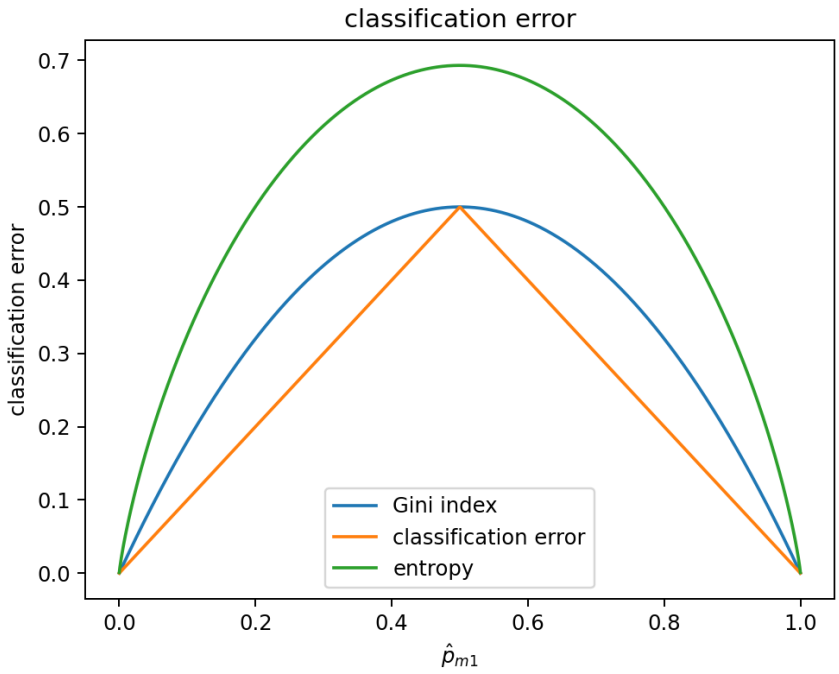
统计学习 作业4

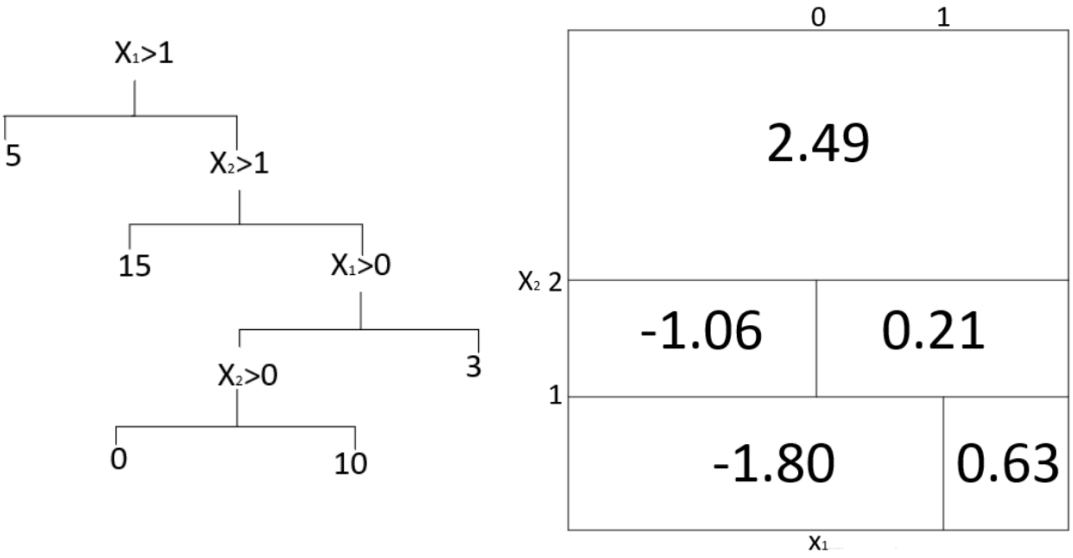
顾格非 3210103528

3

```
import numpy as np
import math
import matplotlib.pylab as plt
p = np.linspace(0.0001,0.9999,1000) # 0的时候entropy遇到log(0)
jini = np.array([2*x*(1-x) for x in p])
claerror = np.array([1- max(x,(1-x)) for x in p])
entropy = np.array([-x*math.log(x)-(1-x)*math.log(1-x) for x in p])
plt.xlabel("$\hat{p}_{m1}$")
plt.ylabel("classification error")
plt.title("classification error")
plt.plot(p,jini,label="Gini index")
plt.plot(p,claerror,label = "classification error")
plt.plot(p,entropy,label = "entropy")
plt.legend()
plt.show()
```

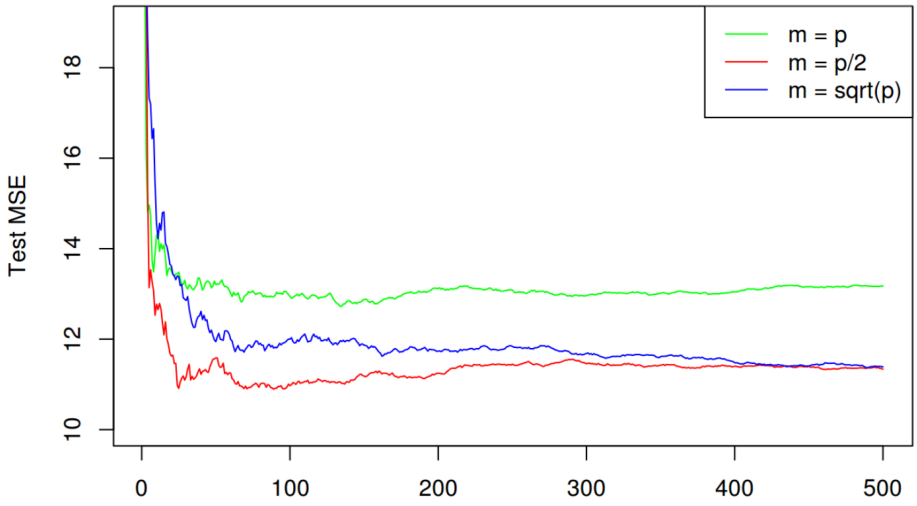


4



7

```
library(MASS)
library(randomForest)
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston) / 2)
Boston.train <- Boston[train, -14]
Boston.test <- Boston[-train, -14]
Y.train <- Boston[train, 14]
Y.test <- Boston[-train, 14]
rf.boston1 <- randomForest(Boston.train, y = Y.train, xtest = Boston.test, ytest =
Y.test, mtry = ncol(Boston) - 1, ntree = 500)
rf.boston2 <- randomForest(Boston.train, y = Y.train, xtest = Boston.test, ytest =
Y.test, mtry = (ncol(Boston) - 1) / 2, ntree = 500)
rf.boston3 <- randomForest(Boston.train, y = Y.train, xtest = Boston.test, ytest =
Y.test, mtry = sqrt(ncol(Boston) - 1), ntree = 500)
plot(1:500, rf.boston1$test$mse, col = "green", type = "l", xlab = "Number of Trees",
ylab = "Test MSE", ylim = c(10, 19))
lines(1:500, rf.boston2$test$mse, col = "red", type = "l")
lines(1:500, rf.boston3$test$mse, col = "blue", type = "l")
legend("topright", c("m = p", "m = p/2", "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时随机森林模型的测试均方误差最小，变量数目为p时随机森林模型的测试均方误差最大。

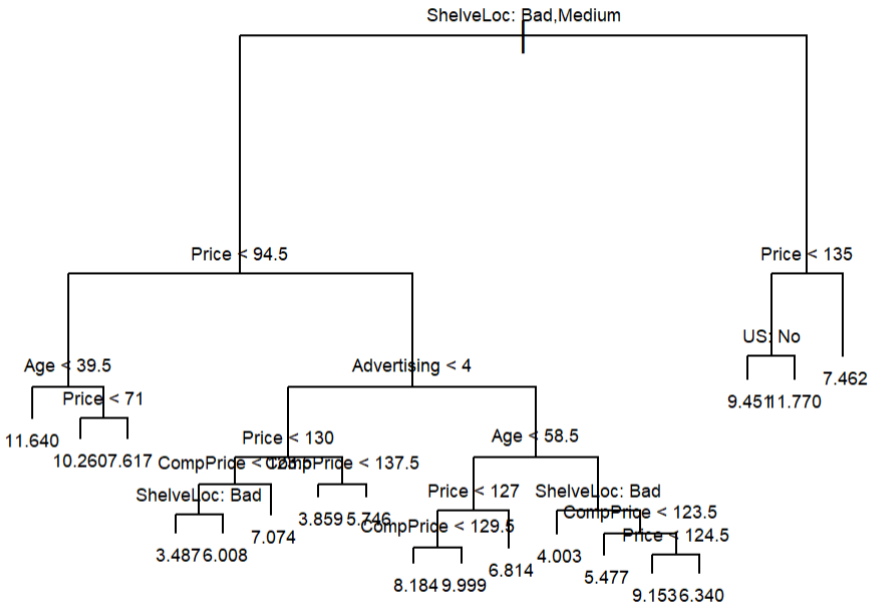
8

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

```
library(ISLR)
set.seed(1)
train <- sample(1:nrow(Carseats), nrow(Carseats) / 2)
Carseats.train <- Carseats[train, ]
Carseats.test <- Carseats[-train, ]
```

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

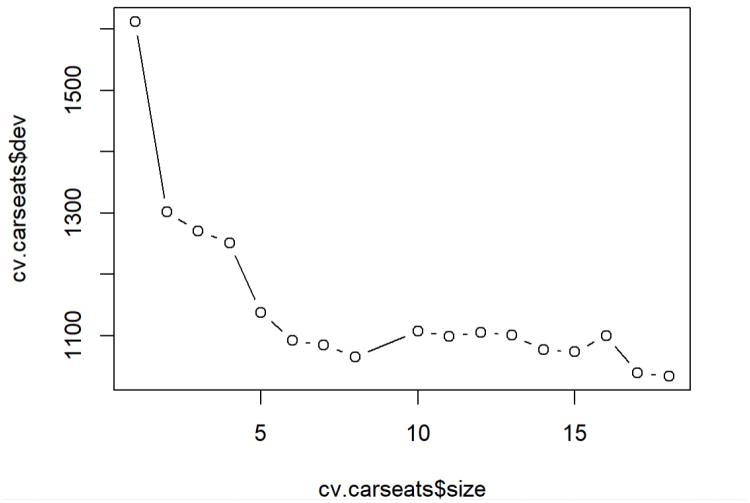
```
library(ISLR)
set.seed(1)
train <- sample(1:nrow(Carseats), nrow(Carseats) / 2)
Carseats.train <- Carseats[train, ]
Carseats.test <- Carseats[-train, ]
install.packages("tree")

library(tree)
tree.carseats <- tree(Sales ~ ., data = Carseats.train)
summary(tree.carseats)
plot(tree.carseats)
text(tree.carseats, pretty = 0,cex=0.5)
yhat <- predict(tree.carseats, newdata = Carseats.test)
mean((yhat - Carseats.test$Sales)^2)
```



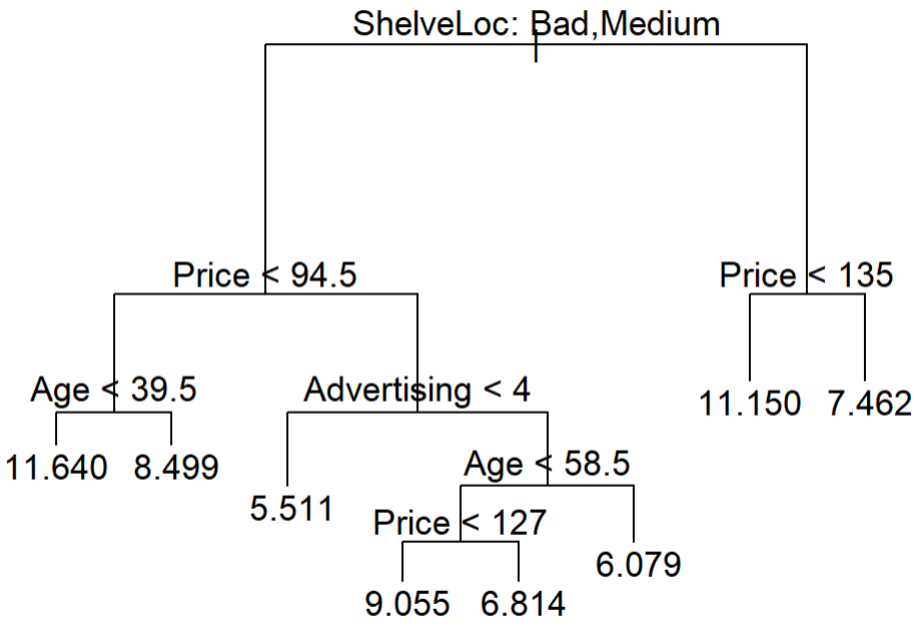

```
yhat <- predict(tree.carseats, newdata = Carseats.test)
mean((yhat - Carseats.test$Sales)^2)
```

得到MSE是4.922039。

```
cv.carseats <- cv.tree(tree.carseats)
plot(cv.carseats$size, cv.carseats$dev, type = "b")
```




```
prune.carseats <- prune.tree(tree.carseats, best = 8)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```



而此时的MSE是5.09085，可以看到并没有提升。

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

用把bagging方法得到的MSE是2.6，比原来有提升。

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

```
##               %IncMSE IncNodePurity
## CompPrice   14.4124562    133.731797
## Income       6.5147532     74.346961
## Advertising 15.7607104    117.822651
## Population   0.6031237     60.227867
## Price       57.8206926    514.802084
## ShelveLoc   43.0486065    319.117972
## Age         19.8789659    192.880596
## Education    2.9319161     39.490093
## Urban       -3.1300102      8.695529
## US           7.6298722     15.723975
```

输出的列名分别为 `%IncMSE` 和 `IncNodePurity`，表示对应的均方误差（MSE）增量和节点纯度增量。可以看出 Price 和 ShelveLoc 特征对于模型的贡献最大

e.

```
rf.carseats <- randomForest(Sales ~ ., data = Carseats.train, mtry = 3, ntree = 500,
importance = TRUE)
yhat.rf <- predict(rf.carseats, newdata = Carseats.test)
mean((yhat.rf - Carseats.test$Sales)^2)
importance(rf.carseats)
```

随机森林的test MSE是3.296

```
##               %IncMSE IncNodePurity
## CompPrice    7.5233429     127.36625
## Income       4.3612064     119.19152
## Advertising 12.5799388     138.13567
## Population  -0.2974474     100.28836
## Price       37.1612032     383.12126
## ShelveLoc   30.3751253     246.19930
## Age         16.0261885     197.44865
## Education    1.7855151      63.87939
## Urban       -1.3928225      16.01173
## US           5.6393475      32.85850
```

可以看出 Price 和 ShelveLoc 特征对于模型的贡献最大

f.

```
library(BART)
x=Carseats[,1:10]
y=Carseats[,"Sales"]
xtrain=x[train, ]
ytrain=y[train]
xtest=x[test, ]
ytest=y[test]
bart.fit=gbart(xtrain,ytrain,x.test=xtest)
yhat.bart=bart.fit$yhat.test.mean
mean((ytest-yhat.bart)^2)
```

发现BART是Bayesian Additive Regression Trees。BART的MSE做到了0.03794509，效果非常好。

9

```
attach(OJ)
set.seed(1)
train=sample(1:nrow(OJ),800)
oj.train=OJ[train, ]
oj.test=OJ[-train, ]
```

b.

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

```
##
## 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
```

可以看到有8个terminal nodes，training error rate是0.165

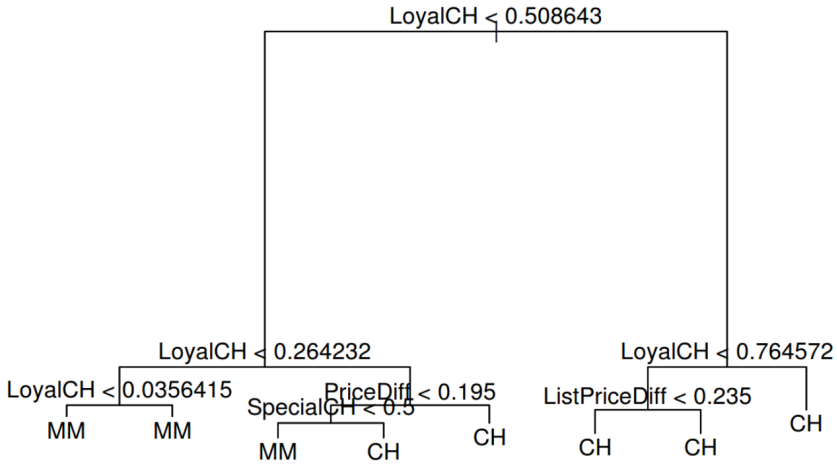
c.

```
tree.oj
```

```
## 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 ) *
```

`2) LoyalCH < 0.508643 350  409.30 MM ( 0.27143 0.72857 )`，比如看第二列，分类依据LoyalCH < 0.508643，有350个样本，偏差409.30，有27.143%的样本被分到了CH，有72.857%被分到了MM类

d.



可以看出LoyaCH是很重要的变量，顶上三个节点都是根据它划分的。

e.

```
tree.pred <- predict(tree.oj, OJ.test, type = "class")
table(tree.pred, OJ.test$Purchase)
```

```
##
## tree.pred  CH  MM
##        CH 147  49
##        MM  12  62
```

结果表明，模型正确地预测了147个 CH 类别样本和62个 MM 类别样本，但也有49个 CH 类别样本被错误地预测为 MM 类别，以及12个 MM 类别样本被错误地预测为 CH 类别。错误率：（12+49)/(12+49+147+62)=23.6%

f.

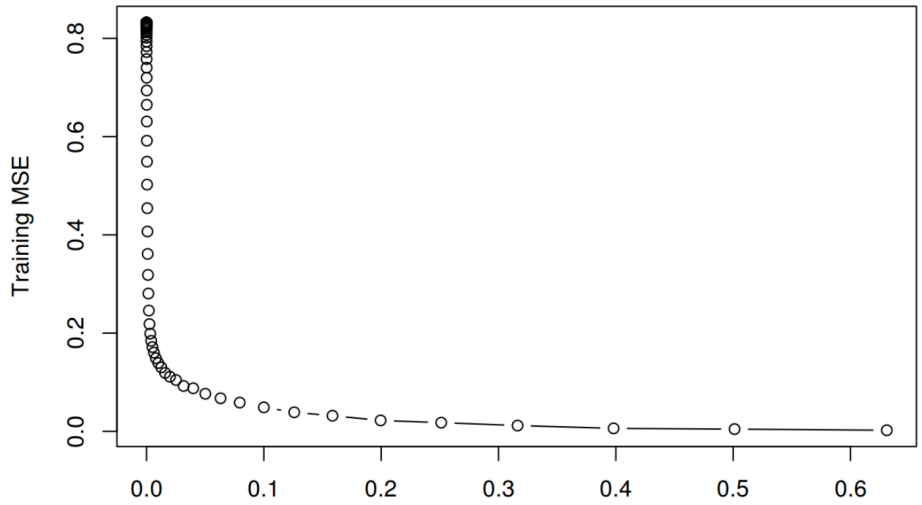
```
cv.oj <- cv.tree(tree.oj, FUN = prune.misclass)
cv.oj
```

```
$size
[1] 9 8 7 4 2 1

$dev
[1] 150 150 149 158 172 315

$k
[1]       -Inf   0.000000   3.000000   4.333333  10.500000 151.000000

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

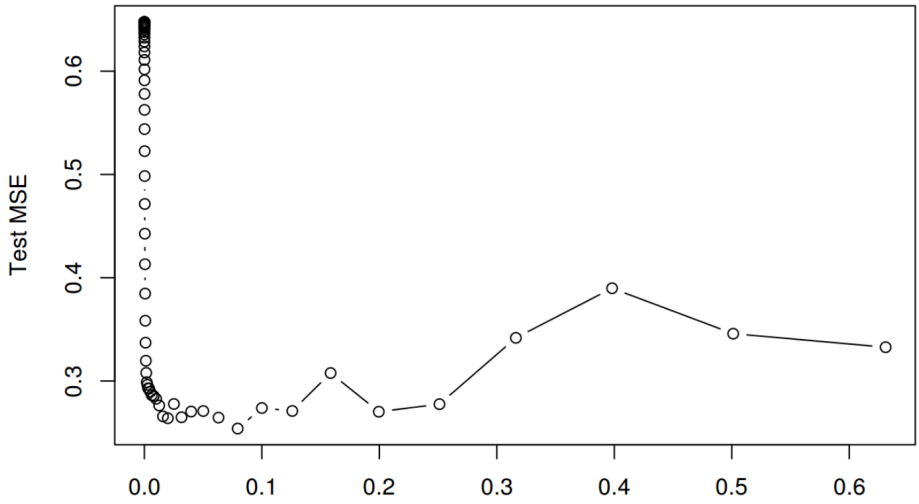
可以看到，size=7时的分类误差最小。

10

```
Hitters <- na.omit(Hitters)
Hitters$Salary <- log(Hitters$Salary)

train <- 1:200
Hitters.train <- Hitters[train, ]
Hitters.test <- Hitters[-train, ]

library(gbm)
set.seed(1)
pows <- seq(-10, -0.2, by = 0.1)
lambdas <- 10^pows
train.err <- rep(NA, length(lambdas))
for (i in 1:length(lambdas)) {
    boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian",
n.trees = 1000, shrinkage = lambdas[i])
    pred.train <- predict(boost.hitters, Hitters.train, n.trees = 1000)
    train.err[i] <- mean((pred.train - Hitters.train$Salary)^2)
}
plot(lambdas, train.err, type = "b", xlab = "Shrinkage values", ylab = "Training MSE")
```




```
set.seed(1)
test.err <- rep(NA, length(lambdas))
for (i in 1:length(lambdas)) {
    boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian",
n.trees = 1000, shrinkage = lambdas[i])
    yhat <- predict(boost.hitters, Hitters.test, n.trees = 1000)
    test.err[i] <- mean((yhat - Hitters.test$Salary)^2)
}
plot(lambdas, test.err, type = "b", xlab = "Shrinkage values", ylab = "Test MSE")
```




```
library(glmnet)
fit1 <- lm(Salary ~ ., data = Hitters.train)
pred1 <- predict(fit1, Hitters.test)
mean((pred1 - Hitters.test$Salary)^2)
```

```
0.4917959
```

```
x <- model.matrix(Salary ~ ., data = Hitters.train)
x.test <- model.matrix(Salary ~ ., data = Hitters.test)
y <- Hitters.train$Salary
fit2 <- glmnet(x, y, alpha = 0)
pred2 <- predict(fit2, s = 0.01, newx = x.test)
mean((pred2 - Hitters.test$Salary)^2)
```

```
0.4570283
```

可以看到，用boosting方法的test MSE更低一点。

```
boost.hitters <- gbm(Salary ~ ., data = Hitters.train, distribution = "gaussian",
n.trees = 1000, shrinkage = lambdas[which.min(test.err)])
summary(boost.hitters)
```

```
##                var    rel.inf
## CAtBat       CAtBat 23.6278225
## Walks         Walks  7.4858538
## PutOuts     PutOuts  7.4438741
## CWalks       CWalks  7.3437919
## CRBI           CRBI  6.7581127
## Years         Years  6.0435181
## CRuns         CRuns  5.8389531
## Assists     Assists  5.2910978
## CHmRun       CHmRun  4.9321654
## Hits           Hits  4.4338073
## CHits         CHits  4.4008947
## RBI             RBI  4.2852503
## HmRun         HmRun  3.2708823
## AtBat         AtBat  3.1569920
## Errors       Errors  2.6144780
```

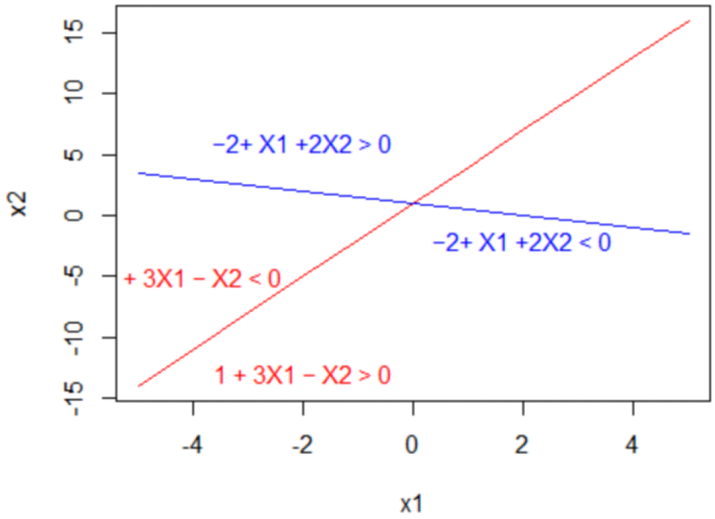
```
## Runs           Runs  1.8095817
## Division   Division  0.6302155
## NewLeague NewLeague  0.5079414
## League       League  0.1247675
```

可以看到CAtBat是最重要的predictor。

```
set.seed(1)
bag.hitters <- randomForest(Salary ~ ., data = Hitters.train, mtry = 19, ntree = 500)
yhat.bag <- predict(bag.hitters, newdata = Hitters.test)
mean((yhat.bag - Hitters.test$Salary)^2)
```

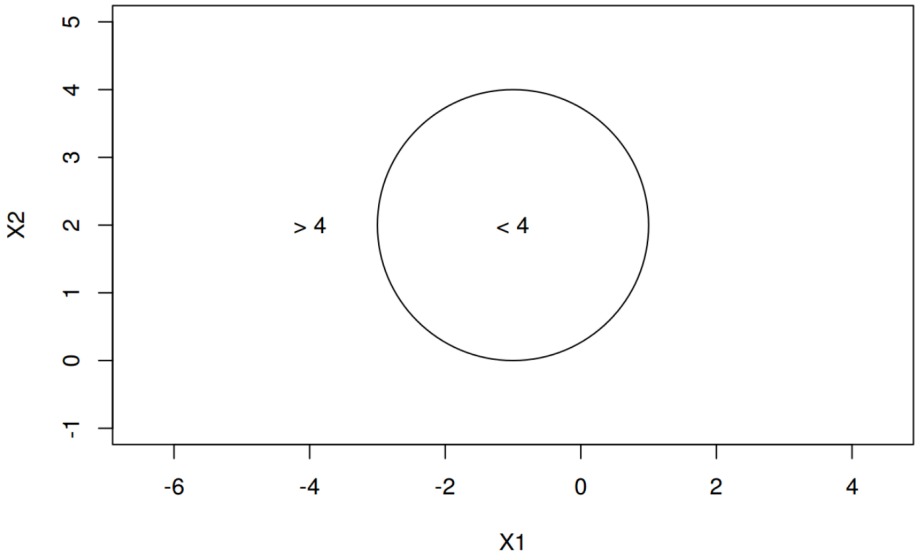
可以得到bagging的test MSE是0.2313593.

1



2

曲线如图，在圆内<4,圆外>4

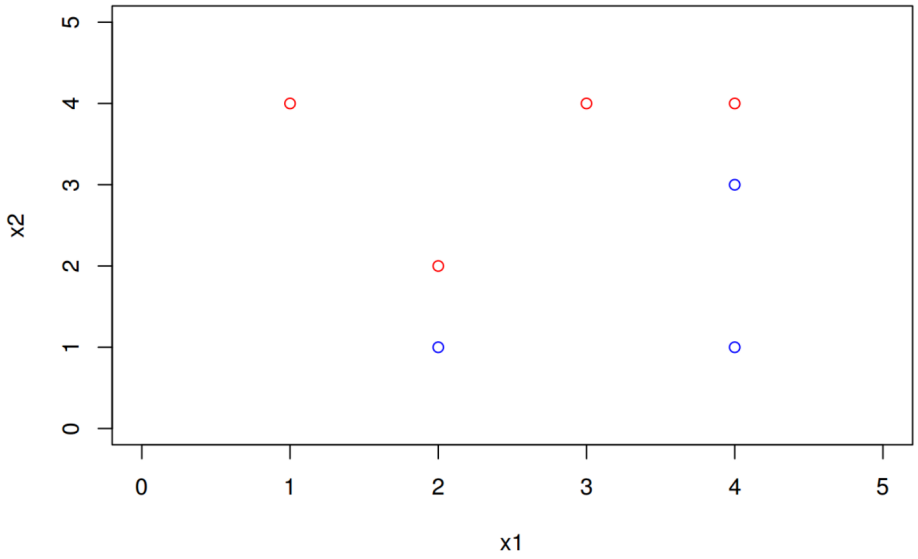


3. (0,0) 蓝色，(-1,1)红色，(2,2)蓝色，(3,8)蓝色
4. 决策边界是$X_1^2 + 2X_1 + X_2^2 - 4X_1 + 1 = 0$.

所以对X_1, X_2不是线性的，对X_1, X_1^2, X_2, X_2^2是线性的。

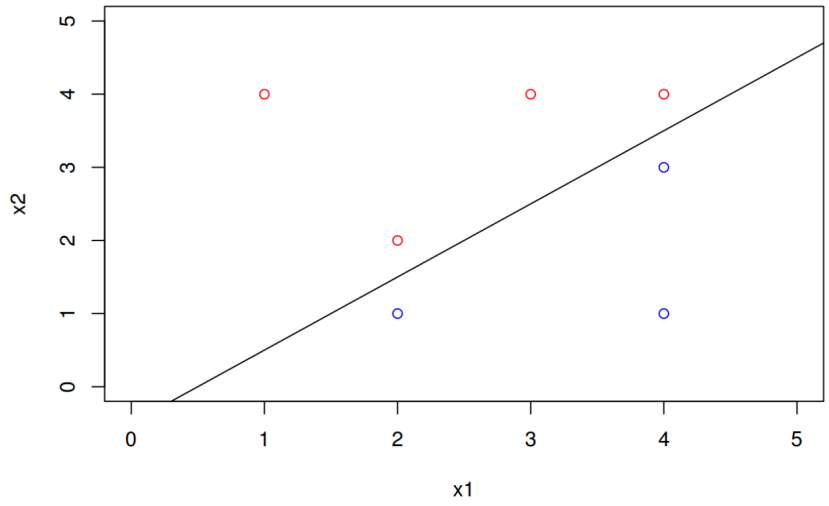
3

```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
```



b.

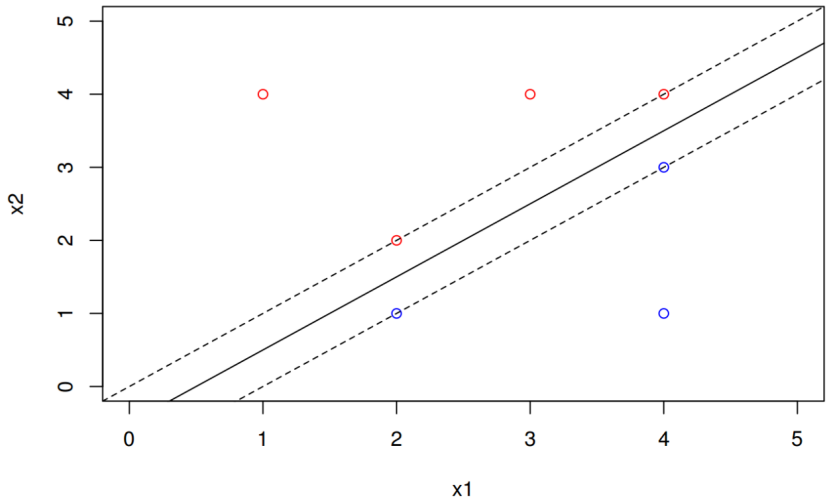
$X1 - X2 - 0.5 = 0$



c.

如果$X1 - X2 - 0.5 > 0$，则归为蓝色，若$X1 - X2 - 0.5 < 0$，则归为红色

d.

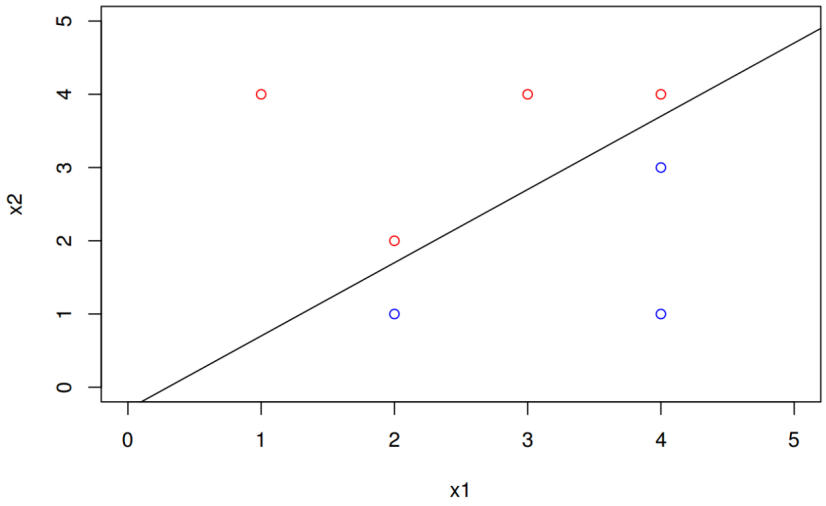


e. 支持向量有(2,1), (2,2), (4,3) (4,4)

f. 因为第7个点不是支持向量，离超平面有一定距离，所以即使他轻微移动，离超平面的距离没有显著变化，所以我们还是采用原来的超平面。

g.

比如$X_1 - X_2 - 0.2 = 0$就是一个不是最优的超平面。



h. 在右下角，比如(5,1)放一个红点，就不能分离了。

8

准备数据集

```
set.seed(1)
train <- sample(nrow(OJ), 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
```

用SVM拟合：

```
svm.linear <- svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)
summary(svm.linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  432
##
##  ( 215 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

可以看出支持向量有432个，其中CH有215个，MM有217个。

误差：

```
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
##     train.pred
##       CH  MM
##   CH 439  55
##   MM  78 228
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  31  80
```

train误差率：(78 + 55) / (439 + 228 + 78 + 55)=0.16625

test误差率：(31 + 18) / (141 + 80 + 31 + 18)=0.1814815

d.

```
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges =
list(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
```

```
## - best performance: 0.1625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17125 0.05172376
## 2   0.01778279 0.16500 0.05197489
## 3   0.03162278 0.16625 0.04604120
## 4   0.05623413 0.16500 0.04594683
## 5   0.10000000 0.16250 0.04787136
## 6   0.17782794 0.16250 0.04249183
## 7   0.31622777 0.16875 0.04379958
## 8   0.56234133 0.16625 0.03998698
## 9   1.00000000 0.16500 0.03670453
## 10  1.77827941 0.16625 0.03682259
## 11  3.16227766 0.16500 0.03717451
## 12  5.62341325 0.16500 0.03525699
## 13 10.00000000 0.16750 0.03917553
```

optimal cost 是 0.1

```
svm.linear <- svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost =
tune.out$best.parameter$cost)
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 438  56
##   MM  71 235
##     test.pred
##       CH  MM
##   CH 140  19
##   MM  32  79
```

得到的train误差是0.15875，test误差是0.1889

可以看到用了cost=0.1后，train误差有了减少。

f.

用上面同样的方法可以得到支持向量有379个，188是CH，191是MM。train误差是14.5%，test误差是17.1%

在找出最优的cost =1，此时train误差为0.145，test误差是0.170

g.

```
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", data = OJ.train, degree = 2)
summary(svm.poly)
```

输出的结果就不贴了，可以看出支持向量有454个，其中CH有224个，MM有230个。train误差0.1725，test误差0.1889

然后用上面同样的方法看出cost=10是最优的，在此基础上得到train 误差0.145，test误差0.1852,可以看到有较大提升。

h

综上，可以看出f题中用的radial kernel SVM效果最好。