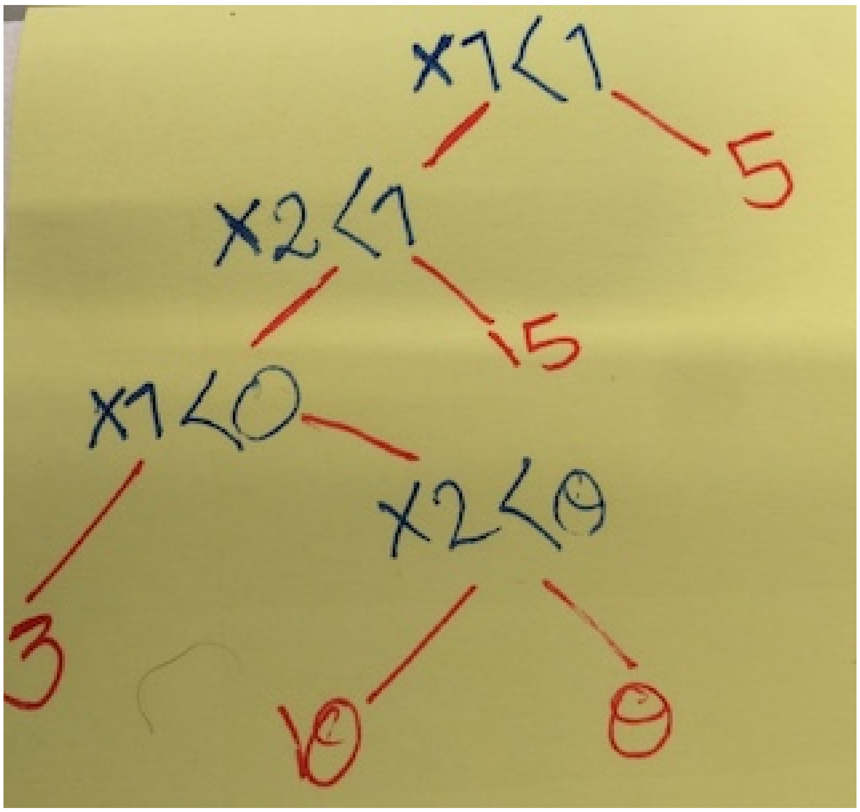
HW7

Ramtin Boustani - SUID# 05999261

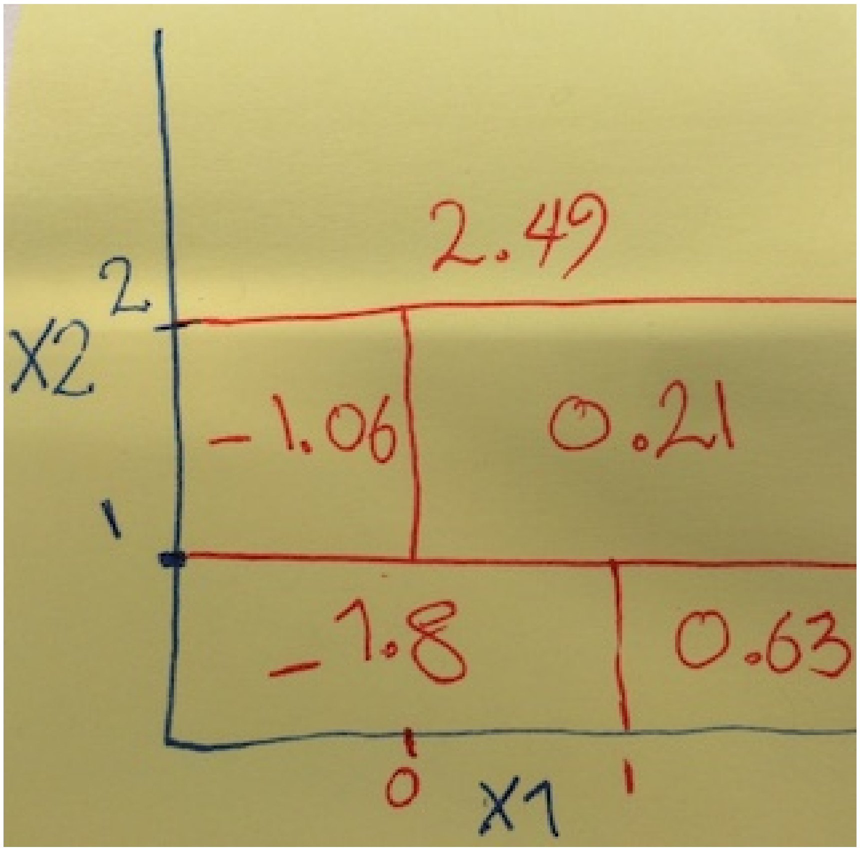
Problem 1

Chapter 8, Exercise 4

(a)



(b)



Problem 2

Chapter 8, Exercise 8

```r
require(tree)
```

```
## Loading required package: tree
```

```r
require(ISLR)
```

```
## Loading required package: ISLR
```

```r
attach(Carseats)
```

(a)

```r
set.seed(1)
train = sample(nrow(Carseats), nrow(Carseats)/2)
```

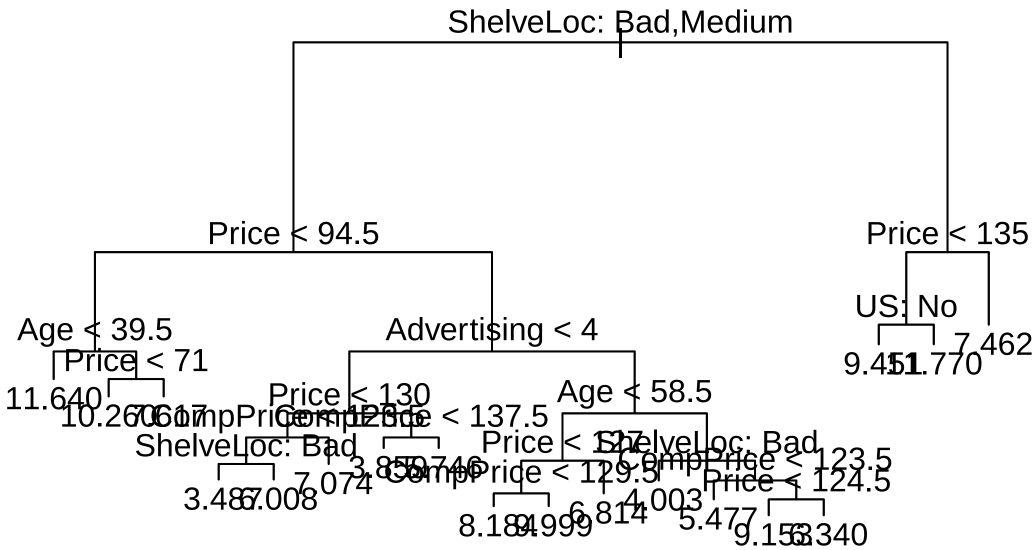
(b)

```r
tree.carseats = tree(Sales ~., data = Carseats[train,])
summary(tree.carseats)
```

```
##
## 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
```

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



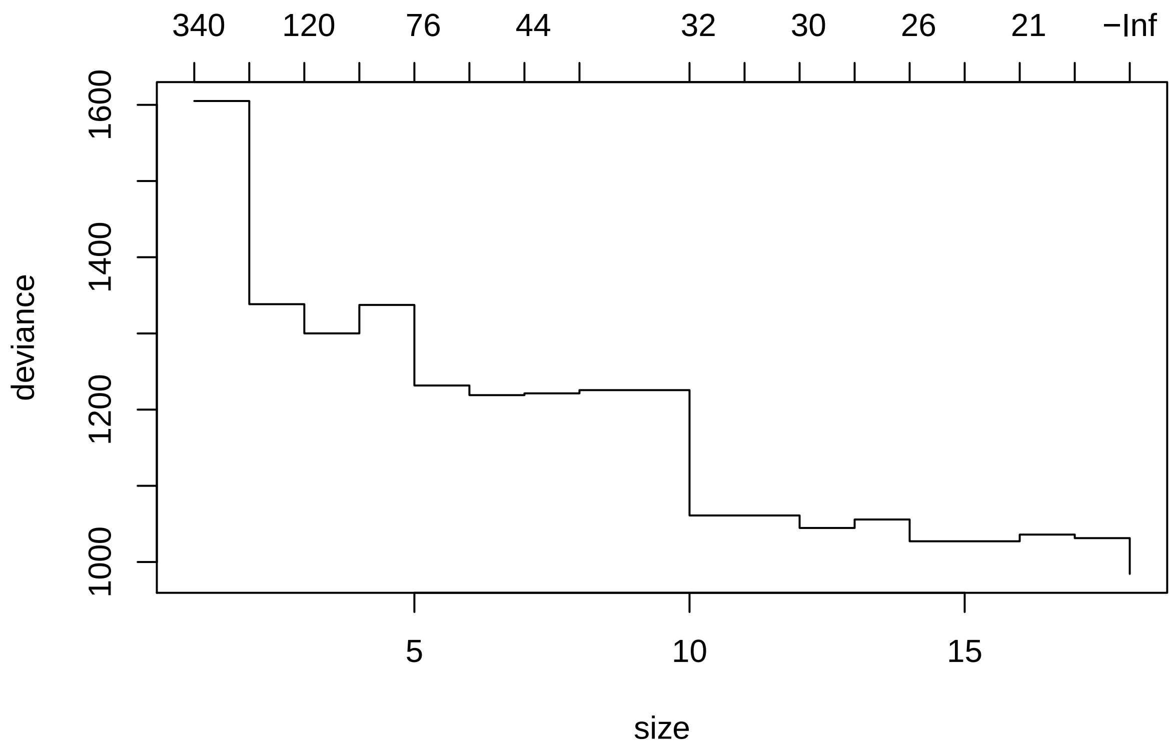

```r
tree.pred = predict(tree.carseats, newdata = Carseats[-train,])
tree.err = with(Carseats[-train,],  mean((Sales-tree.pred)^2))
tree.err
```

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

test MSE = 4.92

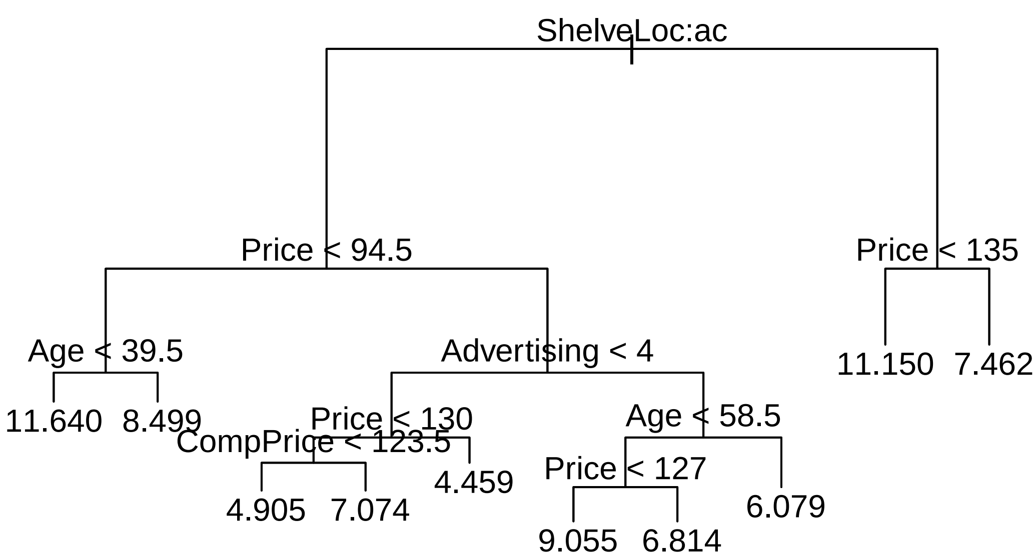
(c)

```r
set.seed(1)
cv.carseats = cv.tree(tree.carseats, FUN = prune.tree)
plot(cv.carseats)
```



pruning tree at size 10 is good

```
prune.carseats = prune.tree(tree.carseats, best = 10)
plot(prune.carseats)
text(prune.carseats)
```




```
prune.predicts = predict(prune.carseats, newdata = Carseats[-train,])
prune.err =  with(Carseats[-train,], mean((Sales - prune.predicts)^2))
prune.err
```

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

test MSE pruning tree at size 10 with CV is 4.91
In this case not much difference between back pruning using CV and simple regression

(d)

```r
require(randomForest)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-14
```

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

```r
bag.carseats = randomForest(Sales ~., data = Carseats, subset = train, mtry=10, ntree=1000, importance='
bag.predict = predict(bag.carseats, Carseats[-train,])
importance(bag.carseats)
```

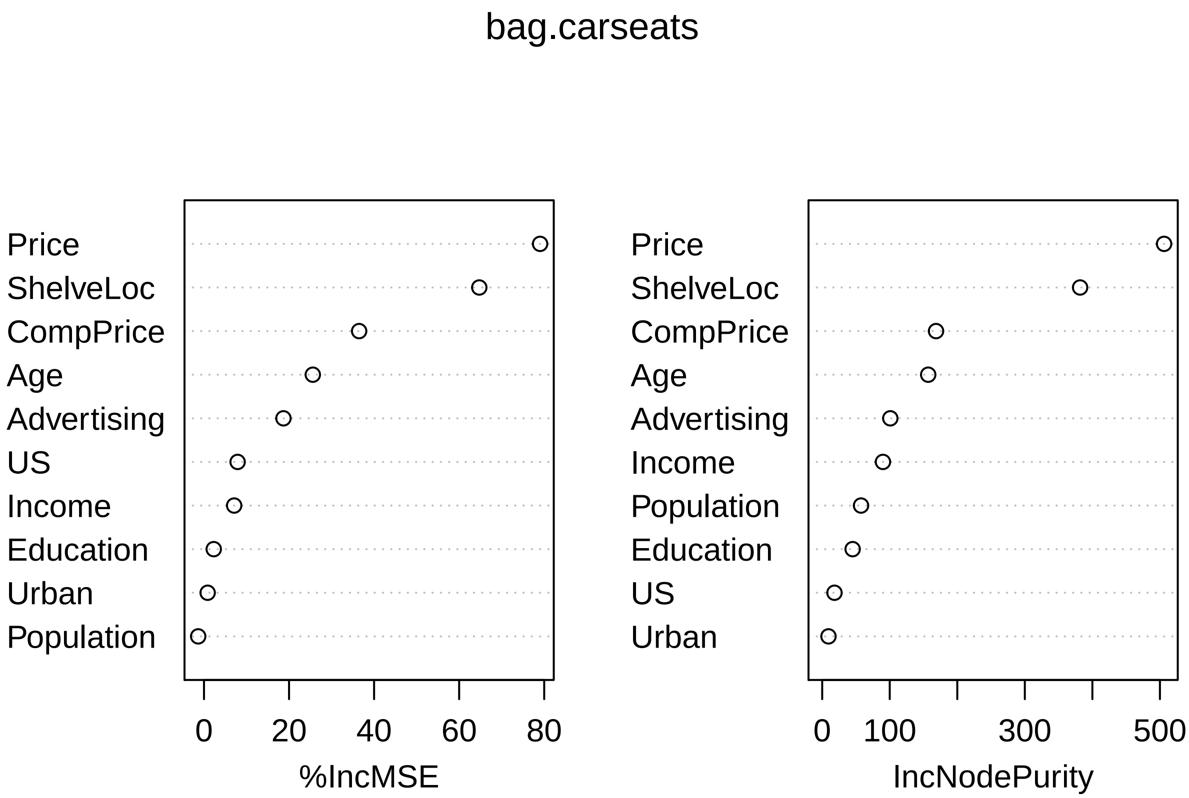
```
##                %IncMSE IncNodePurity
## CompPrice   36.4812214    168.543063
## Income       7.0938296     89.946359
## Advertising 18.6843706    100.823649
## Population  -1.3641849     57.509594
## Price       79.0346806    506.152187
## ShelveLoc   64.7348300    381.814773
## Age         25.5904214    156.951472
## Education    2.2891700     45.081687
## Urban        0.8614702      9.334121
## US           7.9126851     18.184535
```

```r
mean((Carseats[-train,"Sales"]-bag.predict)^2)
```

```
## [1] 2.586853
```

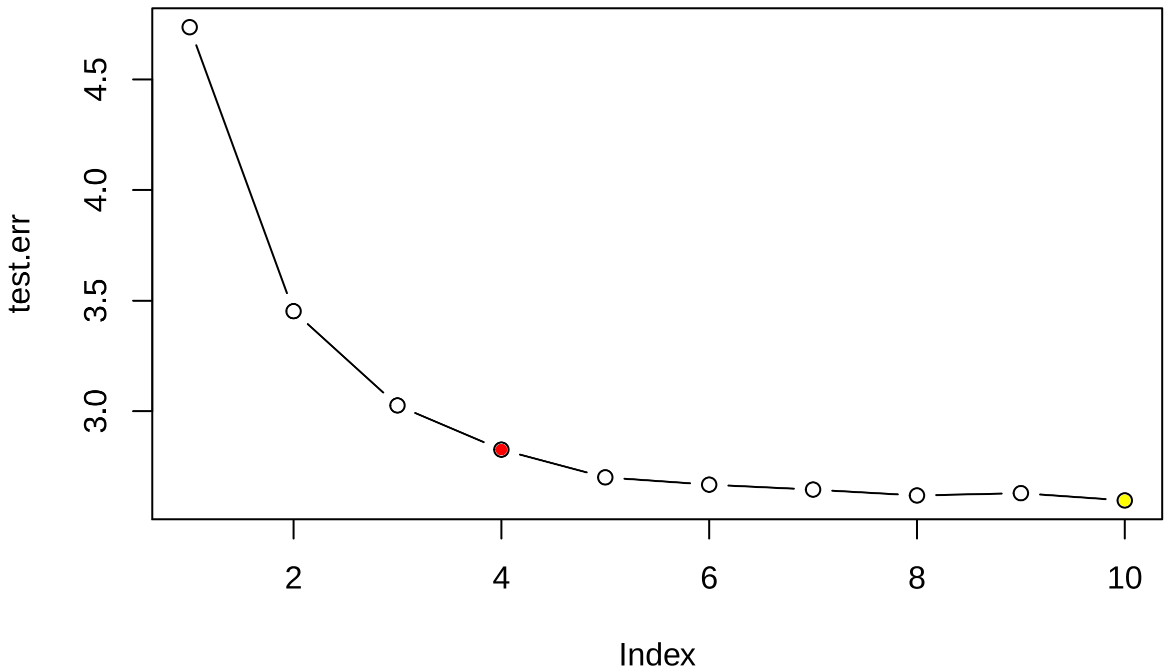
Bagging test MSE is 2.58 which improves a lot compare to back tree pruning using CV
Price, ShelveLoc, CompPrice and Age are important predictors for Sale.

```r
varImpPlot(bag.carseats)
```

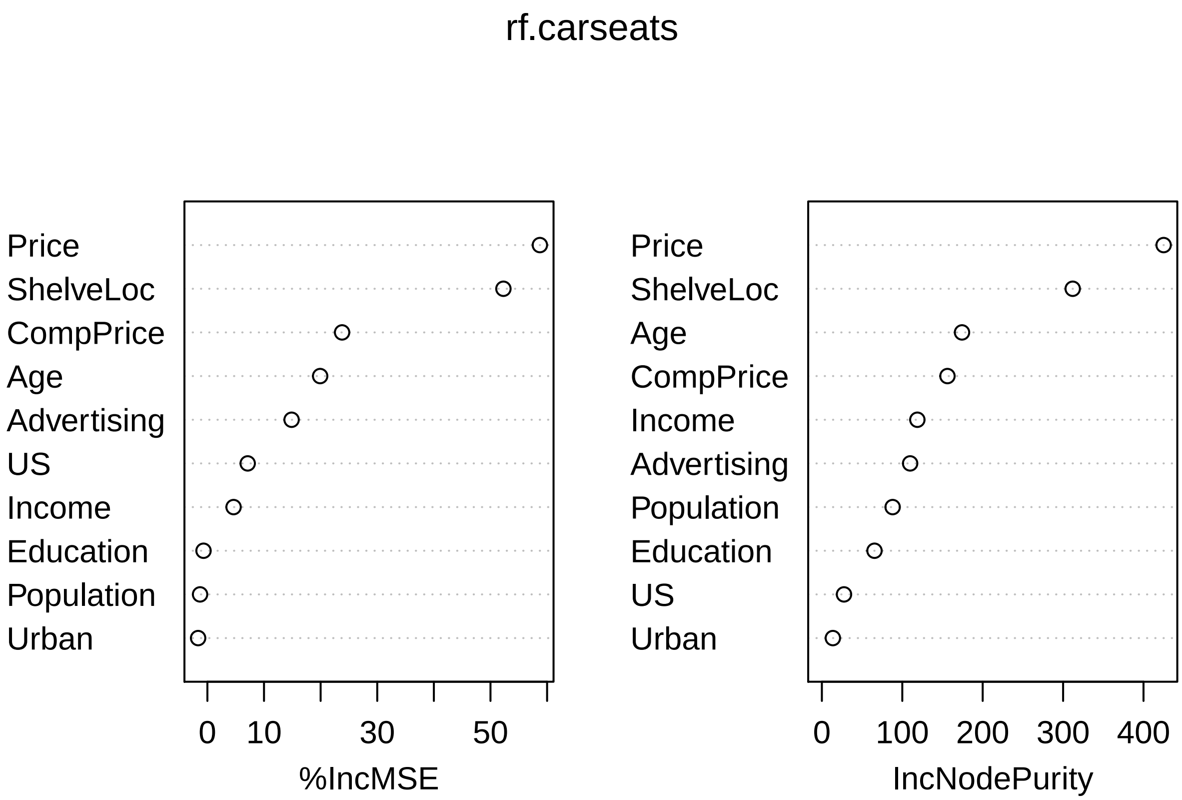


(e)

```
test.err = double(10)
for (mtry in 1:10){
  fit = randomForest(Sales ~., data = Carseats, subset = train, mtry=mtry, ntree=1000, importance=TRUE)
  obb.err = fit$mse[1000]
  pred = predict(fit, Carseats[-train,])
  test.err[mtry] = with(Carseats[-train,], mean((Sales-pred)^2))
}
plot(test.err, type = "b")
points(y=test.err[which.min(test.err)], x=which.min(test.err), col="yellow", pch=20)
points(y=test.err[4], x=4, col="red", pch=20)
```




```
rf.carseats = randomForest(Sales ~., data = Carseats, subset = train, mtry=4, ntree=1000, importance=TRU
varImpPlot(rf.carseats)
```




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

```
##                %IncMSE IncNodePurity
## CompPrice   23.7913562     156.18278
## Income       4.6231162     118.75699
```

```
## Advertising 14.8704051     109.76784
## Population  -1.2950325      88.00063
## Price       58.7265063     425.02687
## ShelveLoc   52.2869953     311.96022
## Age         19.9139571     174.27360
## Education   -0.6865097      65.44015
## Urban       -1.6406557      13.66995
## US           7.1145061      27.52258
```

Price and ShelveLoc are the most important predictors

```r
test.err[4]
```

```
## [1] 2.826501
```

test MSE for randforest is 2.82

Problem 3

Chapter 8, Exercise 10

```r
require(gbm)
```

```
## Loading required package: gbm
```

```
## Loaded gbm 2.1.5
```

```r
require(glmnet)
```

```
## Loading required package: glmnet
```

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

```
## Loaded glmnet 3.0
```

```r
attach(Hitters)
```

(a)

```r
nrow(Hitters)
```

```
## [1] 322
```

```r
Hitters = Hitters[-which(is.na(Hitters$Salary)),]
nrow(Hitters)
```

```
## [1] 263
```

```r
Hitters$Salary = log(Hitters$Salary)
```

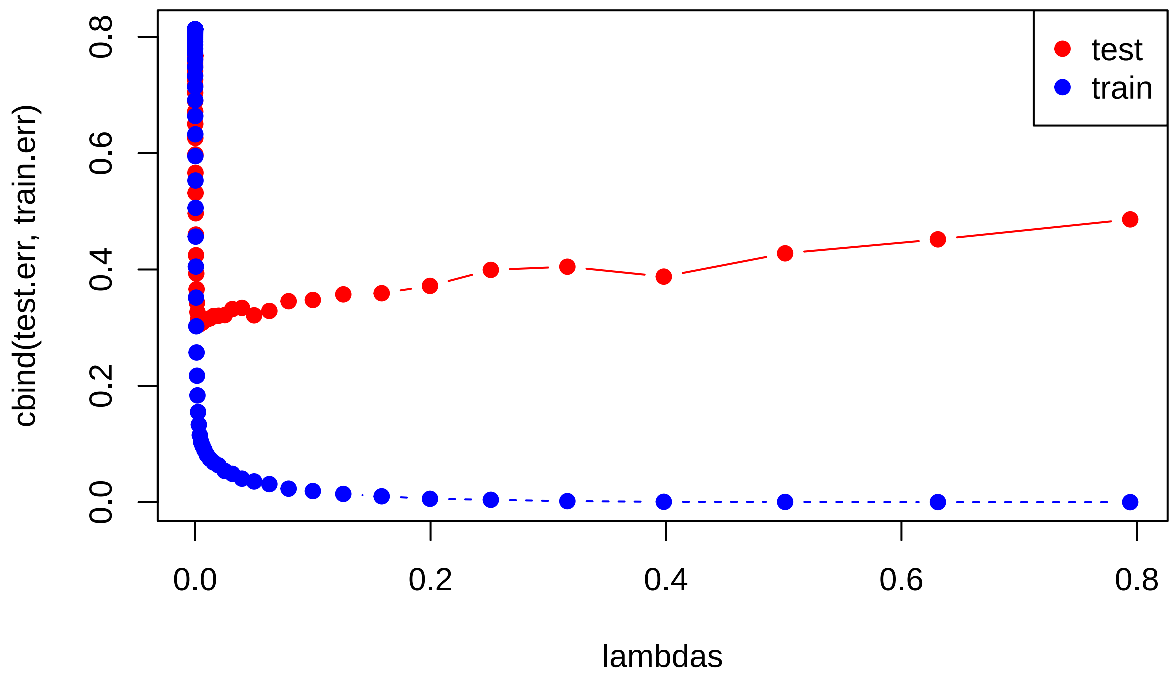
(b)

```r
train = sample(nrow(Hitters), nrow(Hitters)/2)
```

(c) and (d)

```r
lambdas = 10^ (seq(from=-10, to=-0.1, by=0.1))
train.err = rep(NA, length(lambdas))
test.err = rep(NA, length(lambdas))
```

```
for (l in 1:length(lambdas)){
  boost.hitters = gbm(Salary ~., data=Hitters[train,], distribution="gaussian", n.trees = 1000, shrinkag
  train.pred = predict(boost.hitters, Hitters[train,], n.trees = 1000)
  test.pred = predict(boost.hitters, Hitters[-train,], n.trees = 1000)
  train.err[l] = with(Hitters[train,], mean((Salary-train.pred)^2))
  test.err[l] = with(Hitters[-train,], mean((Salary-test.pred)^2))
}
 matplot(lambdas, cbind(test.err,train.err ), pch=19, col=c("red","blue"), type="b" )
 minTest = lambdas[which.min(test.err)]
 mintTrain = lambdas[which.min(train.err)]
 legend("topright", legend = c("test","train"), col=c("red","blue"), pch=19)
```




```
cat("Test error ", min(test.err))
```

```
## Test error  0.3059439
```

```
cat("\n")
```

```
cat("lambda for Test error ", lambdas[which.min(test.err)])
```

```
## lambda for Test error  0.003981072
```

(e)

```
lm.fit =  lm(Salary ~., data=Hitters[train,])
lm.pred = predict(lm.fit, Hitters[-train,])
lm.err = mean((Hitters[-train,"Salary"]-lm.pred)^2)
lm.err
```

```
## [1] 0.4822441
```

Linear regression test MSE is 0.48

```
x.train = model.matrix(Salary ~. , data = Hitters[train,])
y = Hitters[train, "Salary"]
```

```r
x.test = model.matrix(Salary ~. , data = Hitters[-train,])
lasso.fit = glmnet(x.train, y)
lasso.pred = predict(lasso.fit, newx = x.test)
lasso.err = mean((Hitters[-train,"Salary"]-lasso.pred)^2)
lasso.err
```

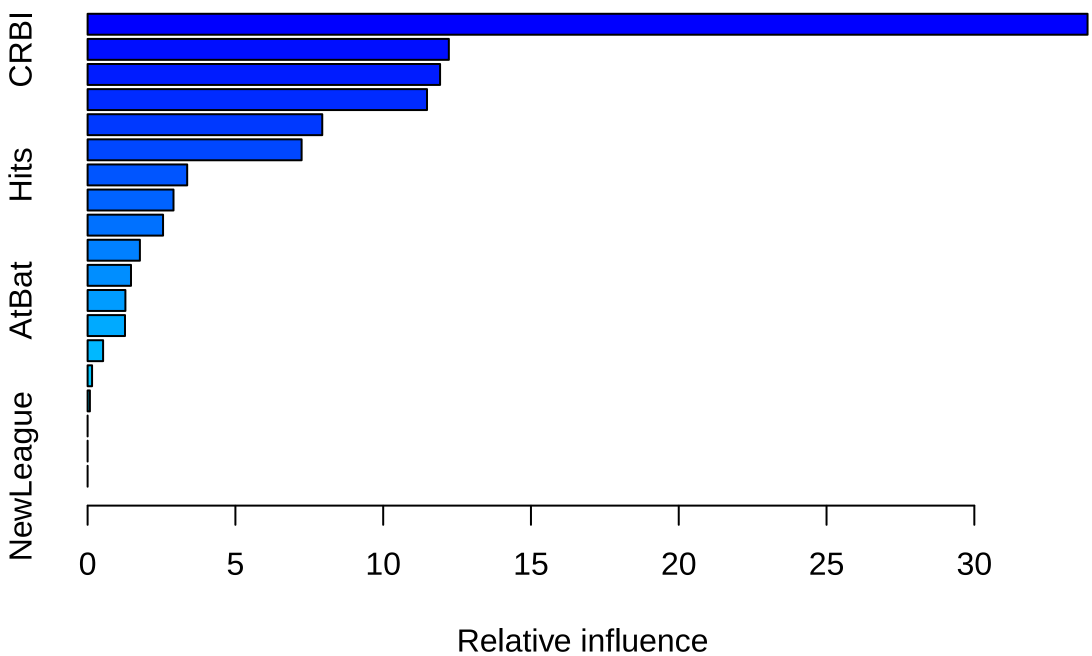
```
## [1] 0.4888379
```

lasso test MSE is 0.48

comparing with Boosting that has much lower test MSE 0.30

(f)

```r
boost.fit = gbm(Salary~., data=Hitters[train,], distribution = "gaussian", n.trees = 1000, shrinkage = 1
summary(boost.fit)
```




```
##                 var     rel.inf
## CRuns         CRuns 33.83096029
## CRBI           CRBI 12.22137568
## CAtBat       CAtBat 11.92479014
## CHits         CHits 11.48232625
## Years         Years  7.93890451
## CWalks       CWalks  7.24060206
## Hits           Hits  3.36887398
## Runs           Runs  2.90614834
## CHmRun       CHmRun  2.55273029
## Walks         Walks  1.76941381
## RBI             RBI  1.46813910
## AtBat         AtBat  1.27801707
## PutOuts     PutOuts  1.26518353
## Errors       Errors  0.52422259
## Assists     Assists  0.15027112
## HmRun         HmRun  0.07804127
```

```
## League       League  0.00000000
## Division   Division  0.00000000
## NewLeague NewLeague  0.00000000
```

(g)

```
rf.hitters =  randomForest(Salary~., data = Hitters[train,], ntree=500, mtry=19)
rf.pred = predict(rf.hitters, Hitters[-train,])
mean((Hitters[-train, "Salary"] - rf.pred)^2)
```

```
## [1] 0.3159374
```

Bagging test MSE is 0.31
Very close to boosting 0.30

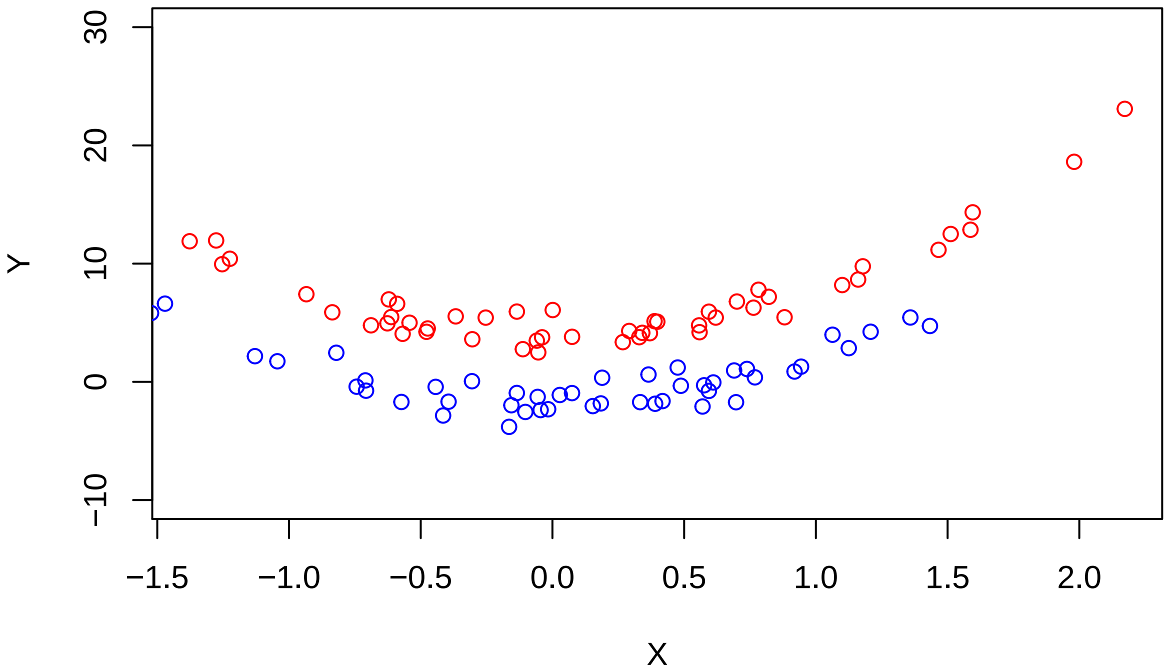
Problem 4

Chapter 9, Exercise 4

```
require(e1071)
```

```
## Loading required package: e1071
```

```
set.seed(1)
x = rnorm(100, 0, 1)
epsilon = rnorm(100, 0, 1)
beta1= 4
degree = 2
classDiff = 3
y = beta1*(x^degree) + 1 + epsilon
class = sample(100, 50)
y[class] = y[class] + classDiff
y[-class] = y[-class] - classDiff
plot(x[class], y[class], col="red", xlab = "X", ylab = "Y", ylim = c(-10, 30))
points(x[-class], y[-class], col="blue")
```



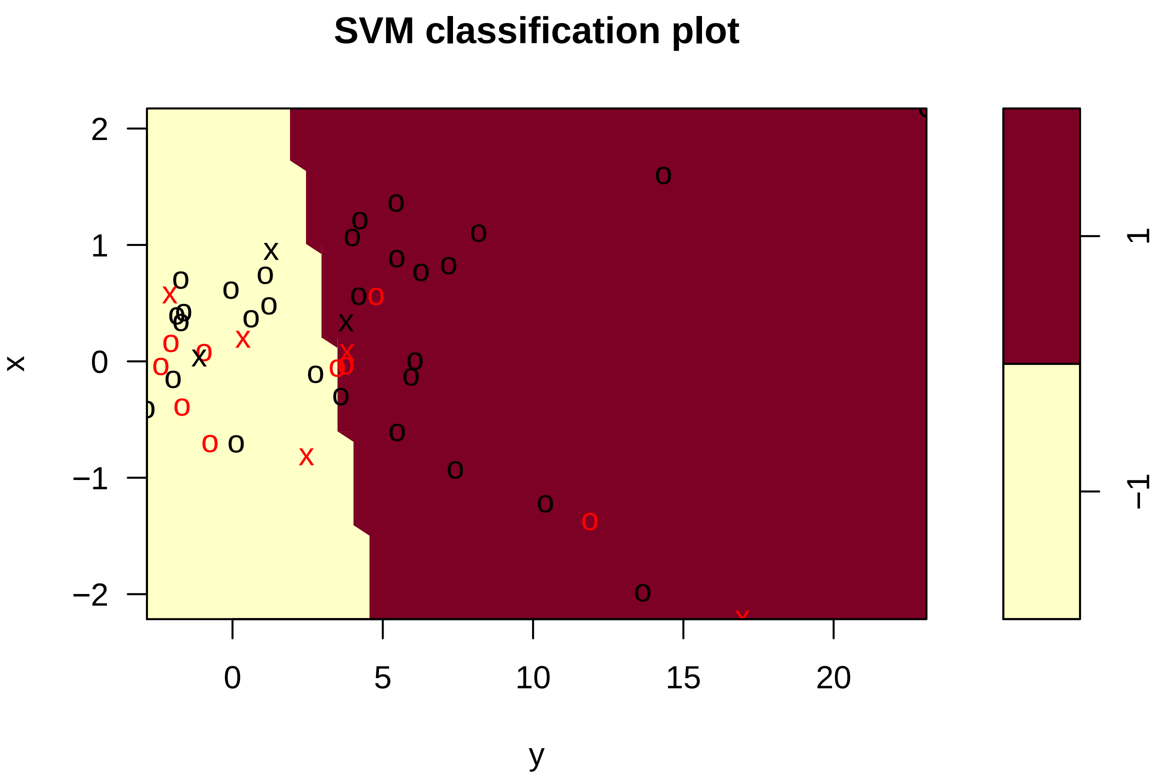
polynomial on linear

```
cst = c(0.01, 0.1, 1, 5, 10, 100, 1000)
label = rep(-1, 100)
label[class] = 1
data = data.frame(x=x, y=y, label=as.factor(label))

best.tune(svm, label~. , data=data[train,], kernel="linear")
```

```
##
## Call:
## best.tune(svm, label ~ ., data = data[train, ], kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  27
```

```
svm.linear = svm(label~., data=data[train,], kernel="linear", cost=1)
plot(svm.linear, data[train,])
```

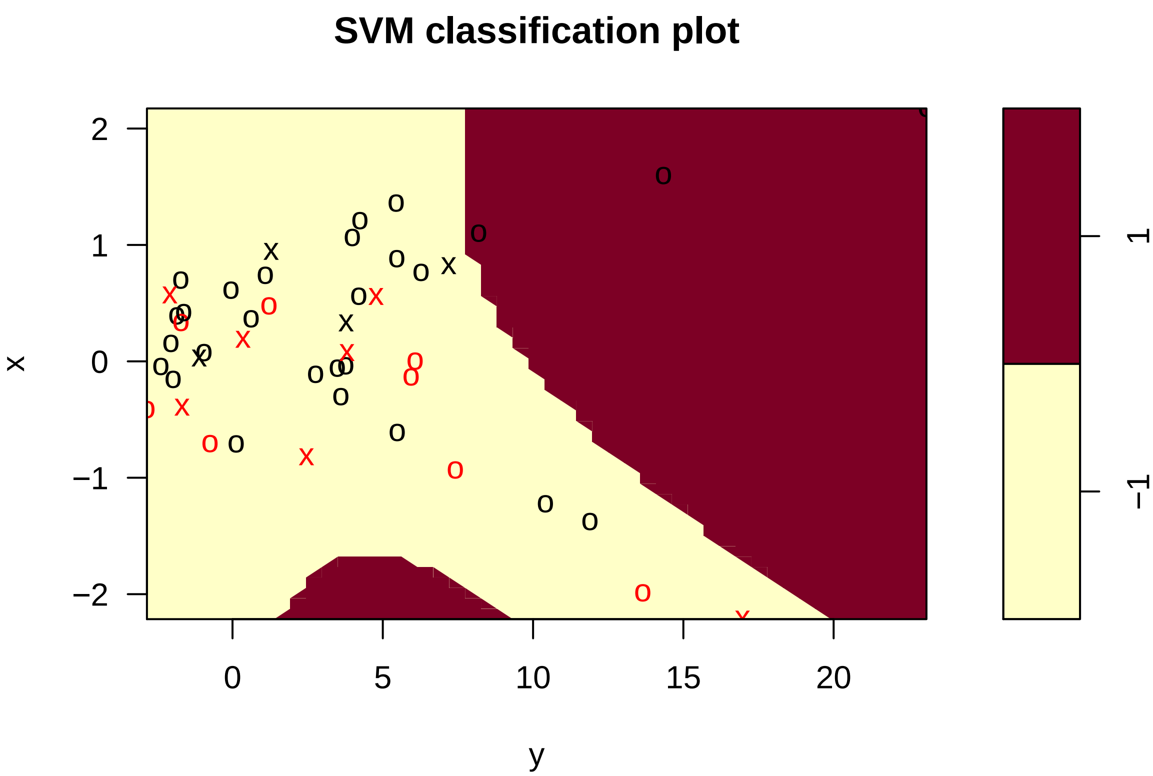


polynomial on training

```r
best.tune(svm, label~. , data=data[train,], kernel="polynomial")
```

```
##
## Call:
## best.tune(svm, label ~ ., data = data[train, ], kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  38
```

```r
svm.poly = svm(label~., data=data[train,], kernel="polynomial", degree=3, cost=1)
plot(svm.poly, data[train,])
```

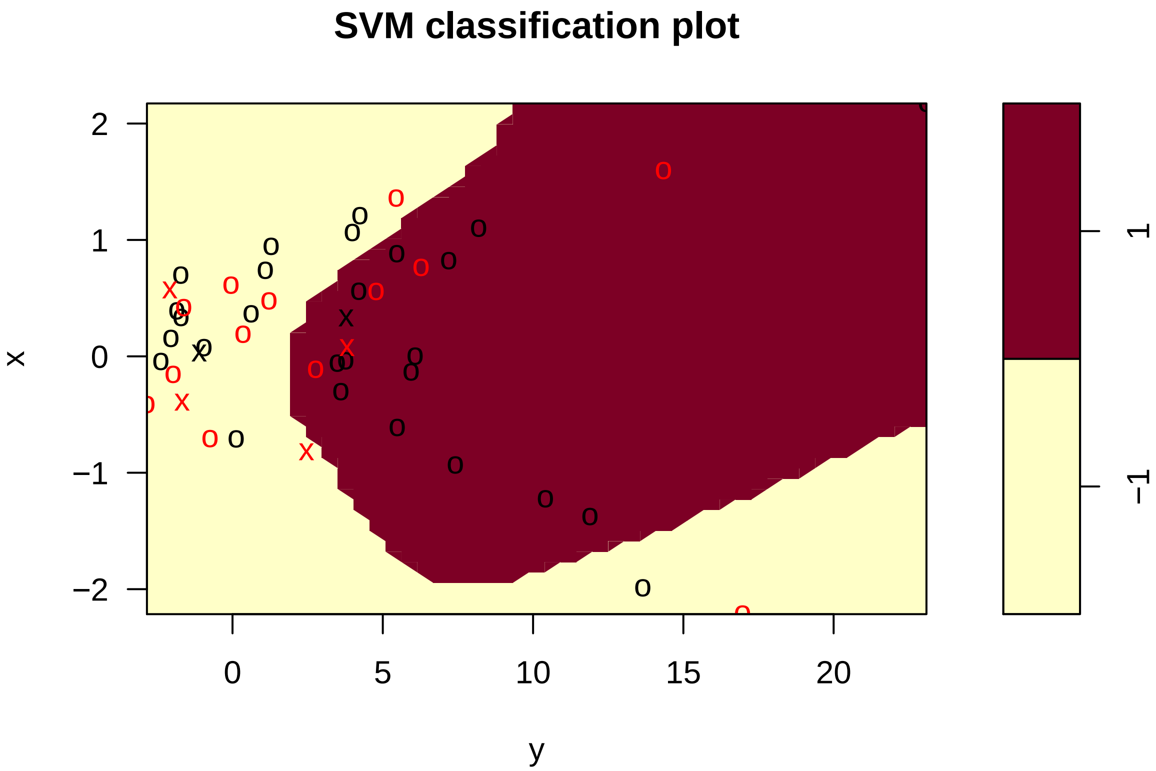


radial on training

```
best.tune(svm, label~. , data=data[train,], kernel="radial")
```

```
##
## Call:
## best.tune(svm, label ~ ., data = data[train, ], kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  24
```

```
svm.radial = svm(label~., data=data[train,], kernel="radial", cost=1)
plot(svm.radial, data[train,])
```



comparing prediction on test results

```r
cat("\npoly")
```

```
##
## poly
```

```r
table(label[-train], predict(svm.poly, data[-train,]))
```

```
##
##       -1  1
##   -1 23  1
##   1  24  6
```

```r
(1 - ((23+6)/(23+6+24+1)))*100
```

```
## [1] 46.2963
```

```r
cat("\nlinear")
```

```
##
## linear
```

```r
table(label[-train], predict(svm.linear, data[-train,]))
```

```
##
##       -1  1
##   -1 18  6
##   1   1 29
```

```r
(1 - ((18+29)/(18+29+6+1)))*100
```

```
## [1] 12.96296
```

```r
cat("\nradial")
```

```
##
## radial
```

```r
table(label[-train], predict(svm.radial, data[-train,]))
```

```
##
##      -1  1
##   -1 20  4
##   1   0 30
```

```r
(1 - (20+30)/(20+30+0+4))*100
```

```
## [1] 7.407407
```

Radial is better than linear, and linear is better than polynomial degree 3

Problem 5

Chapter 9, Exercise 7

(a)

```r
require(ISLR)
require(e1071)
attach(Auto)
y =ifelse(Auto$mpg> mean(Auto$mpg), 1 , 0)
Auto$mpglevel = as.factor(y)
cst = c(0.01, 0.1, 1, 5, 10, 100, 1000)
set.seed(1)
```

(b)

```r
set.seed(1)
tune.out = tune(svm, mpglevel~. , data=Auto, kernel="linear", ranges = list(cost = cst))
tune.out$best.parameters
```

```
##   cost
## 3    1
```

```r
tune.out$best.performance
```

```
## [1] 0.007628205
```

For linear kernel cost 1 has been chosen with lowest 0.0076 cost

(c)

```r
set.seed(1)
tune.out = tune(svm, mpglevel~. , data=Auto, kernel="polynomial", ranges = list(cost = cst), degree = c
tune.out$best.parameters
```

```
##   cost
## 7 1000
```

```
tune.out$best.performance
```

```
## [1] 0.2170513
```

For polynomial kernel cost 1000 has been chosen with lowest 0.21 cost

```
set.seed(1)
tune.out = tune(svm, mpglevel~. , data=Auto, kernel="radial", ranges = list(cost = cst), gama = c(0.01,
tune.out$best.parameters
```

```
##   cost
## 7 1000
```

```
tune.out$best.performance
```

```
## [1] 0.007692308
```

For radial kernel cost 1000 has been chosen with lowest 0.0076 cost

Problem 6

```
library(kernlab)
set.seed(1)

data(reuters)
y = rlabels
x = reuters

train = sample(40,20)

#Spectrum kernel
len = c(2:7)
err.spectrum = rep(NA,6)
for (l in len){
  sk = stringdot(type="spectrum", length=l, normalized=TRUE)
  svp = ksvm(x[train], y[train], kernel=sk, scale=c(), cross=5)
  err.spectrum[l-1] = cross(svp)
}
which.min(err.spectrum)+1
```

```
## [1] 3
```

spectrum with length 3 has the lowest error

```
sk = stringdot(type="spectrum", length=3, normalized=TRUE)
svp = ksvm(x[train], y[train], kernel=sk, scale=c(), cross=5)
pred = predict(svp, x[-train])
table(pred, y[-train])
```

```
##
## pred    acq crude
##   acq     9     0
##   crude   2     9
```

Prediction Spectrum with length 3! 2 errors, 10% !

```r
#gappy kernel
#sgk = gapweightkernel(length=2,lambda=0.1,normalized=TRUE,use_characters=TRUE)
#problem in stringkernels so using pre-computed kernel matrices
#ker.len = read.csv(paste"/Users/rboustan/Documents/Stat202/MySolutions/hw7/matrices/len2lam0.1.csv")

len = c(2:7)
err.gappy = rep(NA,6)
for (l in len){
  ker = read.csv(paste("/Users/rboustan/Documents/Stat202/MySolutions/hw7/matrices/len", l,"lam0.1.csv"
  ker = as.kernelMatrix(as.matrix(ker))
  svp = ksvm(x=ker[train,-1],y=rlabels[train],cross=5)
  err.gappy[l-1] = cross(svp)
}
which.min(err.gappy)+1
```

```
## [1] 4
```

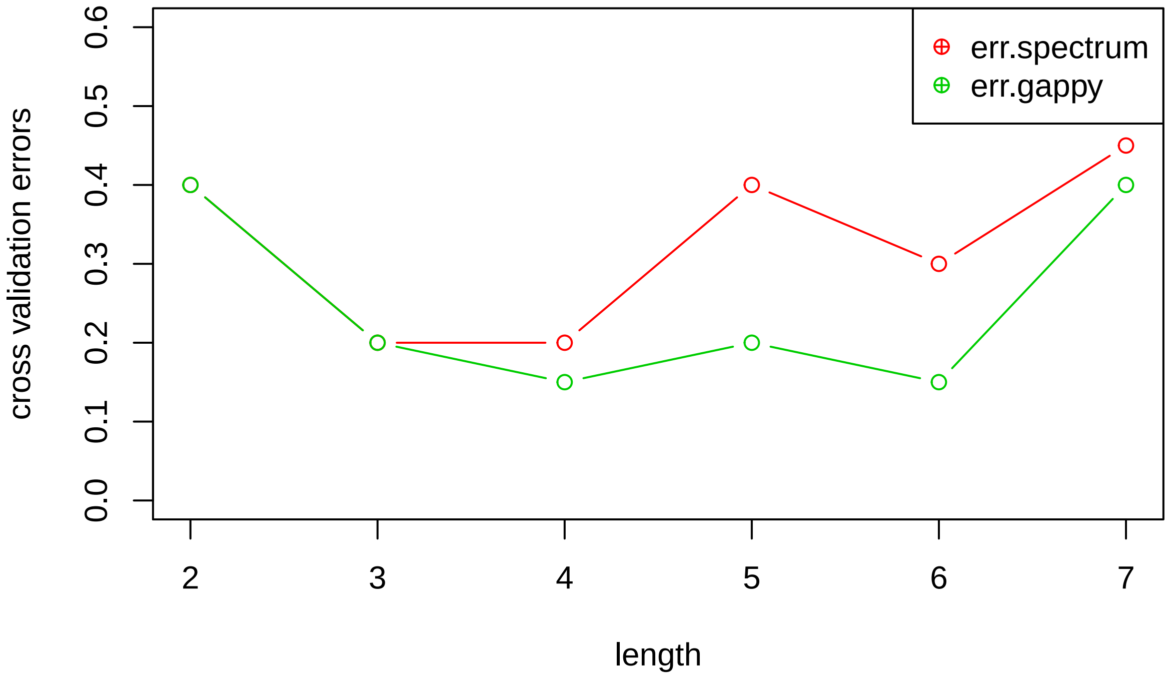
gappy with length 4 has the lowest error

```r
ker = read.csv(paste("/Users/rboustan/Documents/Stat202/MySolutions/hw7/matrices/len4lam0.1.csv", sep =
ker = as.kernelMatrix(as.matrix(ker))
svp = ksvm(x=ker[train,-1],y=rlabels[train],cross=5)
pred = predict(svp, ker[-train,-1])
table(pred, y[-train])
```

```
##
## pred    acq crude
##   acq    11     1
##   crude   0     8
```

Prediction gappy with length 4! 1 error, 5% !
So gappy with length 4 is slightly better than Spectrum with length 3!

```r
plot(x=len, y=err.spectrum, type="b" , col="2", ylim=c(0,0.6), xlim=c(2,7), ylab="cross validation erro
points(x=len, y=err.gappy, col="3", type = "b")
legend("topright",legend=c("err.spectrum", "err.gappy"), col=c(2,3), pch=10)
```


err.spectrum
err.gappy
cross validation errors
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2
3
4
5
6
7
length