 17 22.070 No ( 0.64706 0.35294 )   
## 14) Income < 46 6 0.000 No ( 1.00000 0.00000 ) \*  
## 15) Income > 46 11 15.160 Yes ( 0.45455 0.54545 ) \*

### 6.

ntrain=200   
  
 set.seed(2)  
 train=sample(1: nrow(Carseats ),ntrain )  
 Carseats.test=Carseats[-train ,]  
 tree.carseats =tree(High~.,data=Carseats[train,] )  
 summary (tree.carseats )

##   
## Classification tree:  
## tree(formula = High ~ ., data = Carseats[train, ])  
## Variables actually used in tree construction:  
## [1] "ShelveLoc" "Price" "Income" "Age" "Advertising"  
## [6] "CompPrice" "Population"   
## Number of terminal nodes: 19   
## Residual mean deviance: 0.4282 = 77.51 / 181   
## Misclassification error rate: 0.105 = 21 / 200

### Q7.

High.test=High[-train ]  
  
 tree.pred=predict(tree.carseats, Carseats.test,type="class")  
 table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## No 86 27  
## Yes 30 57

(30+27) /200

## [1] 0.285

The testing missclassification rate is 0.285, and the training missclassification rate is 0.105. There is a difference between two errors may be due to small sample sets, also the data split randomly, so it is likely that there will be some data in the neighborhood.

### Q8

Trees can be viewed as providing a probability model for individuals class membership. So, at each node i, we have a probability distribution pik over the classes. The leaves of the tree give us a random sample nik from a multinomial distribution specified by pik . We can thus define the deviance of a tree, D as the sum over all leaves of Di=−2sum(kniklog(pik)). “Residual mean deviance” is the “Total residual deviance” divided by the “Number of observations” - “Number of Terminal Nodes”.

### Q9.

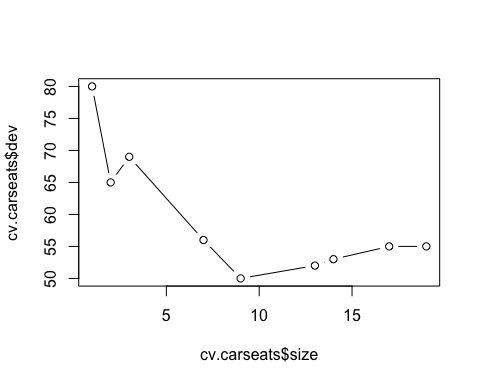
set.seed (3)  
 cv.carseats =cv.tree(tree.carseats ,FUN=prune.misclass )  
 names(cv.carseats )

## [1] "size" "dev" "k" "method"

cv.carseats

## $size  
## [1] 19 17 14 13 9 7 3 2 1  
##   
## $dev  
## [1] 55 55 53 52 50 56 69 65 80  
##   
## $k  
## [1] -Inf 0.0000000 0.6666667 1.0000000 1.7500000 2.0000000  
## [7] 4.2500000 5.0000000 23.0000000  
##   
## $method  
## [1] "misclass"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

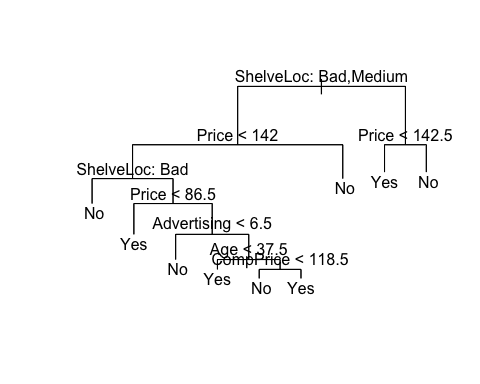
plot(cv.carseats$size ,cv.carseats$dev ,type="b")



The tree with 9 terminal nodes results in the lowest cross-validation error rate, with 50 cross-validation errors, so it’s the best size

### 10.

prune.carseats =prune.misclass (tree.carseats ,best =9)  
 plot(prune.carseats)  
 text(prune.carseats ,pretty =0)



summary(prune.carseats)

##   
## Classification tree:  
## snip.tree(tree = tree.carseats, nodes = c(159L, 6L, 8L, 38L))  
## Variables actually used in tree construction:  
## [1] "ShelveLoc" "Price" "Advertising" "Age" "CompPrice"   
## Number of terminal nodes: 9   
## Residual mean deviance: 0.8103 = 154.8 / 191   
## Misclassification error rate: 0.155 = 31 / 200

tree.pred=predict (prune.carseats , Carseats.test ,type="class")  
table(tree.pred,High.test)

## High.test  
## tree.pred No Yes   
## No 94 24  
## Yes 22 60

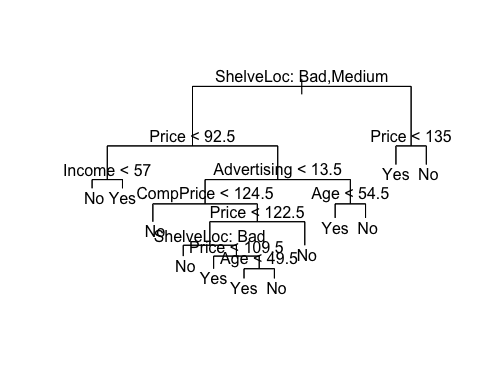
(22+24)/200

## [1] 0.23

With prunned model, the training misclassification rate is 0.155, and the testing misclassfication rate is 0.23. They are different from the one obtained from the unprunned model. While the training misclassfication rate is similar, the testing misclassfication rate for prunne model is a lot lower. So prunned model is better.

### Q11.

tree.carseats =tree(High~.,data=Carseats )  
 prune.carseats =prune.misclass (tree.carseats ,best =9)  
 plot(prune.carseats)  
 text(prune.carseats ,pretty =0)



There is difference compare to the original tree. The new model reduces the US,Education, and Population variables. The testing misclassification rate of

### Q12.

summary(prune.carseats)

##   
## Classification tree:  
## snip.tree(tree = tree.carseats, nodes = c(9L, 22L, 7L, 8L, 20L,   
## 6L, 43L, 23L))  
## Variables actually used in tree construction:  
## [1] "ShelveLoc" "Price" "Income" "Advertising" "CompPrice"   
## [6] "Age"   
## Number of terminal nodes: 12   
## Residual mean deviance: 0.7832 = 303.9 / 388   
## Misclassification error rate: 0.14 = 56 / 400

tree.pred=predict (prune.carseats , Carseats.test ,type="class")  
table(tree.pred,High.test)

## High.test  
## tree.pred No Yes   
## No 101 10  
## Yes 15 74

(10+15)/200

## [1] 0.125

I would choose the new model with 9 nodes,it has a lower testing misclassification rate of 0.125, and fewer nodes which help reduce the overfitting.

### Q13.

library(randomForest)

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

## randomForest 4.6-14

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

bag.carseat =randomForest(High~.,data=Carseats,subset =train ,  
mtry=4, ntree =200,importance=TRUE)  
 tree.pred = predict ( bag.carseat ,newdata =Carseats[-train ,],type="class")  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## No 98 21  
## Yes 18 63

mean(((tree.pred==" Yes ") -(High.test==" Yes "))^2)

## [1] 0.195

bag.carseat =randomForest(High~.,data=Carseats,subset =train ,  
mtry=6, ntree =100,importance=TRUE)  
 tree.pred = predict ( bag.carseat ,newdata =Carseats[-train ,],type="class")  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## No 96 23  
## Yes 20 61

mean(((tree.pred==" Yes ") -(High.test==" Yes "))^2)

## [1] 0.215

bag.carseat =randomForest(High~.,data=Carseats,subset =train ,  
mtry=6, ntree =500,importance=TRUE)  
 tree.pred = predict ( bag.carseat ,newdata =Carseats[-train ,],type="class")  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## No 100 19  
## Yes 16 65

mean(((tree.pred==" Yes ") -(High.test==" Yes "))^2)

## [1] 0.175

bag.carseat =randomForest(High~.,data=Carseats,subset =train ,  
mtry=3, ntree =600,importance=TRUE)  
 tree.pred = predict ( bag.carseat ,newdata =Carseats[-train ,],type="class")  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## No 97 23  
## Yes 19 61

mean(((tree.pred==" Yes ") -(High.test==" Yes "))^2)

## [1] 0.21

bag.carseat =randomForest(High~.,data=Carseats,subset =train ,  
mtry=3, ntree =100,importance=TRUE)  
 tree.pred = predict ( bag.carseat ,newdata =Carseats[-train ,],type="class")  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## No 102 19  
## Yes 14 65

mean(((tree.pred==" Yes ") -(High.test==" Yes "))^2)

## [1] 0.165

Among five combinations, my choice of mtry = 6 and ntree=100 gives the lowest testing misclassification rate of 0.155. mtry is the number of variables randomly sampled as candidates at each split. ntree is the number of trees to grow.

### Q14.

library(gbm)

## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3

set.seed (1)  
 boost.carseat =gbm((High==" Yes ")~.,data=Carseats[train ,],  
distribution="bernoulli", n.trees=8000, interaction.depth =2)

tree.pred = predict(boost.carseat,newdata =Carseats[-train ,],n.trees=5000,  
type="link")  
  
tree.pred=tree.pred>0.5  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## FALSE 111 26  
## TRUE 5 58

mean(((tree.pred=="TRUE") -(High.test==" Yes "))^2)

## [1] 0.155

tree.pred = predict(boost.carseat,newdata =Carseats[-train ,],n.trees=4000,  
type="link")  
  
tree.pred=tree.pred>0.5  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## FALSE 113 30  
## TRUE 3 54

mean(((tree.pred=="TRUE") -(High.test==" Yes "))^2)

## [1] 0.165

tree.pred = predict(boost.carseat,newdata =Carseats[-train ,],n.trees=3000,  
type="link")  
  
tree.pred=tree.pred>0.5  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## FALSE 113 35  
## TRUE 3 49

mean(((tree.pred=="TRUE") -(High.test==" Yes "))^2)

## [1] 0.19

tree.pred = predict(boost.carseat,newdata =Carseats[-train ,],n.trees=8000,  
type="link")  
  
tree.pred=tree.pred>0.5  
table(tree.pred ,High.test)

## High.test  
## tree.pred No Yes   
## FALSE 109 21  
## TRUE 7 63

mean(((tree.pred=="TRUE") -(High.test==" Yes "))^2)

## [1] 0.14

The best number of boosted trees is 5000 with a testing missclassification rate of 0.155. THe larger the n.trees, the lower the missclassification rate.

### Q15.

Boosting worked best for the dataset careseats as it has a lower missclassfication rate. With bagging, we lose the simple interpretation of classification tree, so the final classifier is a weighted sum of trees which does not really represent by a single tree. While with AdaBoost, the computation is straightforward, and since we grow small trees,each step can be donw relatively quickly.