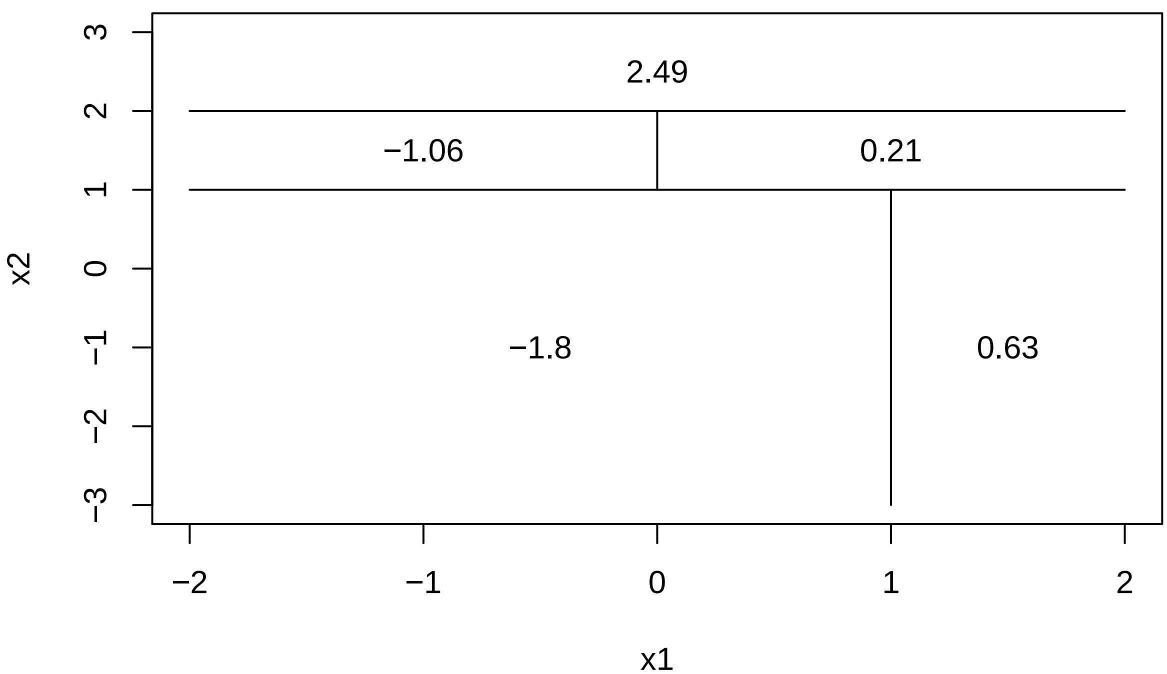
Assignment7 Chapter 8,9

Problem 1–Exercise 8-4

(a) if x1>1, y=5, else if x2>1, y=15, else if x1<0, y=3, else if x2>0, y=0

(b)

```
plot(NA, NA, type="n", xlim = c(-2,2), ylim = c(-3,3), xlab = "x1", ylab = "x2")
lines(x=c(-2,2), y=c(1,1))
lines(x = c(1, 1), y = c(-3, 1))
text(x = (-2 + 1)/2, y = -1, labels = c(-1.8))
text(x = 1.5, y = -1, labels = c(0.63))
lines(x = c(-2, 2), y = c(2, 2))
text(x = 0, y = 2.5, labels = c(2.49))
lines(x = c(0, 0), y = c(1, 2))
text(x = -1, y = 1.5, labels = c(-1.06))
text(x = 1, y = 1.5, labels = c(0.21))
```



Problem 2–Exercise 8-8

(a)

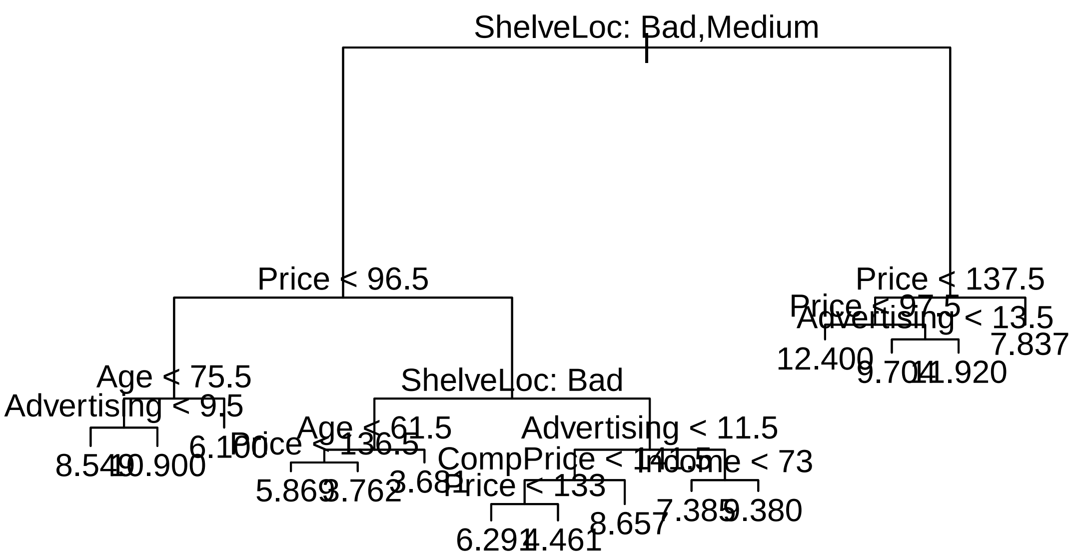
```
library(ISLR)
data("Carseats")
train=sample(1:nrow(Carseats), nrow(Carseats)/2)
test.data=Carseats[-train,]
train.data=Carseats[train,]
```

(b)

```r
library(tree)
tree.car=tree(Sales~. ,data=train.data)
summary(tree.car)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = train.data)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Advertising" "CompPrice"
## [6] "Income"
## Number of terminal nodes:  15
## Residual mean deviance:  2.281 = 421.9 / 185
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -4.461  -1.023   0.116   0.000   1.042   3.234
```

```r
plot(tree.car)
text(tree.car, pretty = 0)
```



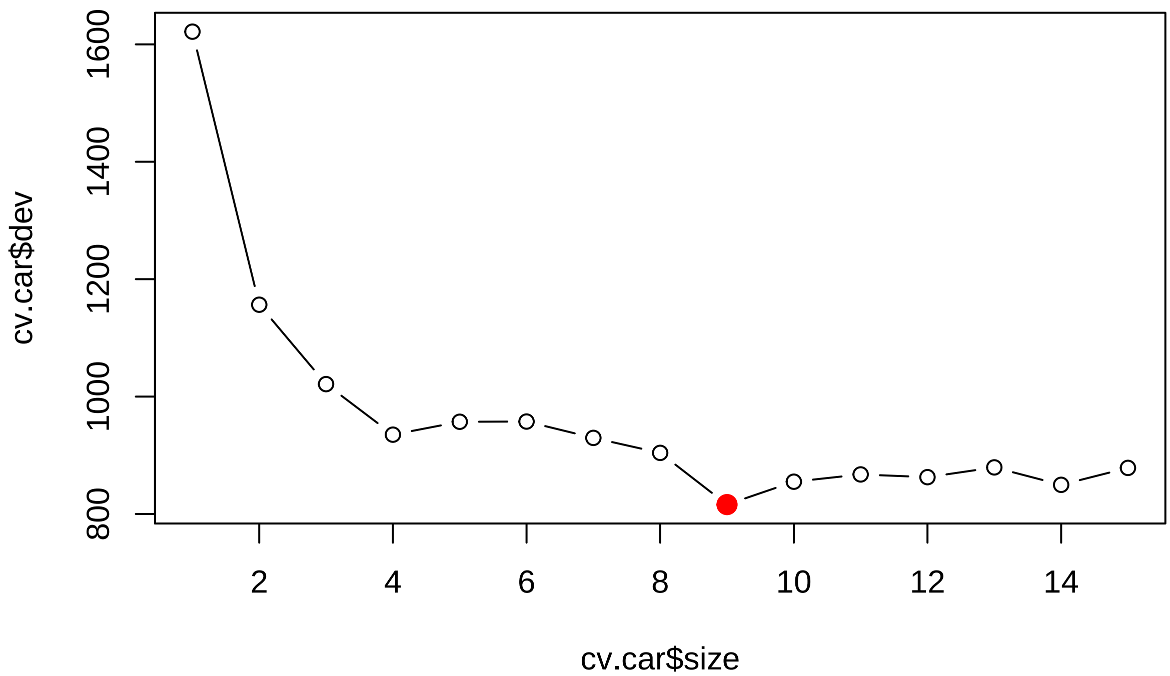

```r
y.test=predict(tree.car, test.data)
mean((y.test-test.data$Sales)^2)
```

```
## [1] 4.766264
```

Test MSE is 4.92

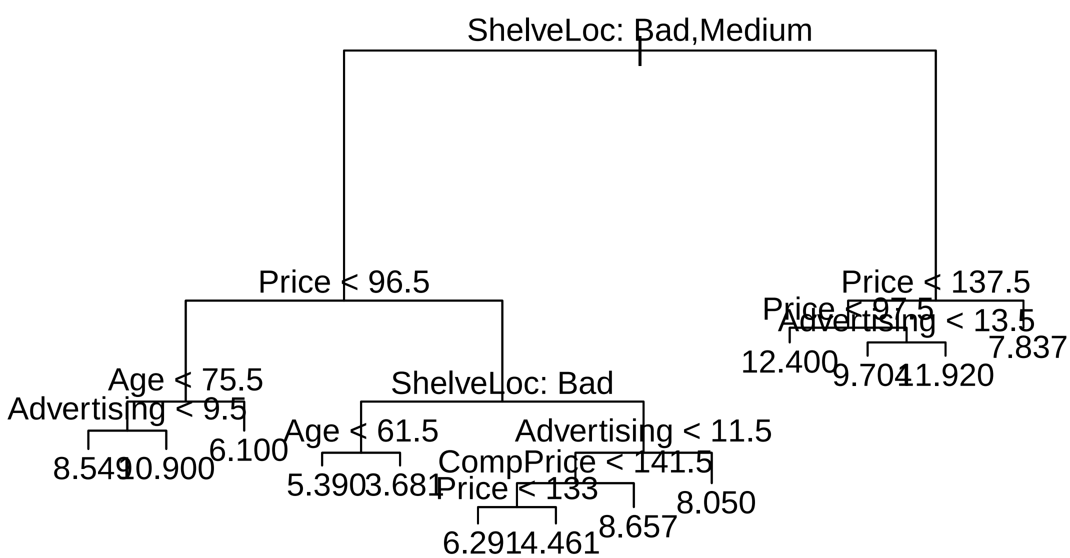
(c)

```r
cv.car=cv.tree(tree.car)
plot(cv.car$size, cv.car$dev, type = "b")
tree.min=which.min(cv.car$dev)
points(cv.car$size[tree.min], cv.car$dev[tree.min], col="red", cex=2, pch=20)
```



the tree size is 13

```r
prune.car=prune.tree(tree.car, best = 13)
plot(prune.car)
text(prune.car, pretty=0)
```




```r
prune.carpred=predict(prune.car, test.data)
mean((prune.carpred-test.data$Sales)^2)
```

```
## [1] 4.84172
```

the test MSE increases to 5

(d)

```r
library(randomForest)
```

```
## randomForest 4.6-14

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

```r
bag.car=randomForest(Sales~., data=train.data, mtry=10, importance = TRUE)
bag.carpred=predict(bag.car, test.data)
mean((bag.carpred-test.data$Sales)^2)
```

```
## [1] 2.758083
```

the test MSE is 3.1

```r
importance(bag.car)
```

```
##                %IncMSE IncNodePurity
## CompPrice   15.81982171    102.308575
## Income       6.67669047     79.729921
## Advertising 16.70663568    104.284485
## Population  -2.31107168     55.123982
## Price       47.49726297    439.766487
## ShelveLoc   63.90907134    566.852572
## Age         19.47426106    169.340052
## Education    0.07144119     32.462408
## Urban        0.69894057      8.377415
## US           2.83611418      5.550616
```

price and shelveloc are the two most important variables

(e)

```r
random.car=randomForest(Sales~., data=train.data, mtry=5, importance = TRUE)
random.carpred=predict(random.car, test.data)
mean((random.carpred-test.data$Sales)^2)
```

```
## [1] 2.943767
```

the test MSE is 3.25

```r
importance(random.car)
```

```
##               %IncMSE IncNodePurity
## CompPrice   11.6705602     120.02852
## Income       3.4699962     101.73429
## Advertising 13.4950174     118.82860
## Population  -0.1515029      78.32390
## Price       38.8484912     405.15725
## ShelveLoc   50.0518969     479.39237
## Age         17.7807411     193.85016
## Education   -2.0076830      45.65537
## Urban       -1.2364473       7.08890
## US           3.7472499      10.99367
```

price and shelveloc are still the two most important variables

Problem 3–Exercise 8-10

(a)

```
Hitters= na.omit(Hitters)
Hitters$Salary= log(Hitters$Salary)
```

(b)

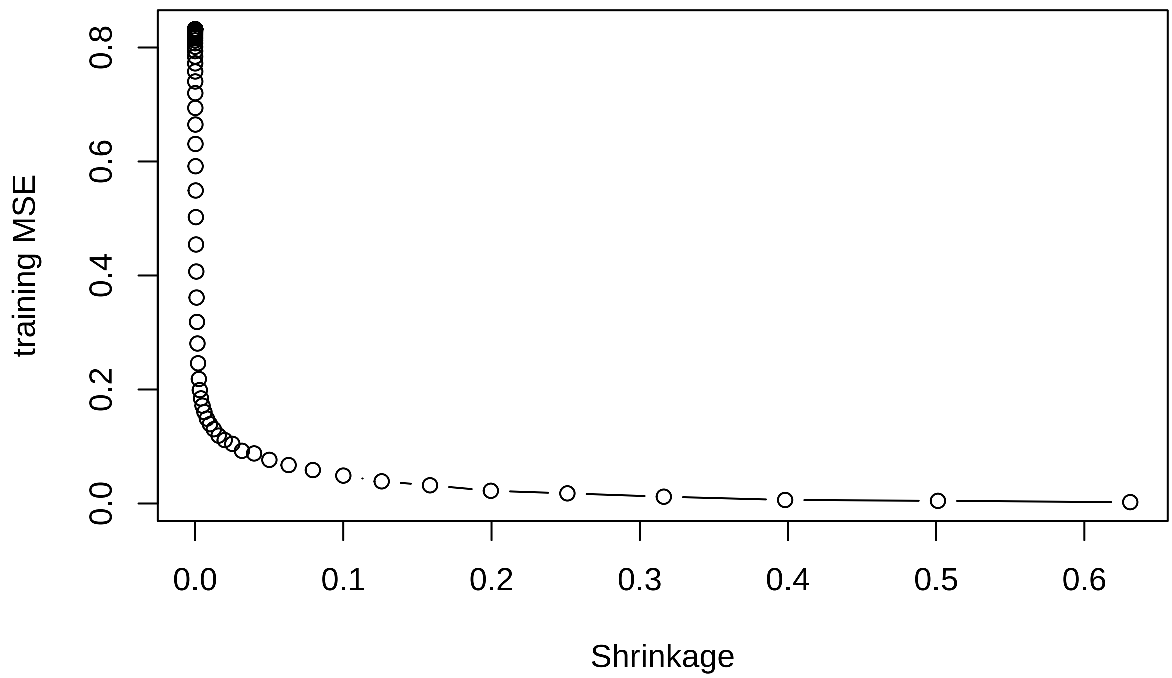
```
train=1:200
hitters.train=Hitters[train,]
hitters.test=Hitters[-train,]
```

(c)

```
library(gbm)
```

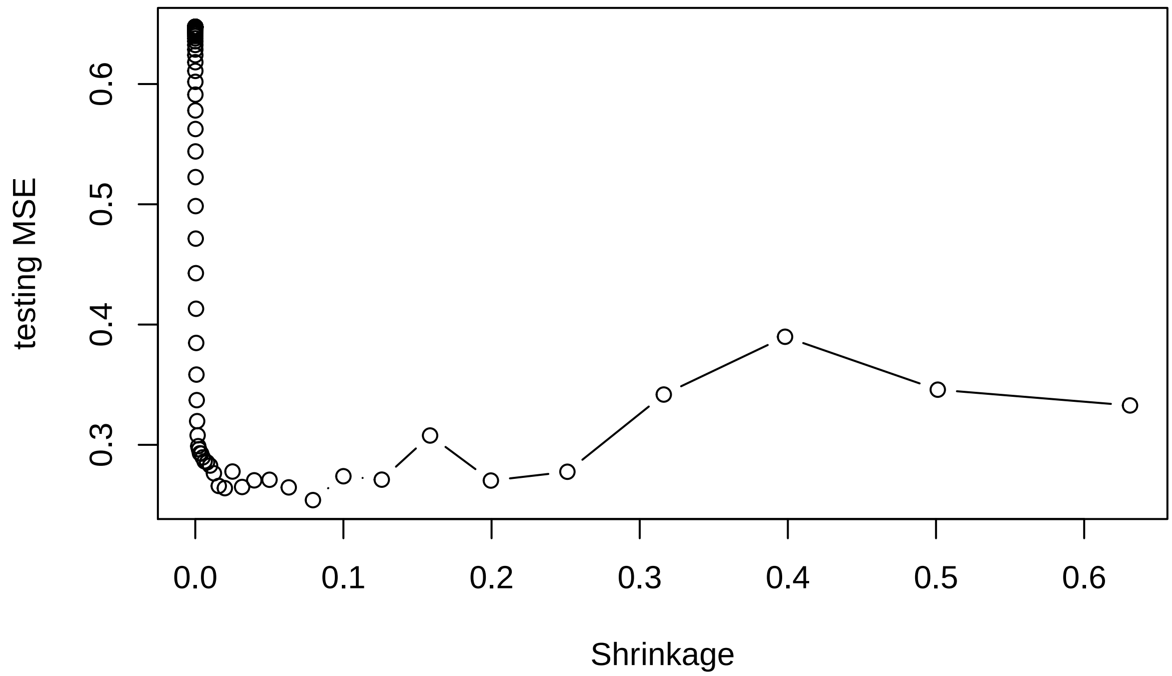
```
## Loaded gbm 2.1.5
```

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



(d)

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




```
min(test.err)
```

```
## [1] 0.2540265
```

```
penalty[which.min(test.err)]
```

```
## [1] 0.07943282
```

Minimum test MSE is 0.25, shrinkage value is 0.079

(e)

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach

## Loaded glmnet 2.0-18
```

```
lr=lm(Salary~., data = hitters.train)
pred.lr=predict(lr, hitters.test)
mean((pred.lr-hitters.test$Salary)^2)
```

```
## [1] 0.4917959
```

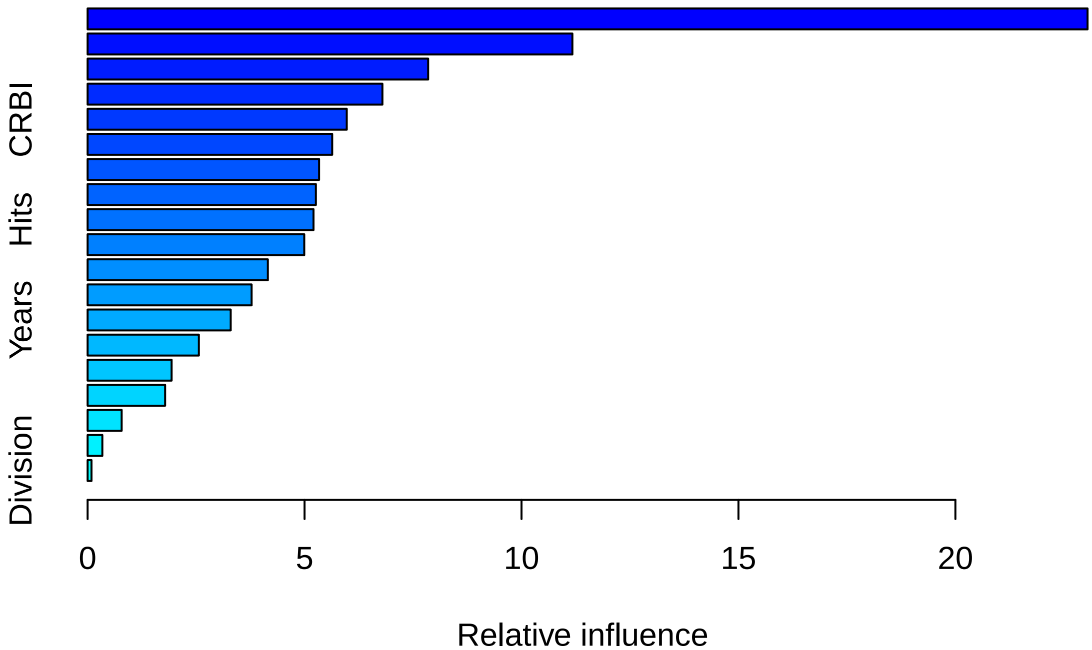
```
x=model.matrix(Salary~., data = hitters.train)
x.test=model.matrix(Salary~., data=hitters.test)
y=hitters.train$Salary
ridge= glmnet(x,y,alpha = 0)
pred.rid=predict(ridge, x.test)
mean((pred.rid-hitters.test$Salary)^2)
```

```
## [1] 0.5145349
```

both of the methods are higher than the boosting method.

(e)

```
boost.min=gbm(Salary~., data = hitters.train, distribution = "gaussian", n.trees = 1000, shrinkage = x[w
summary(boost.min)
```




```
##                 var     rel.inf
## CAtBat       CAtBat 23.04578739
## PutOuts     PutOuts 11.17279753
```

```
## Walks         Walks  7.84853274
## Assists     Assists  6.79471425
## CRBI           CRBI  5.97259164
## CWalks       CWalks  5.63737221
## RBI             RBI  5.33460070
## Runs           Runs  5.26075800
## Hits           Hits  5.20648613
## CHmRun       CHmRun  4.99293491
## AtBat         AtBat  4.15403411
## HmRun         HmRun  3.77979314
## Years         Years  3.29803057
## CRuns         CRuns  2.56421970
## CHits         CHits  1.93572414
## Errors       Errors  1.78691790
## NewLeague NewLeague  0.78480354
## League       League  0.33993881
## Division   Division  0.08996259
```

the most importnat variable is CAtBat

(f)

```
bag.hitters=randomForest(Salary~., data = hitters.train, mtry=19, ntree=500)
bag.hitterspred=predict(bag.hitters, hitters.test)
mean((bag.hitterspred-hitters.test$Salary)^2)
```

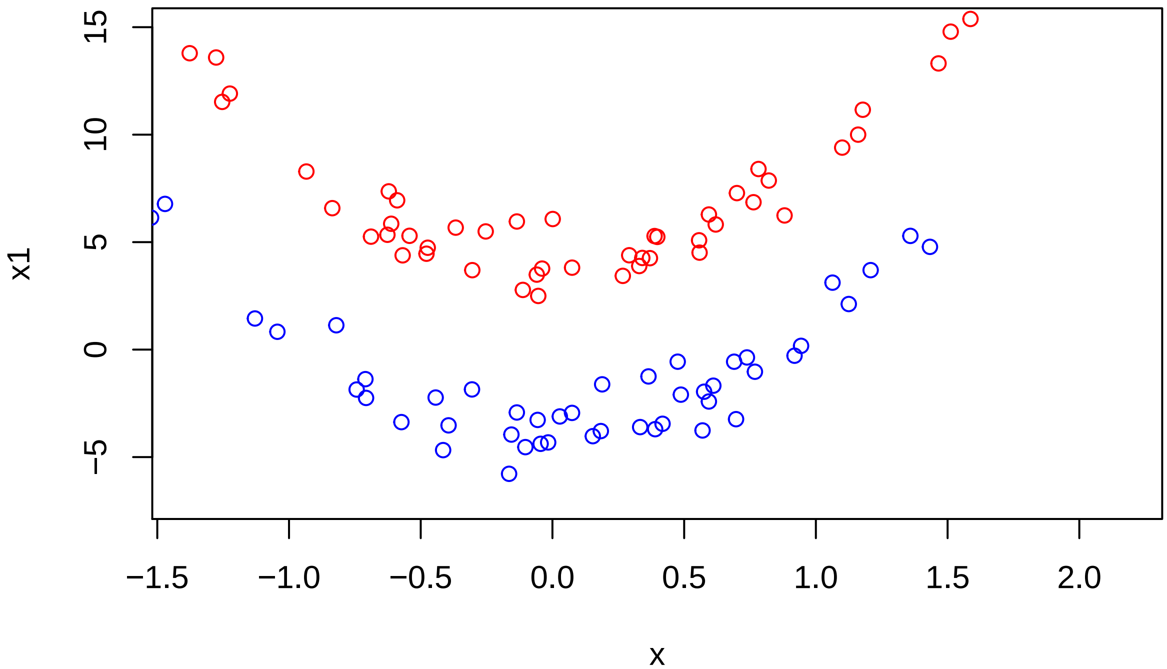
```
## [1] 0.2304652
```

slightly lower than the boosting

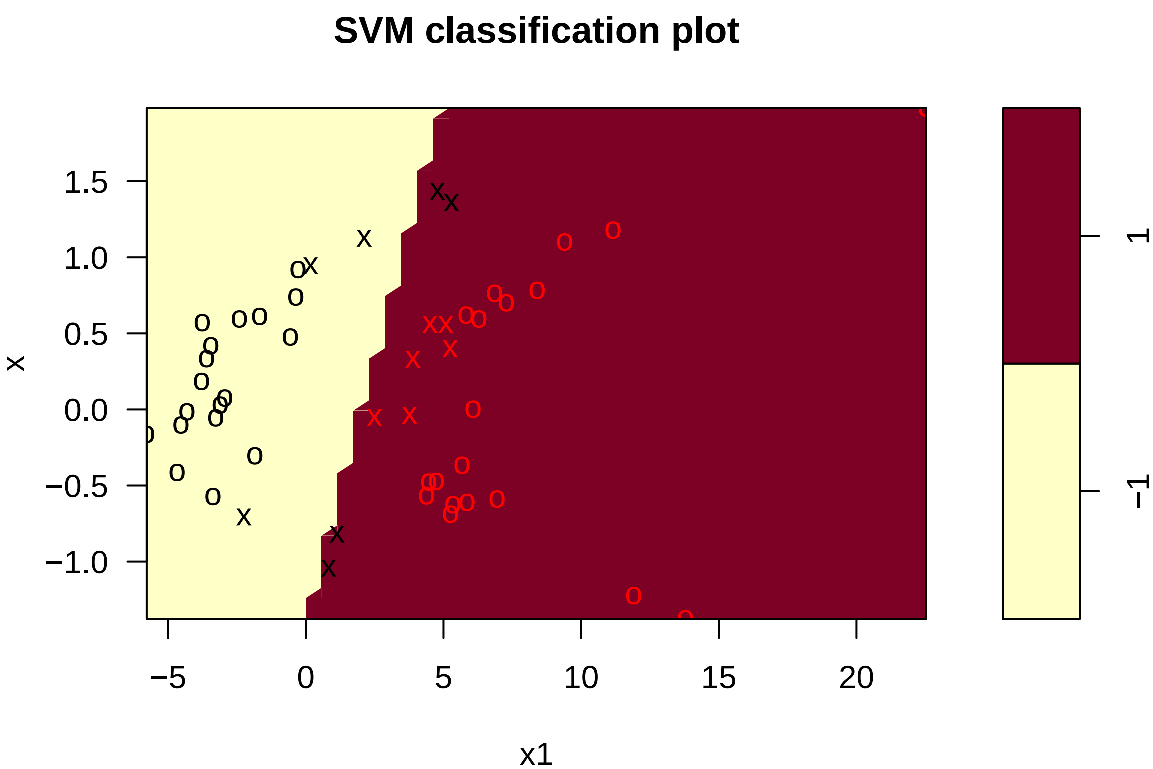
Problem 4–Exercise 9-4

(a)

```
library(e1071)
set.seed(1)
x=rnorm(100)
x1= 5*x^2 + rnorm(100)
randomsample=sample(100,50)
x1[randomsample]=x1[randomsample]+4
x1[-randomsample]=x1[-randomsample]-4
plot(x[randomsample], x1[randomsample], col="red", xlab = "x", ylab = "x1", ylim = c(-7,15))
points(x[-randomsample], x1[-randomsample], col="blue")
```




```
y= rep(-1,100)
y[randomsample]=1
data = data.frame(x=x, x1=x1, y = as.factor(y))
train=sample(100,50)
data.train=data[train,]
data.test=data[-train,]
svm.linear=svm(y~., data = data.train, kernel="linear", cost=1)
plot(svm.linear, data.train)
```

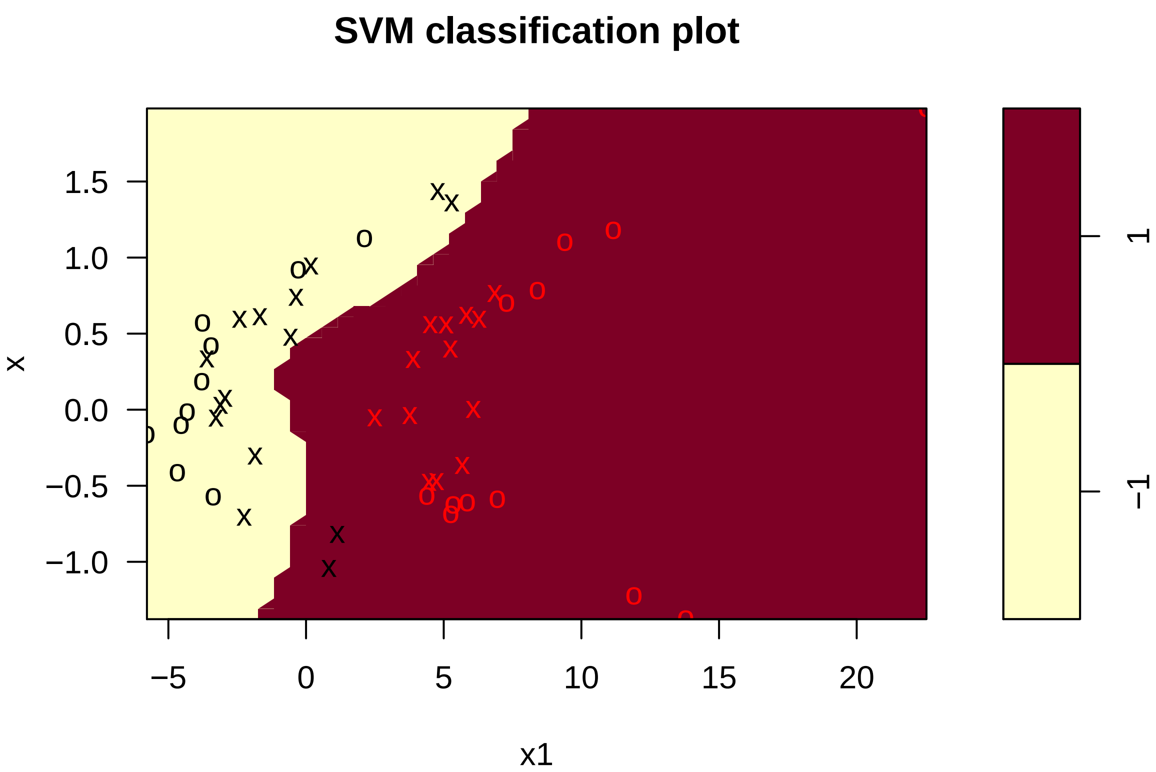



```r
table(predict=predict(svm.linear, data.train), truth=data.train$y)
```

```
##         truth
## predict -1  1
##      -1 21  0
##      1   4 25
```

the support vector classifier make 5 training errors. then lets try svm

```r
svm.poly=svm(y~., data = data.train, kernel="polynomial", cost=1)
plot(svm.poly, data.train)
```

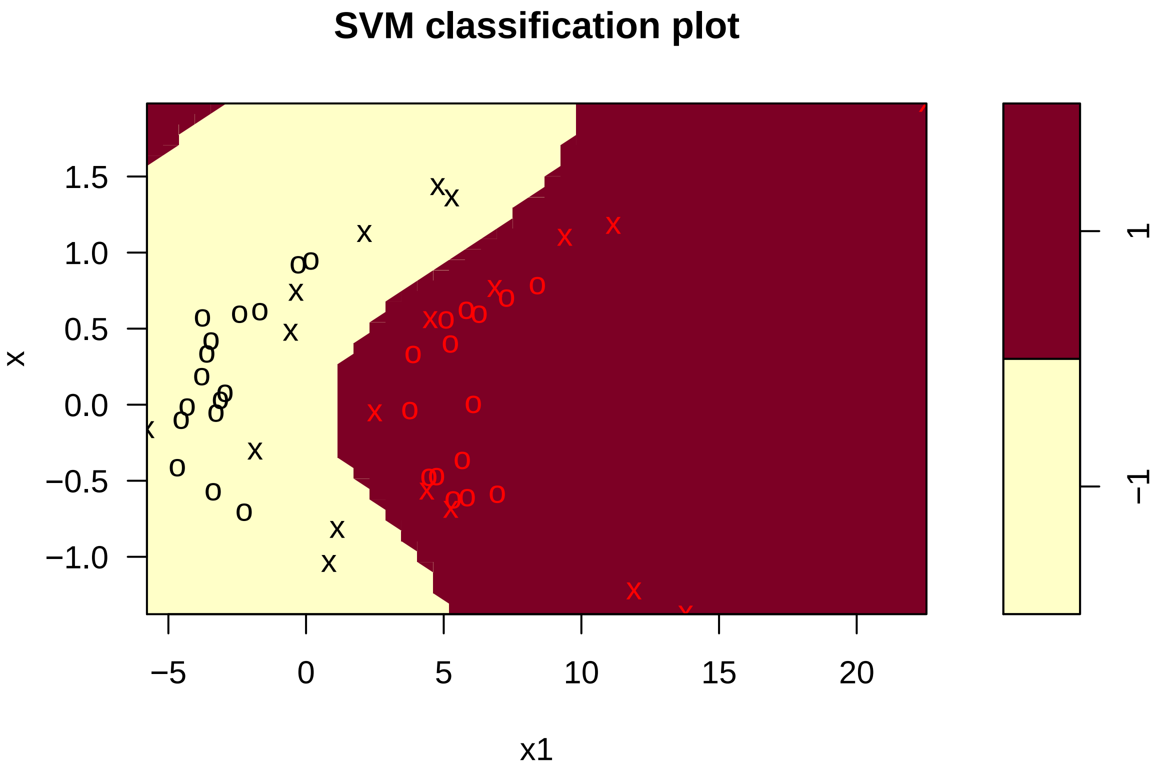



```r
table(predict=predict(svm.poly, data.train), truth=data.train$y)
```

```
##          truth
## predict -1  1
##      -1 23  0
##      1   2 25
```

the support vector machine poly 2 make 11 training errors. radial

```r
svm.rad=svm(y~., data = data.train, kernel="radial", gamma= 1, cost=1)
plot(svm.rad, data.train)
```

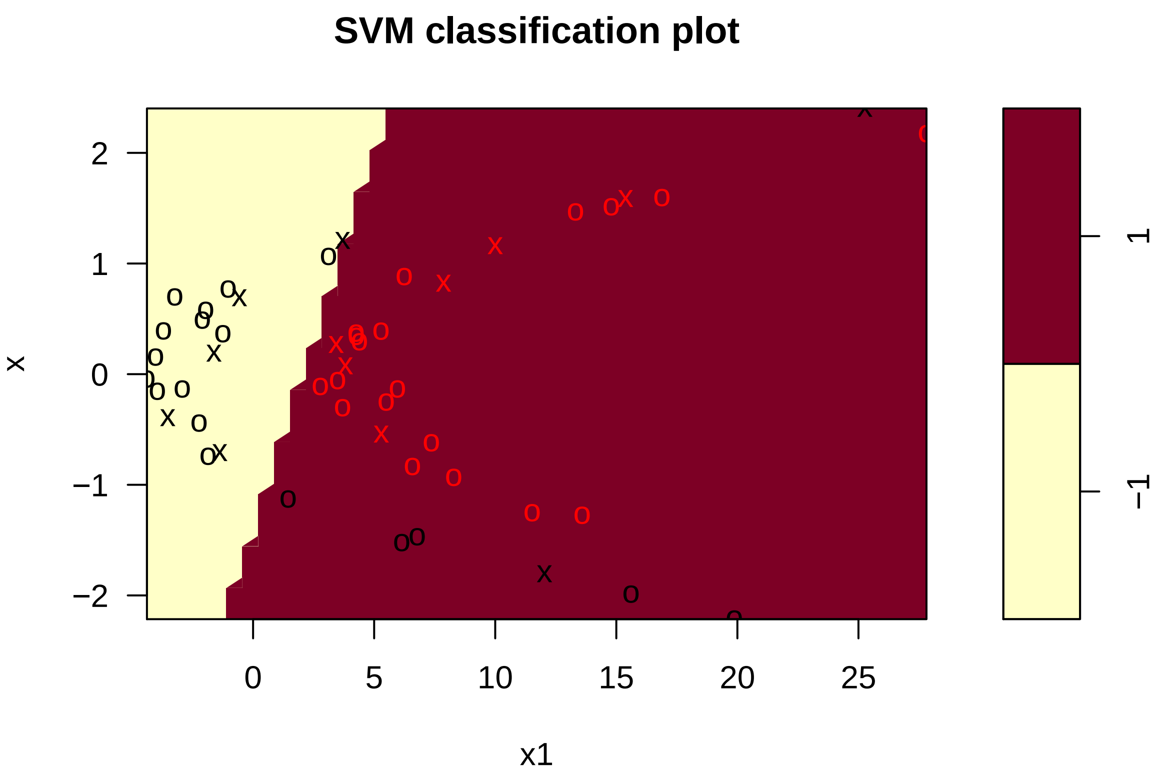



```r
table(predict=predict(svm.rad, data.train), truth=data.train$y)
```

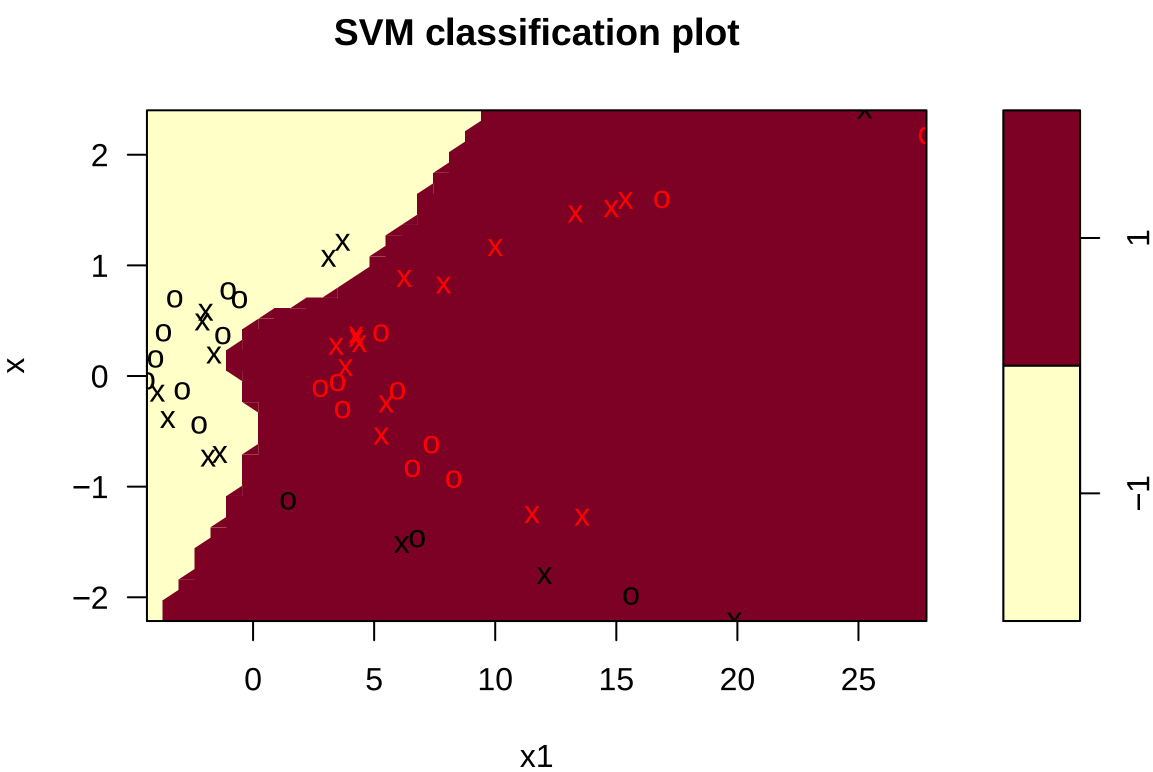
```
##         truth
## predict -1  1
##      -1 25  0
##      1   0 25
```

the support vector machine rad make 1 training errors. let try test error

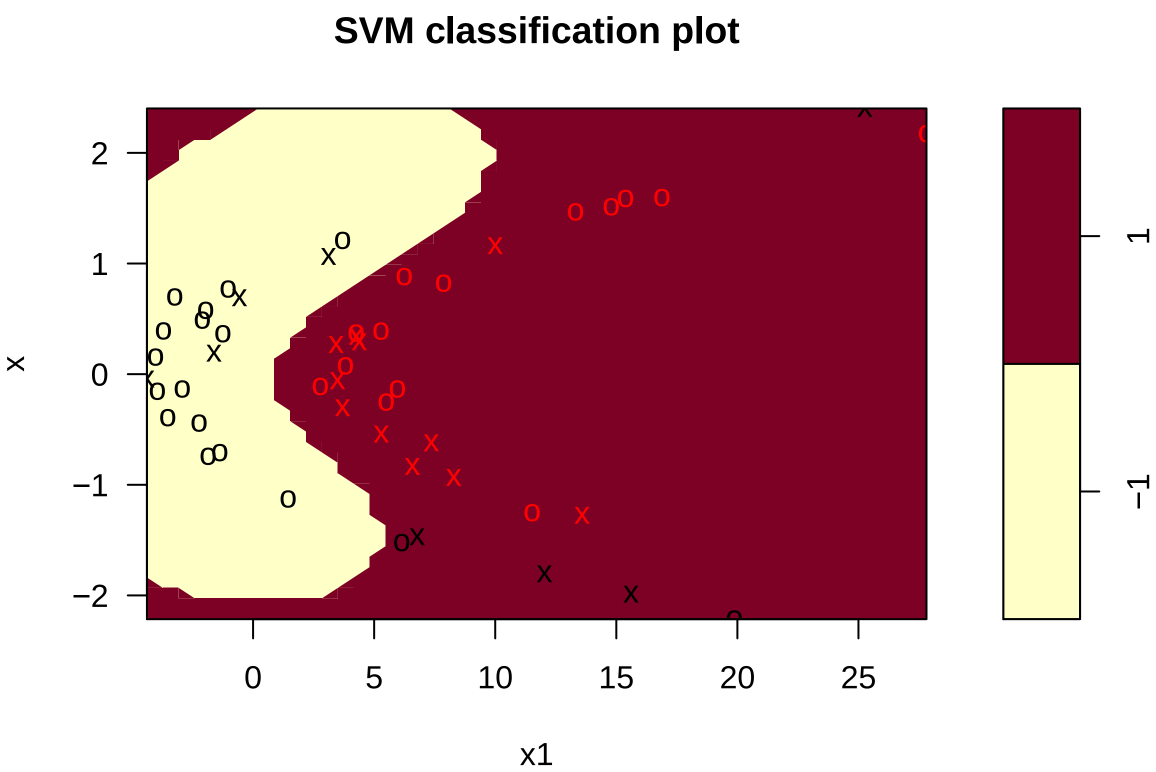
```r
plot(svm.linear, data.test, main="linear")
```




```r
plot(svm.poly, data.test, main="poly")
```




```r
plot(svm.rad, data.test, main="rad")
```




```r
table(lineartestpredict = predict(svm.linear, data.test), truth=data.test$y)
```

```
##                  truth
## lineartestpredict -1  1
##                -1 17  0
##                1   8 25
```

```r
table(polytestpredict=predict(svm.poly, data.test), truth=data.test$y)
```

```
##                truth
## polytestpredict -1  1
##              -1 18  0
##              1   7 25
```

```r
table(radtesttpredict=predict(svm.rad, data.test), truth=data.test$y)
```

```
##                truth
## radtesttpredict -1  1
##              -1 19  0
##              1   6 25
```

like training data error, the radial classifier is still the best model for the test data, only making 3 testing error.

Problem 5–Exercise 9-7

(a)

```
library(ISLR)
dummy.mile=ifelse(Auto$mpg>median(Auto$mpg), 1, 0)
Auto$mpgdummy=as.factor(dummy.mile)
```

(b)

```
set.seed(1)
tune.svc=tune(svm, mpgdummy~., data = Auto, kernel="linear", ranges = list(cost= c(0.01, 0.1, 1, 5, 10,
summary(tune.svc)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01025641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.07653846 0.03617137
## 2 1e-01 0.04596154 0.03378238
## 3 1e+00 0.01025641 0.01792836
## 4 5e+00 0.02051282 0.02648194
## 5 1e+01 0.02051282 0.02648194
## 6 1e+02 0.03076923 0.03151981
## 7 1e+03 0.03076923 0.03151981
```

cost 1 and 5 perform best.

(c)

```
set.seed(1)
tune.poly=tune(svm, mpgdummy~., data = Auto, kernel="polynomial", ranges = list(cost= c(0.01, 0.1, 1, 5
summary(tune.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##  1000      2
##
## - best performance: 0.2454487
##
## - Detailed performance results:
```

```
##     cost degree     error dispersion
## 1  1e-02      2 0.5511538 0.04366593
## 2  1e-01      2 0.5511538 0.04366593
## 3  1e+00      2 0.5511538 0.04366593
## 4  5e+00      2 0.5511538 0.04366593
## 5  1e+01      2 0.5130128 0.08963366
## 6  1e+02      2 0.3013462 0.09961961
## 7  1e+03      2 0.2454487 0.11551451
## 8  1e-02      3 0.5511538 0.04366593
## 9  1e-01      3 0.5511538 0.04366593
## 10 1e+00      3 0.5511538 0.04366593
## 11 5e+00      3 0.5511538 0.04366593
## 12 1e+01      3 0.5511538 0.04366593
## 13 1e+02      3 0.3446154 0.09821588
## 14 1e+03      3 0.2528846 0.09383590
## 15 1e-02      4 0.5511538 0.04366593
## 16 1e-01      4 0.5511538 0.04366593
## 17 1e+00      4 0.5511538 0.04366593
## 18 5e+00      4 0.5511538 0.04366593
## 19 1e+01      4 0.5511538 0.04366593
## 20 1e+02      4 0.5511538 0.04366593
## 21 1e+03      4 0.5435897 0.05056569
## 22 1e-02      5 0.5511538 0.04366593
## 23 1e-01      5 0.5511538 0.04366593
## 24 1e+00      5 0.5511538 0.04366593
## 25 5e+00      5 0.5511538 0.04366593
## 26 1e+01      5 0.5511538 0.04366593
## 27 1e+02      5 0.5511538 0.04366593
## 28 1e+03      5 0.5511538 0.04366593
```

cost 1000 with degree= 2 perform best.

```r
set.seed(1)
tune.rad=tune(svm, mpgdummy~., data = Auto, kernel="radial", ranges = list(cost= c(0.01, 0.1, 1, 5, 10,
summary(tune.rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100  0.01
##
## - best performance: 0.01282051
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-02 1e-02 0.55115385 0.04366593
## 2  1e-01 1e-02 0.08929487 0.04382379
## 3  1e+00 1e-02 0.07403846 0.03522110
## 4  5e+00 1e-02 0.04852564 0.03303346
```

```
## 5  1e+01 1e-02 0.02557692 0.02093679
## 6  1e+02 1e-02 0.01282051 0.01813094
## 7  1e+03 1e-02 0.02820513 0.02549818
## 8  1e-02 1e-01 0.21711538 0.09865227
## 9  1e-01 1e-01 0.07903846 0.03874545
## 10 1e+00 1e-01 0.05371795 0.03525162
## 11 5e+00 1e-01 0.02820513 0.03299190
## 12 1e+01 1e-01 0.03076923 0.03375798
## 13 1e+02 1e-01 0.03583333 0.02759051
## 14 1e+03 1e-01 0.03583333 0.02759051
## 15 1e-02 1e+00 0.55115385 0.04366593
## 16 1e-01 1e+00 0.55115385 0.04366593
## 17 1e+00 1e+00 0.06384615 0.04375618
## 18 5e+00 1e+00 0.05884615 0.04020934
## 19 1e+01 1e+00 0.05884615 0.04020934
## 20 1e+02 1e+00 0.05884615 0.04020934
## 21 1e+03 1e+00 0.05884615 0.04020934
## 22 1e-02 5e+00 0.55115385 0.04366593
## 23 1e-01 5e+00 0.55115385 0.04366593
## 24 1e+00 5e+00 0.49493590 0.04724924
## 25 5e+00 5e+00 0.48217949 0.05470903
## 26 1e+01 5e+00 0.48217949 0.05470903
## 27 1e+02 5e+00 0.48217949 0.05470903
## 28 1e+03 5e+00 0.48217949 0.05470903
## 29 1e-02 1e+01 0.55115385 0.04366593
## 30 1e-01 1e+01 0.55115385 0.04366593
## 31 1e+00 1e+01 0.51794872 0.05063697
## 32 5e+00 1e+01 0.51794872 0.04917316
## 33 1e+01 1e+01 0.51794872 0.04917316
## 34 1e+02 1e+01 0.51794872 0.04917316
## 35 1e+03 1e+01 0.51794872 0.04917316
## 36 1e-02 1e+02 0.55115385 0.04366593
## 37 1e-01 1e+02 0.55115385 0.04366593
## 38 1e+00 1e+02 0.55115385 0.04366593
## 39 5e+00 1e+02 0.55115385 0.04366593
## 40 1e+01 1e+02 0.55115385 0.04366593
## 41 1e+02 1e+02 0.55115385 0.04366593
## 42 1e+03 1e+02 0.55115385 0.04366593
## 43 1e-02 1e+03 0.55115385 0.04366593
## 44 1e-01 1e+03 0.55115385 0.04366593
## 45 1e+00 1e+03 0.55115385 0.04366593
## 46 5e+00 1e+03 0.55115385 0.04366593
## 47 1e+01 1e+03 0.55115385 0.04366593
## 48 1e+02 1e+03 0.55115385 0.04366593
## 49 1e+03 1e+03 0.55115385 0.04366593
```

cost 100 with gamma= 0.01 perform best.

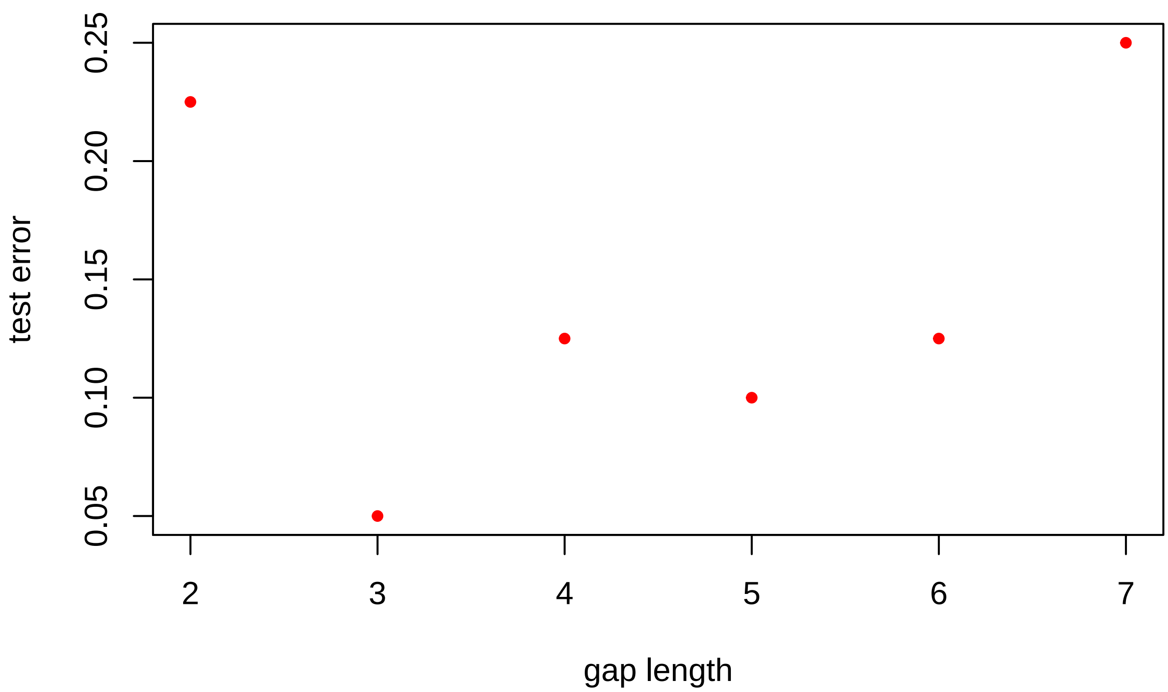
Problem 6

(a)

```r
library(kernlab)
library(datasets)
set.seed(3)
data(reuters)
y <- rlabels # article topic
x <- reuters # article
gap=rep(NA,7)
for (i in 2:7) {
ker = read.csv(paste('~/Desktop/len',i,'lam0.1.csv',sep = ""))
ker = as.kernelMatrix(as.matrix(ker))
svgap = ksvm(x=ker[,-1],y=rlabels,cross=5)
gap[i]=cross(svgap)
}
gap
```

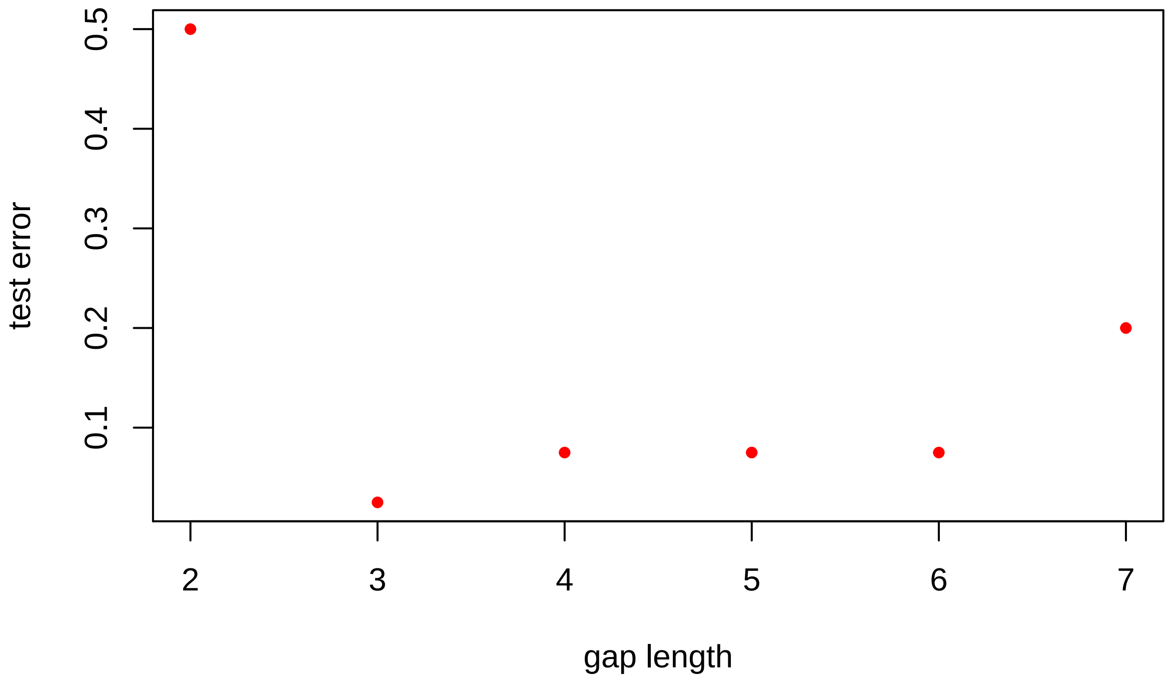
```
## [1]    NA 0.225 0.050 0.125 0.100 0.125 0.250
```

```r
plot(2:7,gap[2:7], xlab="gap length", ylab="test error", pch=20, col="red")
```



For gappy kernel, length 3 got the best test error.

```r
data(reuters)
y <- rlabels # article topic
x <- reuters # article
set.seed(1)
spec=rep(NA,7)
for (i in 2:7) {
sk <- stringdot(type="spectrum", length=i, normalized=TRUE) #spectrum kernel count the word appear in t
svp <- ksvm(x,y,kernel=sk,scale=c(),cross=5) #run an SVM with the kernel sk
spec[i]=cross(svp) #test error estimate
}
plot(2:7,spec[2:7], xlab="gap length", ylab="test error", pch=20, col="red")
```



for the spectrum kernel, length 3 got the best test error