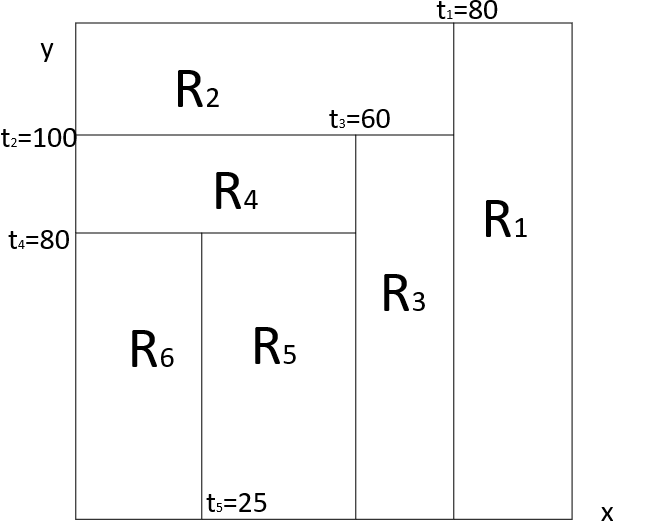
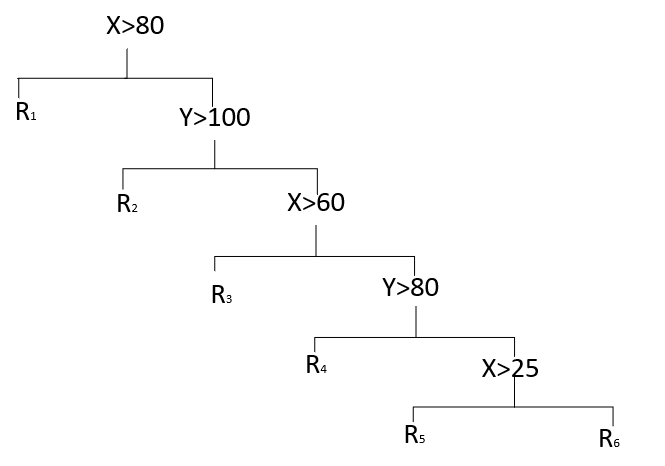
**Chapter 8: Tree-Based Methods**

**1**





**2**

对于深度为1的树（或树桩），对训练集中的所有，令。

接下来，对，

(a)对训练数据建立一棵有一个分裂点的树。

(b)将压缩后的新树加入模型以更新：

(c)更新残差

重复该过程直到次，

(a)建立一棵有一个分裂点的树。

(b)

**3**

> p = seq(0, 1, 0.01)

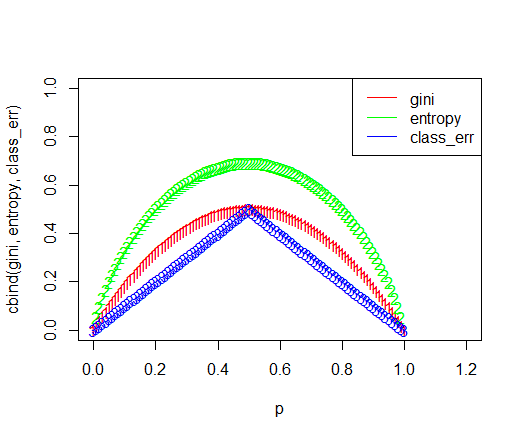
> gini = p \* (1 - p) \* 2

> entropy = -(p \* log(p) + (1 - p) \* log(1 - p))

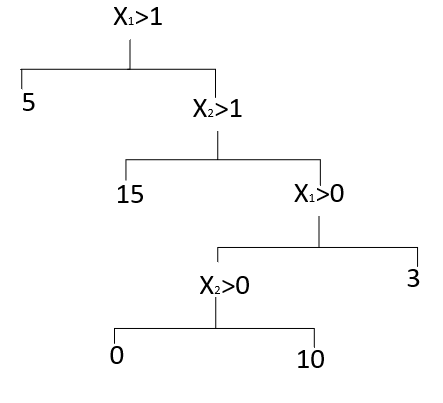
> class\_err = 1 - pmax(p, 1 - p)

> matplot(p, cbind(gini, entropy, class\_err), col = c("red", "green", "blue"),xlim = c(0,1.2),ylim = c(0,1))

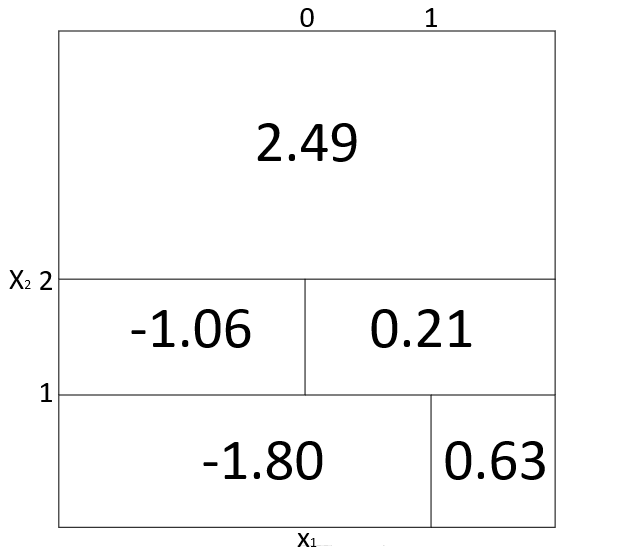
> legend("topright", c("gini", "entropy", "class\_err"), col = c("red" , "green" , "blue"),cex = 1, lty = 1)



**4(a)**



**4(b)**



**5**

1. Majority vote approach

P(Class is Red|X)大于0.5的估计有6个，超过一半，根据majority vote approach，分类结果为红球。

2．Average Probability

> (0.1+0.15+0.2+0.2+0.55+0.6+0.6+0.65+0.7+0.75)/10

[1] 0.45

平均概率为0.45，小于0.5，分类结果为绿球。

**6**

见本章8.1.1部分内容。

**7**

> library(randomForest)

> set.seed(1)

> train = sample(dim(Boston)[1], dim(Boston)[1]/2)

> X.train = Boston[train, -14]

> X.test = Boston[-train, -14]

> Y.train = Boston[train, 14]

> Y.test = Boston[-train, 14]

> p = dim(Boston)[2] - 1

> p.2 = p/2

> p.sq = sqrt(p)

> rf.boston.p = randomForest(X.train, Y.train, xtest = X.test, ytest = Y.test, mtry = p, ntree = 500)

> rf.boston.p.2 = randomForest(X.train, Y.train, xtest = X.test, ytest = Y.test, mtry = p.2, ntree = 500)

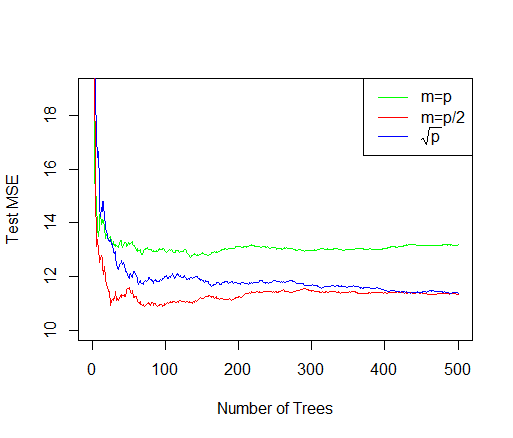
> rf.boston.p.sq = randomForest(X.train, Y.train, xtest = X.test, ytest = Y.test,mtry = p.sq, ntree = 500)

> plot(1:500, rf.boston.p$test$mse, col = "green", type = "l", xlab = "Number of Trees",ylab = "Test MSE", ylim = c(10, 19))

> lines(1:500, rf.boston.p.2$test$mse, col = "red", type = "l")

> lines(1:500, rf.boston.p.sq$test$mse, col = "blue", type = "l")

> legend("topright", c("m=p", "m=p/2", expression(m=sqrt(p))), col = c("green", "red", "blue"),cex = 1, lty = 1)



变量数目相同时，随着树的数目增加，测试均方误差呈减小的趋势直到趋于平稳。树的数目相同时，变量数目为p/2即6时随机森林模型的测试均方误差最小，变量数目为p即13时随机森林模型的测试均方误差最大。

**8(a)**

> library(ISLR)

> attach(Carseats)

> set.seed(1)

> train = sample(dim(Carseats)[1], dim(Carseats)[1]/2)

> Carseats.train = Carseats[train, ]

> Carseats.test = Carseats[-train, ]

**8(b)**

> library(tree)

> summary(tree.carseats)

Regression tree:

tree(formula = Sales ~ ., data = Carseats, subset = train)

Variables actually used in tree construction:

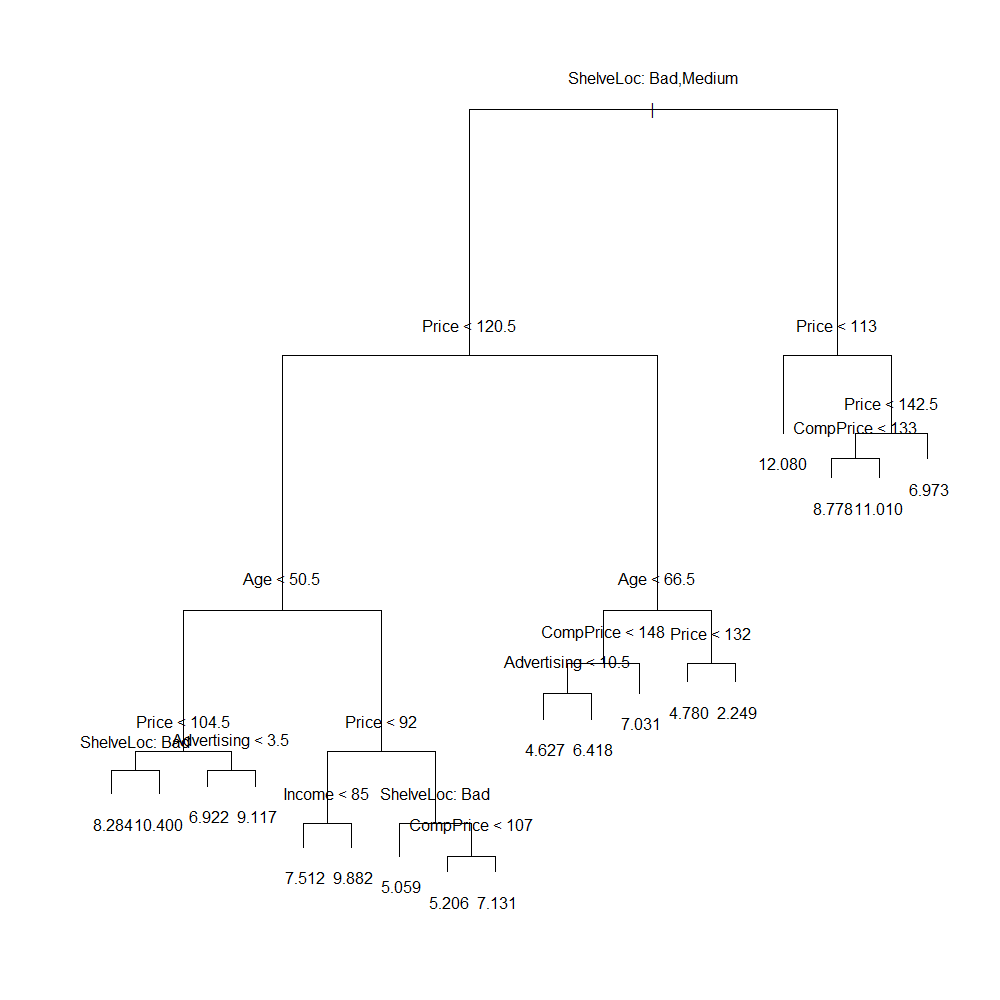
[1] "ShelveLoc" "Price" "Age" "Advertising" "Income" "CompPrice"

Number of terminal nodes: 18

Residual mean deviance: 2.36 = 429.5 / 182

> plot(tree.carseats)

> text(tree.carseats, pretty = 0)



> pred.carseats = predict(tree.carseats, Carseats.test)

> mean((Carseats.test$Sales - pred.carseats)^2)

[1] 4.148897

**8(c)**

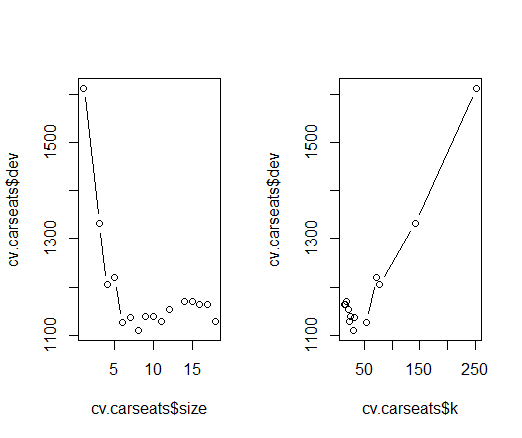
> set.seed(1)

> 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")



终端结点数为9时，交叉验证错误率最低。

> pruned.carseats = prune.tree(tree.carseats, best = 9)

> par(mfrow = c(1, 1))

> plot(pruned.carseats)

> text(pruned.carseats, pretty = 0)

> pruned.carseats = prune.tree(tree.carseats, best = 9)

> pruned.carseats.pred = predict(pruned.carseats, Carseats.test)

> mean((Carseats.test$Sales - pruned.carseats.pred)^2)

[1] 4.993124

测试均方误差增大了。

**8(d)**

> set.seed(1)

> bag.carseats = randomForest(Sales ~ .,Carseats, subset = train,mtry = 10, ntree = 500,importance = T)

> bag.carseats.pred = predict(bag.carseats, Carseats.test)

> mean((Carseats.test$Sales - bag.carseats.pred)^2)

[1] 2.554292

测试均方误差为2.554292。

> importance(bag.carseats)

%IncMSE IncNodePurity

CompPrice 14.032030 129.568747

Income 5.523038 75.448682

Advertising 13.571285 131.246840

Population 1.968853 63.042648

Price 56.863812 504.158108

ShelveLoc 44.720455 323.055042

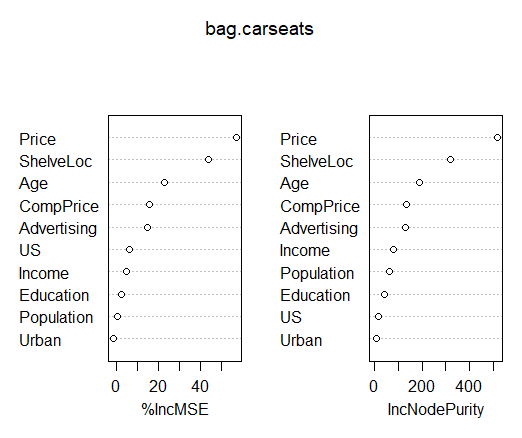
Age 22.225468 194.915976

Education 4.823966 40.810991

Urban -1.902185 8.746566

US 6.632887 14.599565

> varImpPlot(bag.carseats)



Price和ShelveLoc是最重要的两个变量，Age，CompPrice以及Advertising是三个比较重要的变量。

**8(e)**

> set.seed(1)

> rf.carseats.mse = 0

> for (i in 1:10) {

+ rf.carseats = randomForest(Sales ~ .,Carseats, subset = train, mtry = i, ntree = 500,importance = T)

+ rf.pred = predict(rf.carseats, Carseats.test)

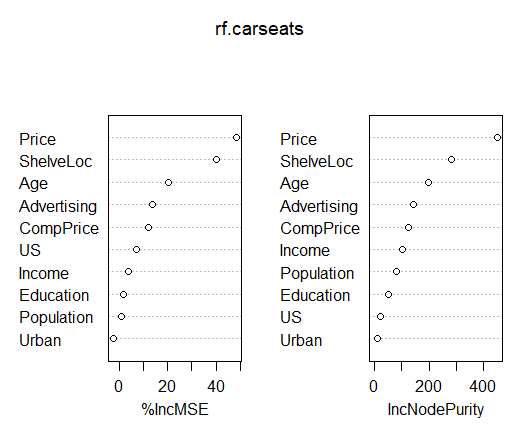
+ rf.carseats.mse[i] = mean((Carseats.test$Sales - rf.pred)^2)

+ #varImpPlot(rf.carseats)

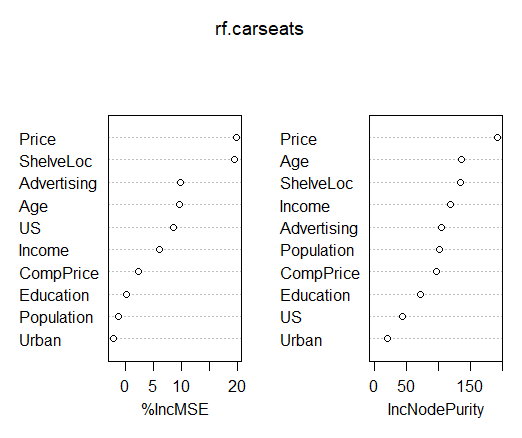
+ }

> rf.carseats.mse

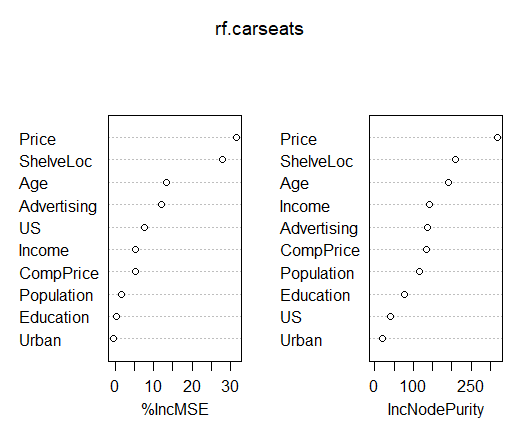
[1] 5.079480 3.809142 3.320381 3.088756 2.824119 2.742526 2.632177 2.607881 2.571821 2.551485



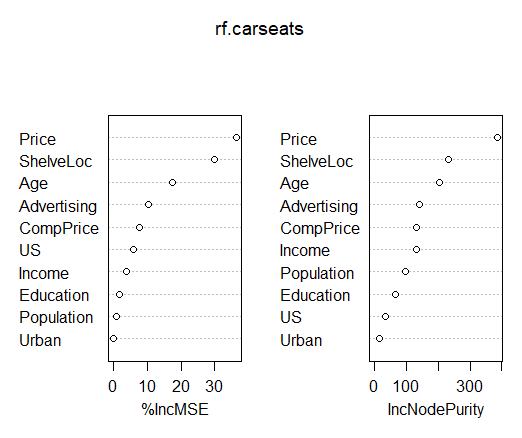
mtry = 1



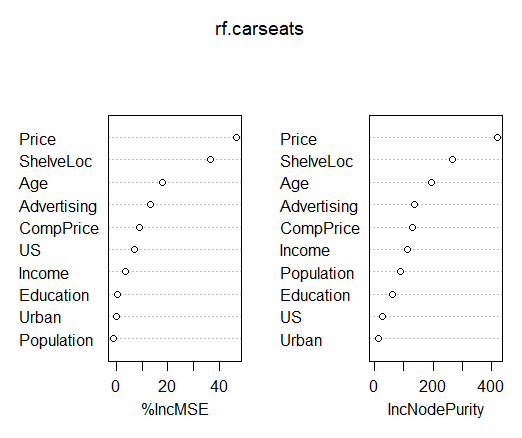
mtry = 1



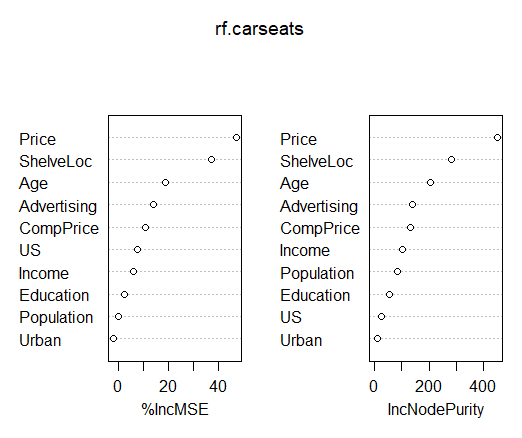
mtry = 2



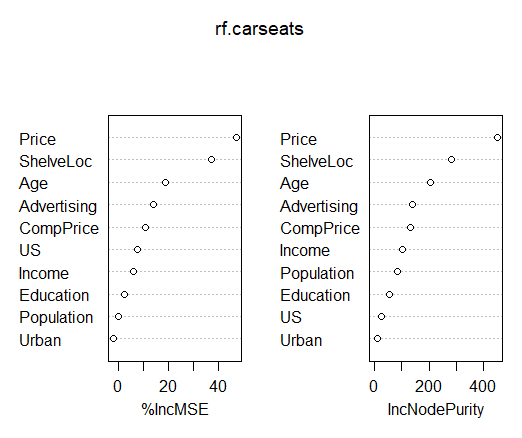
mtry = 3



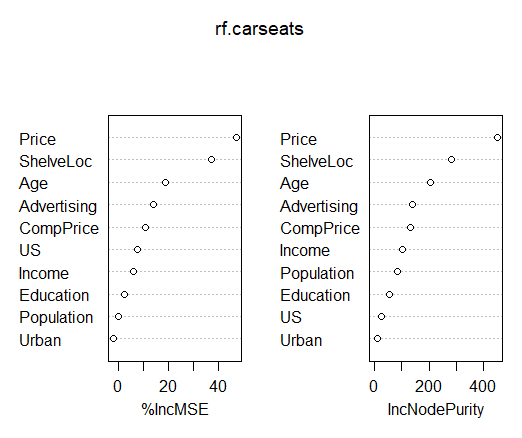
mtry = 4



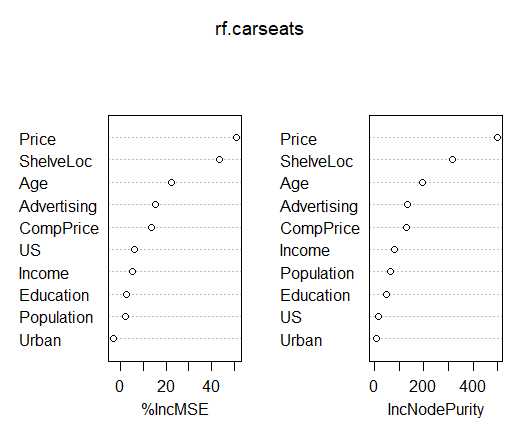
mtry = 5



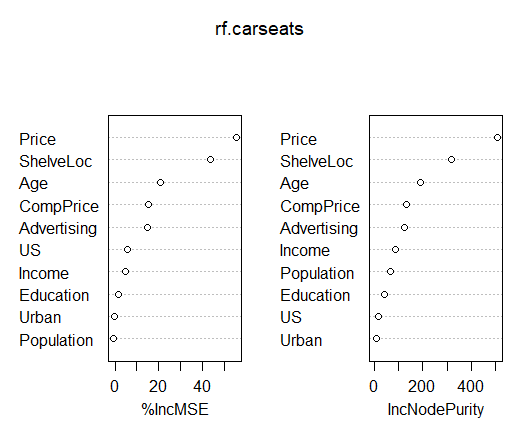
mtry = 6



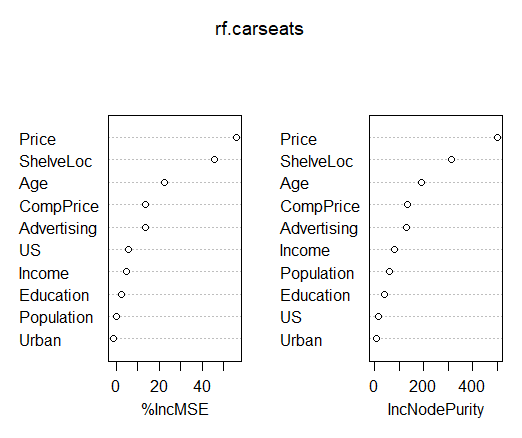
mtry = 7



mtry = 8



mtry = 9



mtry = 10

随着终端结点数从1增加到10，测试均方误差在逐渐减小。Price和ShelveLoc是最重要的两个变量，Age，CompPrice以及Advertising是三个比较重要的变量。

**9(a)**

> library(ISLR)

> attach(OJ)

> set.seed(1)

> train = sample(dim(OJ)[1], 800)

> OJ.train = OJ[train, ]

> OJ.test = OJ[-train, ]

**9(b)**

> library(tree)

> oj.tree = tree(Purchase ~ ., data = OJ.train)

> summary(oj.tree)

Classification tree:

tree(formula = Purchase ~ ., data = OJ.train)

Variables actually used in tree construction:

[1] "LoyalCH" "PriceDiff" "SpecialCH" "ListPriceDiff"

Number of terminal nodes: 8

Residual mean deviance: 0.7305 = 578.6 / 792

Misclassification error rate: 0.165 = 132 / 800

实际用到的变量有LoyalCH，PriceDiff，SpecialCH以及ListPriceDiff。Training error rate是0.165。包含8个terminal nodes。

**9(c)**

> oj.tree

node), split, n, deviance, yval, (yprob)

\* denotes terminal node

1) root 800 1064.00 CH ( 0.61750 0.38250 )

2) LoyalCH < 0.508643 350 409.30 MM ( 0.27143 0.72857 )

4) LoyalCH < 0.264232 166 122.10 MM ( 0.12048 0.87952 )

8) LoyalCH < 0.0356415 57 10.07 MM ( 0.01754 0.98246 ) \*

9) LoyalCH > 0.0356415 109 100.90 MM ( 0.17431 0.82569 ) \*

5) LoyalCH > 0.264232 184 248.80 MM ( 0.40761 0.59239 )

10) PriceDiff < 0.195 83 91.66 MM ( 0.24096 0.75904 )

20) SpecialCH < 0.5 70 60.89 MM ( 0.15714 0.84286 ) \*

21) SpecialCH > 0.5 13 16.05 CH ( 0.69231 0.30769 ) \*

11) PriceDiff > 0.195 101 139.20 CH ( 0.54455 0.45545 ) \*

3) LoyalCH > 0.508643 450 318.10 CH ( 0.88667 0.11333 )

6) LoyalCH < 0.764572 172 188.90 CH ( 0.76163 0.23837 )

12) ListPriceDiff < 0.235 70 95.61 CH ( 0.57143 0.42857 ) \*

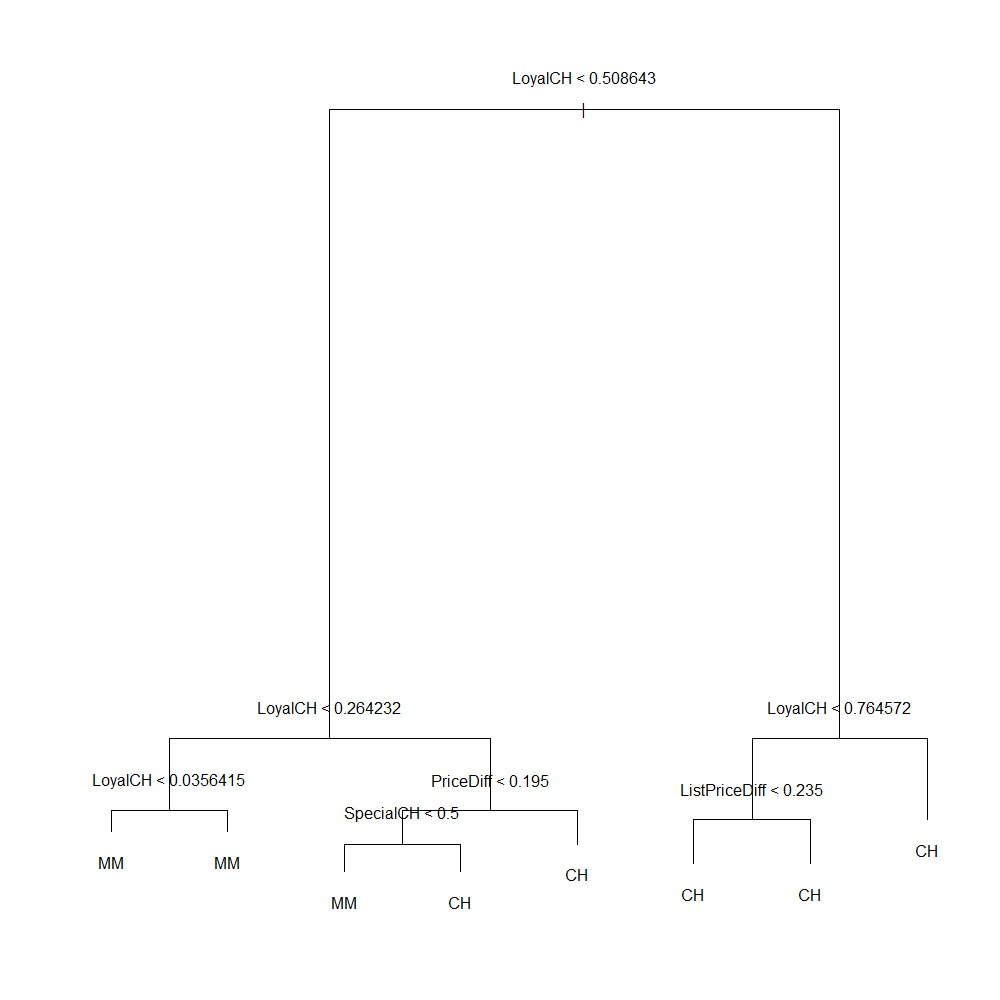
13) ListPriceDiff > 0.235 102 69.76 CH ( 0.89216 0.10784 ) \*

7) LoyalCH > 0.764572 278 86.14 CH ( 0.96403 0.03597 ) \*

考虑结点8) LoyalCH < 0.0356415 57 10.07 MM ( 0.01754 0.98246 ) \*

这个结点引出了终端结点。分裂规则为LoyalCH < 0.0356415，这一分支上观测值的数量为57，偏差为10.07，这一分支的整体预测为MM，这一分支取CH和MM的观测值的比例分别为0.01754和0.98246。

**9(d)**



LoyalCH是最重要的变量。如果LoyalCH<0.0356415，那么Purchase的预测值为MM， LoyalCH<0.264232且PriceDiff<0.195且SpecialCH<0.5，那么Purchase的预测值为MM。剩下的情况，Purchase的预测值都为CH。

**9(e)**

> oj.pred = predict(oj.tree, OJ.test, type = "class")

> table(OJ.test$Purchase, oj.pred)

oj.pred

CH MM

CH 147 12

MM 49 62

> mean(OJ.test$Purchase != oj.pred)

[1] 0.2259259

Test error rate为0.2259259。

**9(f)**

> cv.oj = cv.tree(oj.tree, FUN = prune.tree)

> cv.oj

$size

[1] 8 7 6 5 4 3 2 1

$dev

[1] 689.1001 685.8030 654.9314 653.7774 666.8890 721.2494 733.6936 1066.6499

$k

[1] -Inf 11.20965 14.72877 17.88334 23.55203 38.37537 43.02529 337.08200

$method

[1] "deviance"

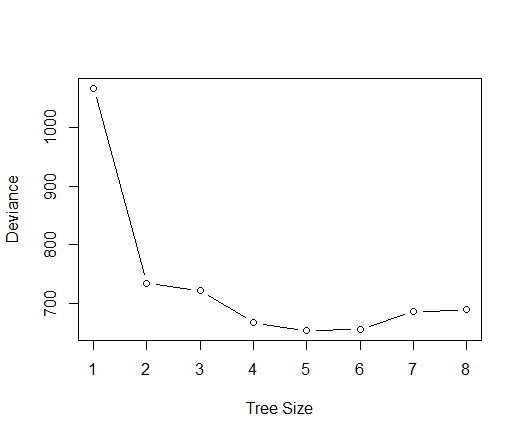
attr(,"class")

[1] "prune" "tree.sequence"

size=5时，交叉验证错误率最低，即终端结点数为5时，模型最好。

**9(g)**

> plot(cv.oj$size, cv.oj$dev, type = "b", xlab = "Tree Size", ylab = "Deviance")



**9(h)**

size=5时，交叉验证错误率最低。

**9(i)**

> oj.prune = prune.tree(oj.tree, best = 5)

**9(j)**

> summary(oj.prune)

Classification tree:

snip.tree(tree = oj.tree, nodes = 4:5)

Variables actually used in tree construction:

[1] "LoyalCH" "ListPriceDiff"

Number of terminal nodes: 5

Residual mean deviance: 0.7829 = 622.4 / 795

Misclassification error rate: 0.1825 = 146 / 800

Training error rate为0.1825。剪枝后Training error rate更高了。

**9(k)**

> oj.prune.pred = predict(oj.prune,OJ.test,type = "class")

> mean(oj.prune.pred != OJ.test$Purchase)

[1] 0.2592593

Test error rate也变高了。

**10(a)**

> library(ISLR)

> adjusted\_Hitters = Hitters[-which(is.na(Hitters$Salary)), ]

> adjusted\_Hitters$Salary = log(adjusted\_Hitters$Salary)

**10(b)**

> adjusted\_Hitters.train = adjusted\_Hitters[1:200, ]

> adjusted\_Hitters.test = adjusted\_Hitters[-(1:200), ]

**10(c)**

> library(gbm)

> set.seed(1000)

> pow = seq(-10, -0.2, by = 0.1)

> lambda = 10^pow

> length.lambda = length(lambda)

> train.error = rep(NA, length.lambda)

> test.error = rep(NA, length.lambda)

> for (i in 1:length.lambda) {

+ boost.hitters = gbm(Salary ~ ., data = adjusted\_Hitters.train, distribution = "gaussian",n.trees = 1000, shrinkage = lambda[i])

+ train.pred = predict(boost.hitters, adjusted\_Hitters.train, n.trees = 1000)

+ test.pred = predict(boost.hitters, adjusted\_Hitters.test, n.trees = 1000)

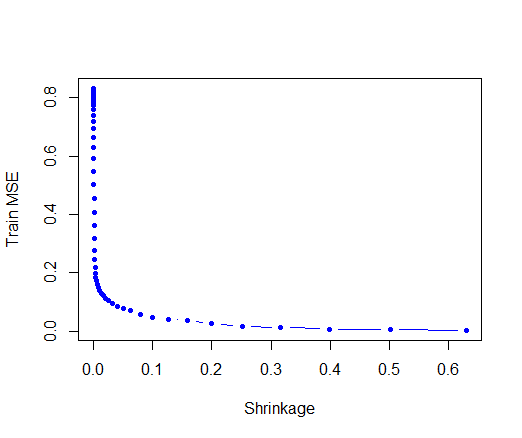
+ train.error[i] = mean((adjusted\_Hitters.train$Salary - train.pred)^2)

+ test.error[i] = mean((adjusted\_Hitters.test$Salary - test.pred)^2)

+ }

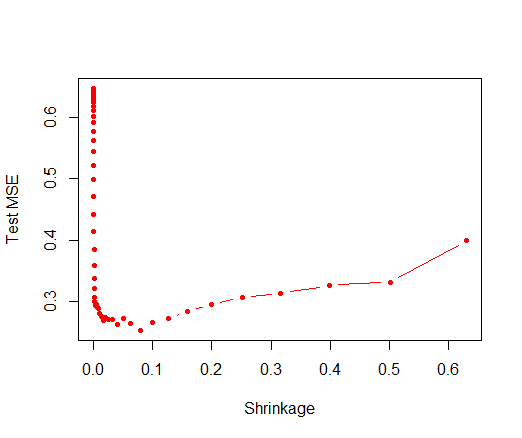
>

> plot(lambda, train.error, type = "b", xlab = "Shrinkage", ylab = "Train MSE",col = "blue", pch = 20)



**10(d)**

> plot(lambda, test.error, type = "b", xlab = "Shrinkage", ylab = "Test MSE",col = "red", pch = 20)



**10(e)**

> lambda[which.min(test.error)]

[1] 0.07943282

> min(test.error)

[1] 0.2525246

取值为0.07943282时，对应有最小的test MSE为0.2525246。

对于第三章的线性回归，代码及结果如下：

> lm.fit = lm(Salary ~ ., data = adjusted\_Hitters.train)

> lm.pred = predict(lm.fit, adjusted\_Hitters.test)

> mean((adjusted\_Hitters.test$Salary - lm.pred)^2)

[1] 0.4917959

对于第六章的lasso，代码及结果如下：

> library(glmnet)

> x.train = model.matrix(Salary~.,adjusted\_Hitters.train)

> y.train = adjusted\_Hitters.train$Salary

> x.test = model.matrix(Salary~.,adjusted\_Hitters.test)

> set.seed(1000)

> cv.out = cv.glmnet(x.train,y.train,alpha = 1)

> bestlam = cv.out$lambda.min

> lasso.mod = glmnet(x.train,y.train,alpha = 1)

> lasso.pred = predict(lasso.mod,s = bestlam,newx = x.test)

> mean((adjusted\_Hitters.test$Salary - lasso.pred)^2)

[1] 0.4713617

无论是简单线性回归还是lasso，test MSE都比boosting高很多。

**10(f)**

> boost.best = gbm(Salary ~ ., data = adjusted\_Hitters.train, distribution = "gaussian",n.trees = 1000, shrinkage = lambda[which.min(test.error)])

> summary(boost.best)

var rel.inf

CAtBat CAtBat 16.1059191

CWalks CWalks 9.9098705

CRBI CRBI 9.3131264

PutOuts PutOuts 8.0988415

CRuns CRuns 7.4182320

Walks Walks 7.0052184

Years Years 6.7854549

CHmRun CHmRun 4.9787970

Hits Hits 4.9337721

AtBat AtBat 4.3404519

Assists Assists 4.1430906

RBI RBI 3.6874789

CHits CHits 3.5868088

HmRun HmRun 3.3923631

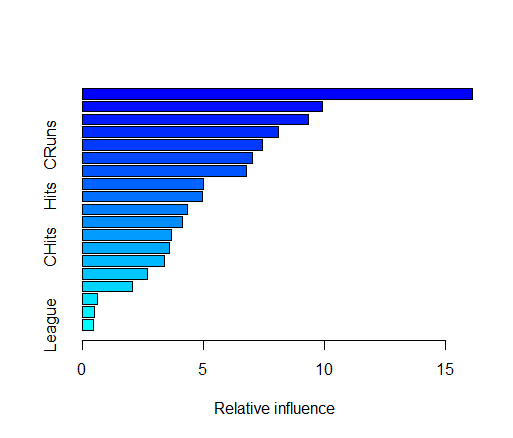
Errors Errors 2.6791088

Runs Runs 2.0635819

Division Division 0.6083552

NewLeague NewLeague 0.4813175

League League 0.4682114



最重要的变量是CAtBat，其次是CWalks， CRBI和PutOuts。

**10(g)**

> library(randomForest)

> set.seed(36)

> bag.hitters = randomForest(Salary~., data = adjusted\_Hitters.train, ntree = 1000, mtry = 19)

> bag.pred = predict(bag.hitters, adjusted\_Hitters.test)

> mean((adjusted\_Hitters.test$Salary - bag.pred)^2)

[1] 0.2294973

test MSE为0.2294973。

**11(a)**

> library(ISLR)

> Caravan$Purchase = ifelse(Caravan$Purchase == "Yes", 1, 0)

> Caravan.train = Caravan[1:1000, ]

> Caravan.test = Caravan[-(1:1000), ]

**11(b)**

> library(gbm)

> set.seed(10)

> boost.caravan = gbm(Purchase ~ ., data = Caravan.train, n.trees = 1000, shrinkage = 0.01,distribution = "bernoulli")

> summary(boost.caravan)

var rel.inf

PPERSAUT PPERSAUT 15.38257285

MKOOPKLA MKOOPKLA 10.46605387

MOPLHOOG MOPLHOOG 7.55417441

PBRAND PBRAND 4.91302326

MBERMIDD MBERMIDD 4.75162262

ABRAND ABRAND 4.10695497

MGODGE MGODGE 4.08460852

MINK3045 MINK3045 3.57754339

MOSTYPE MOSTYPE 2.95142262

MSKA MSKA 2.31330767

MSKC MSKC 2.26066183

MBERARBG MBERARBG 2.23298183

PBYSTAND PBYSTAND 2.12038372

PWAPART PWAPART 2.11822298

MAUT2 MAUT2 2.09432337

MSKB1 MSKB1 2.04625792

MAUT1 MAUT1 2.01682392

MGODOV MGODOV 1.81440468

MGODPR MGODPR 1.76389034

MBERHOOG MBERHOOG 1.62253130

MRELGE MRELGE 1.61409689

MINKGEM MINKGEM 1.54973694

MRELOV MRELOV 1.16048896

MGODRK MGODRK 1.00271859

MFWEKIND MFWEKIND 0.98670997

MHHUUR MHHUUR 0.98255028

MAUT0 MAUT0 0.95127063

MGEMLEEF MGEMLEEF 0.92105017

MINKM30 MINKM30 0.85528510

APERSAUT APERSAUT 0.83742532

MINK7512 MINK7512 0.81474065

MOPLMIDD MOPLMIDD 0.76091591

MGEMOMV MGEMOMV 0.72333895

MFGEKIND MFGEKIND 0.72106858

MOSHOOFD MOSHOOFD 0.69538457

PLEVEN PLEVEN 0.66896464

MINK123M MINK123M 0.58561784

PMOTSCO PMOTSCO 0.53489044

MBERBOER MBERBOER 0.45829619

MZFONDS MZFONDS 0.43597782

MSKD MSKD 0.42079495

MINK4575 MINK4575 0.37339174

MHKOOP MHKOOP 0.35897344

MSKB2 MSKB2 0.35869222

MZPART MZPART 0.34433413

MBERARBO MBERARBO 0.22410217

MBERZELF MBERZELF 0.18600145

MRELSA MRELSA 0.18042416

MAANTHUI MAANTHUI 0.05935146

ALEVEN ALEVEN 0.04163979

MFALLEEN MFALLEEN 0.00000000

MOPLLAAG MOPLLAAG 0.00000000

PWABEDR PWABEDR 0.00000000

PWALAND PWALAND 0.00000000

PBESAUT PBESAUT 0.00000000

PVRAAUT PVRAAUT 0.00000000

PAANHANG PAANHANG 0.00000000

PTRACTOR PTRACTOR 0.00000000

PWERKT PWERKT 0.00000000

PBROM PBROM 0.00000000

PPERSONG PPERSONG 0.00000000

PGEZONG PGEZONG 0.00000000

PWAOREG PWAOREG 0.00000000

PZEILPL PZEILPL 0.00000000

PPLEZIER PPLEZIER 0.00000000

PFIETS PFIETS 0.00000000

PINBOED PINBOED 0.00000000

AWAPART AWAPART 0.00000000

AWABEDR AWABEDR 0.00000000

AWALAND AWALAND 0.00000000

ABESAUT ABESAUT 0.00000000

AMOTSCO AMOTSCO 0.00000000

AVRAAUT AVRAAUT 0.00000000

AAANHANG AAANHANG 0.00000000

ATRACTOR ATRACTOR 0.00000000

AWERKT AWERKT 0.00000000

ABROM ABROM 0.00000000

APERSONG APERSONG 0.00000000

AGEZONG AGEZONG 0.00000000

AWAOREG AWAOREG 0.00000000

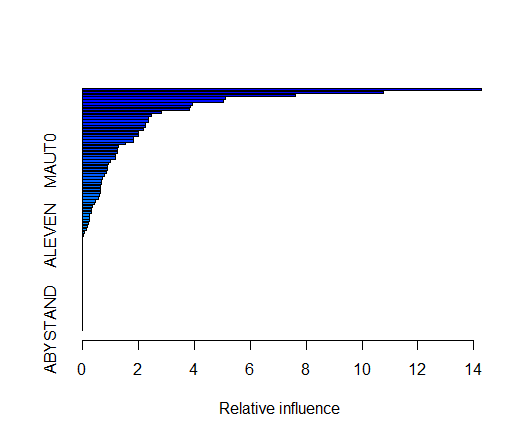
AZEILPL AZEILPL 0.00000000

APLEZIER APLEZIER 0.00000000

AFIETS AFIETS 0.00000000

AINBOED AINBOED 0.00000000

ABYSTAND ABYSTAND 0.00000000



PPERSAUT和MKOOPKLA是最重要的两个变量。

**11(c)**

> boost.prob = predict(boost.caravan, Caravan.test, n.trees = 1000, type = "response")

> boost.pred = ifelse(boost.prob > 0.2, 1, 0)

> table(Caravan.test$Purchase, boost.pred)

boost.pred

0 1

0 4425 108

1 258 31

> 31/(31+108)

[1] 0.2230216

在被预测为购买的用户中，有大约22.30%的用户确实是购买了。

> lm.caravan = glm(Purchase ~ ., data = Caravan.train, family = binomial)

> lm.prob = predict(lm.caravan, Caravan.test, type = "response")

> lm.pred = ifelse(lm.prob > 0.2, 1, 0)

> table(Caravan.test$Purchase, lm.pred)

lm.pred

0 1

0 4183 350

1 231 58

> 58/(350 + 58)

[1] 0.1421569

在被预测为购买的用户中，有大约14.22%的用户确实是购买了。

> knn.pred = knn(Caravan.train,Caravan.test,Caravan.train$Purchase,100)

> table(Caravan.test$Purchase,knn.pred)

knn.pred

0 1

0 4533 0

1 289 0

K取值为100时，预测没有人购买。

Boosting的预测结果比Logistic回归和KNN的预测结果好。

**12**

选择College数据集，选取一半数据作为训练集，剩余数据作为测试集。

> library(ISLR)

> College$Private = ifelse(College$Private == "Yes",1,0)

> set.seed(1)

> College\_test<-sample(1:nrow(College),nrow(College)/2)

> College\_train<-(-College\_test)

> College\_test<-College[College\_test,]

> College\_train<-College[College\_train,]

首先用Boosting方法。

> set.seed(100)

> boost.college = gbm(Private~.,data = College\_train,distribution = "bernoulli",n.trees = 1000)

> boost.prob\_college = predict(boost.college,newdata = College\_test,n.tree = 1000)

> boost.pred\_college = ifelse(boost.prob\_college > 0.5, 1, 0)

> table(College\_test$Private,boost.pred\_college)

boost.pred\_college

0 1

0 67 53

1 4 264

> mean(College\_test$Private == boost.pred\_college)

[1] 0.8530928

然后是Bagging方法。

> library(randomForest)

> set.seed(1000)

> bag.college = randomForest(as.factor(Private)~.,data = College\_train,mtry = 17)

> bag.pred\_college = predict(bag.college,newdata = College\_test)

> table(College\_test$Private,bag.pred\_college)

bag.pred\_college

0 1

0 101 19

1 13 255

> mean(College\_test$Private == bag.pred\_college)

[1] 0.9175258

接下来是Random Forests

> set.seed(10000)

> rf.college = randomForest(as.factor(Private)~.,data = College\_train,mtry = sqrt(17))

> rf.pred\_college = predict(rf.college,newdata = College\_test)

> table(College\_test$Private,rf.pred\_college)

rf.pred\_college

0 1

0 102 18

1 9 259

> mean(College\_test$Private == rf.pred\_college)

[1] 0.9304124

最后是Logistic回归

> log.college = glm(Private~.,data = College\_train,family = binomial)

> log.prob\_college = predict(log.college,College\_test,type = "response")

> log.pred\_college =ifelse(log.prob\_college > 0.5, 1, 0)

> table(College\_test$Private,log.pred\_college)

log.pred\_college

0 1

0 101 19

1 20 248

> mean(College\_test$Private == log.pred\_college)

[1] 0.8994845

准确率最高的是Random Forests，准确率为93.04%，其次是Bagging，准确率为91.75%，然后是Logistic回归，准确率为89.95%，最差的是Boosting，准确率为85.31%。