HW7

Anish Mohan

November 16, 2015

Q1

Q2

- Booststrap estimates for P(Class is Red | X)={ 0.1, 0.15, 0.2, 0.2, 0.55,0.6, 0.6, 0.65, 0.7, 0.75}
- Majority Approach: X=Red
- Number of times P(Class is Red | X)>0.5. In this case 6 times P(Class is Red | X)>0.5 hence X=Red
- Average Approach: X=Green
- Take average of the probability values. i.e P(Class is Red| X)=0.45. Hence P(Class is Green|X)=0.55. X=Green

Q3

- 3a.

```
library(tree)
library(ISLR)
attach(Carseats)
set.seed(1)
train=sample(1:nrow(Carseats),200)
Carseats.train=Carseats[train,]
Carseats.test=Carseats[-train,]
```

- 3b.

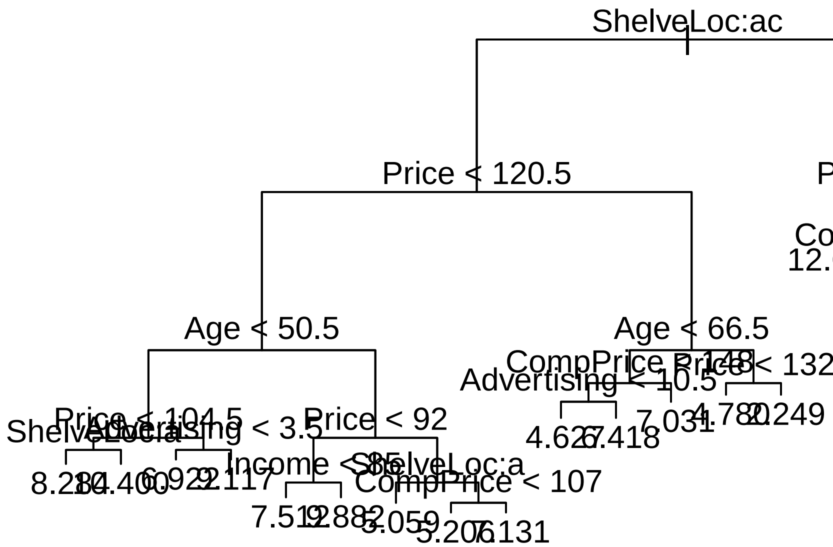
```
tree.carseats=tree(Sales~.,Carseats,subset=train)
summary(tree.carseats)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats, subset = train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Advertising" "Income"
## [6] "CompPrice"
## Number of terminal nodes:  18
## Residual mean deviance:  2.36 = 429.5 / 182
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -4.2570 -1.0360  0.1024  0.0000  0.9301  3.9130
```

```r
par(mfrow=c(1,1))

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

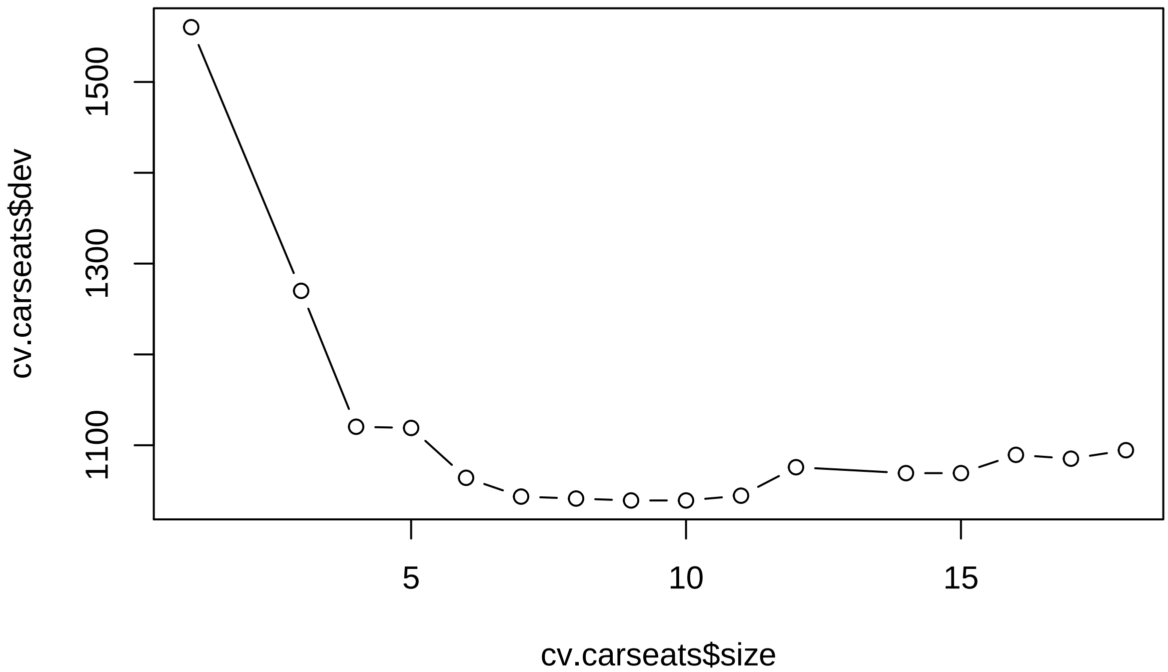
```
## [1] 4.148897
```

```r
plot(tree.carseats)
text(tree.carseats)
```

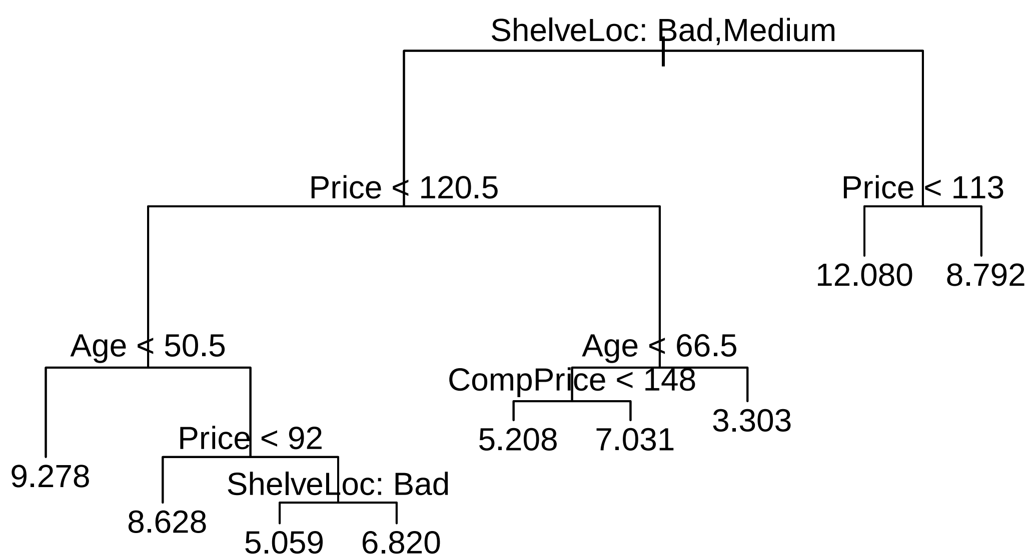


- Variables that were included in the construction of the tree were: ShelveLoc, Price, Age, Advertising, Income and Comp Price. There are 18 terminal nodes or leaves. The RSS for the training data is 2.36
  - The key variable is ShelveLoc which is split into 2 parts Bad and Medium. ShelfLoc = Good is not a critical indicator of sale price. Price is another key varible which is combined when with ShelveLoc=Bad to give to give split at 120.5 and combined with ShelvLoc=Medium to give a split at price=113. Best Sales number seems to be when the ShelveLoc is a Medium and the sale price is 113.
- the test MSE is 4.15
- 3c.

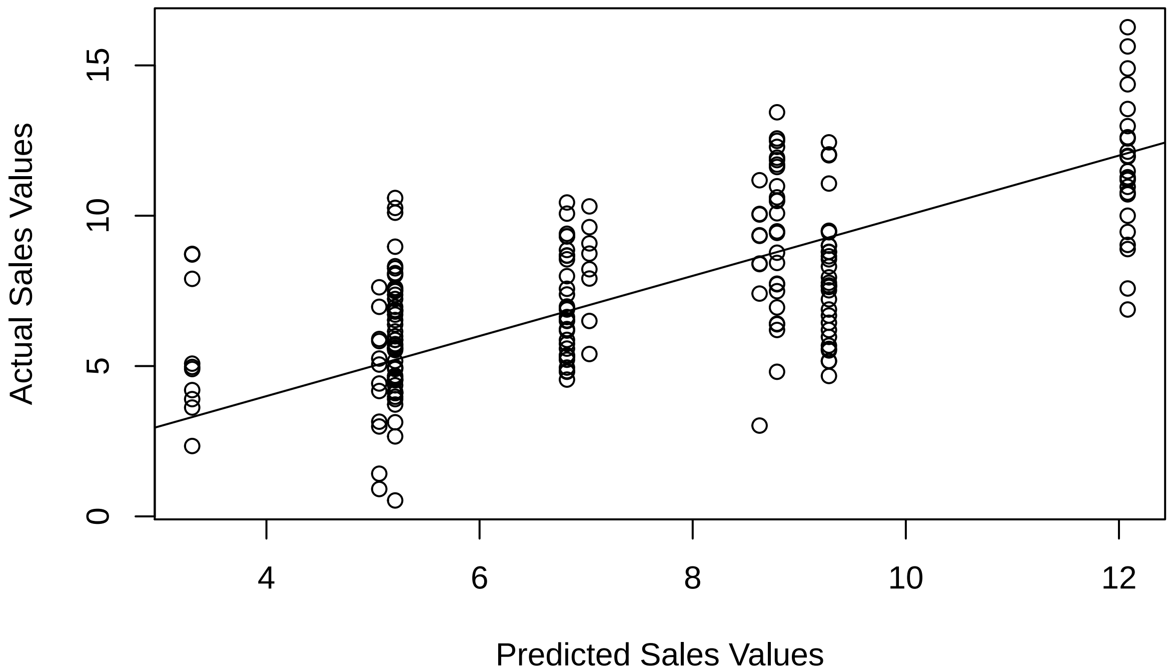
```r
cv.carseats=cv.tree(tree.carseats)
plot(cv.carseats$size,cv.carseats$dev, type='b')
```




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




```
yhat=predict(prune.carseats, newdata=Carseats.test)
plot(yhat, Carseats.test$Sales, xlab="Predicted Sales Values", ylab="Actual Sales Values")
abline(0,1)
```




```r
mean((yhat-Carseats.test$Sales)^2)
```

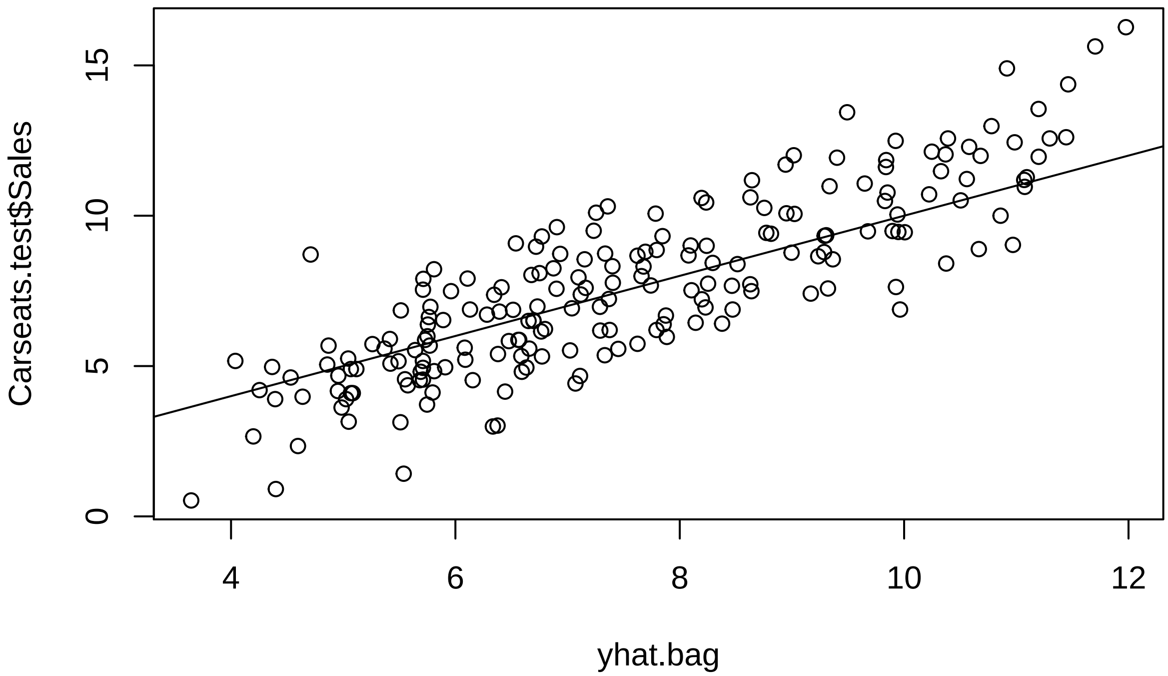
```
## [1] 4.993124
```

- Pruning the tree does not reduce the test MSE. Infact the test MSE increases to 4.99
- 3d.

```r
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

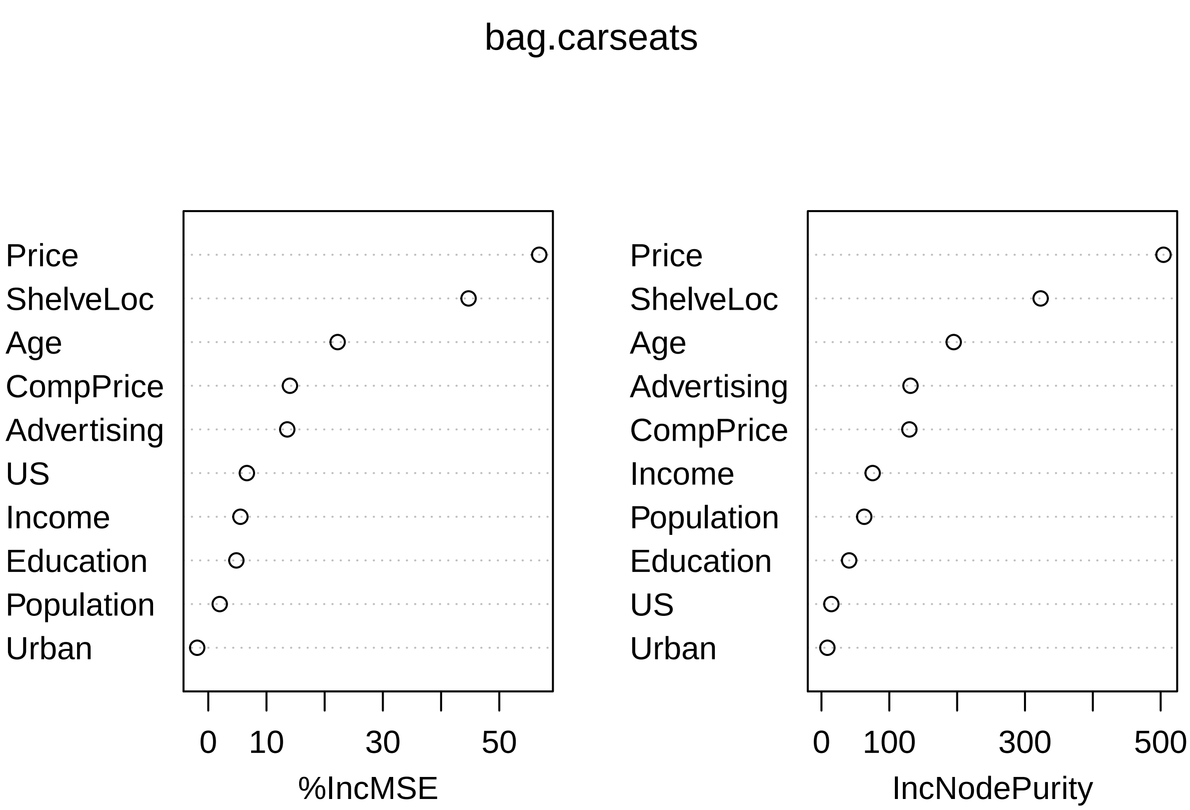
```r
set.seed(1)
bag.carseats=randomForest(Sales~.,data=Carseats,subset=train,mtry=10,importance=T)

yhat.bag=predict(bag.carseats,newdata=Carseats.test)
plot(yhat.bag, Carseats.test$Sales)
abline(0,1)
```




```r
mean((yhat.bag-Carseats.test$Sales)^2)
```

```
## [1] 2.554292
```

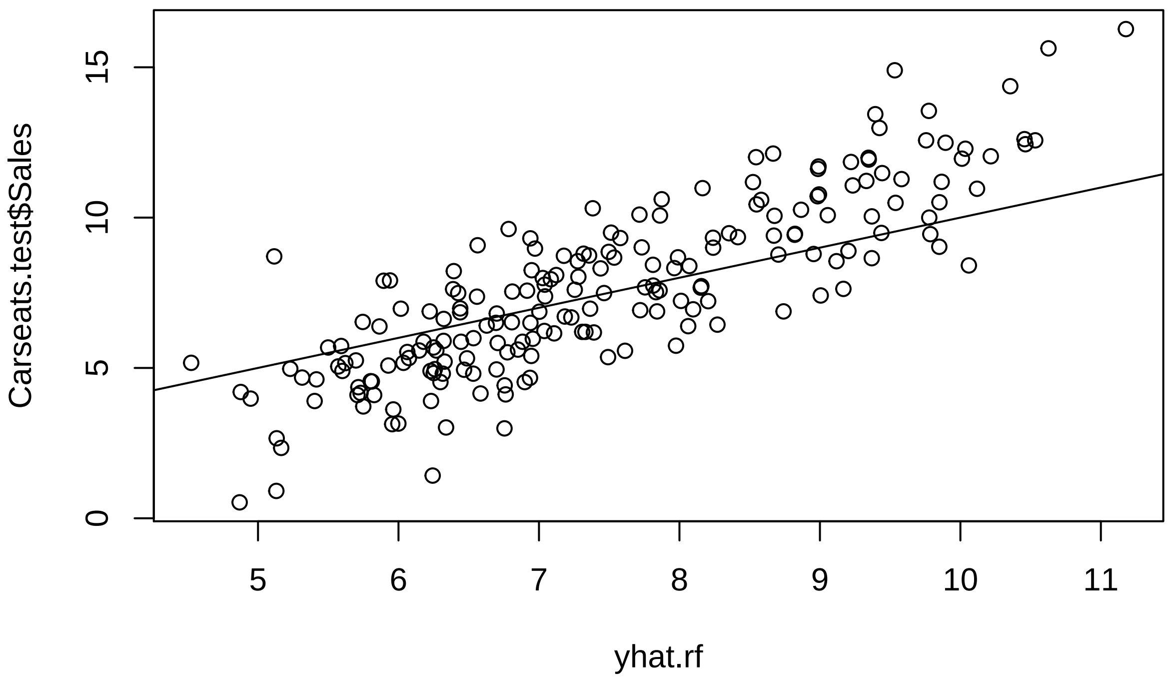
```r
varImpPlot(bag.carseats)
```



  - Test MSE after doing bagging is 2.55
  - Price and ShelveLoc are the two most important variables for determining amount of Sales

- 3e.

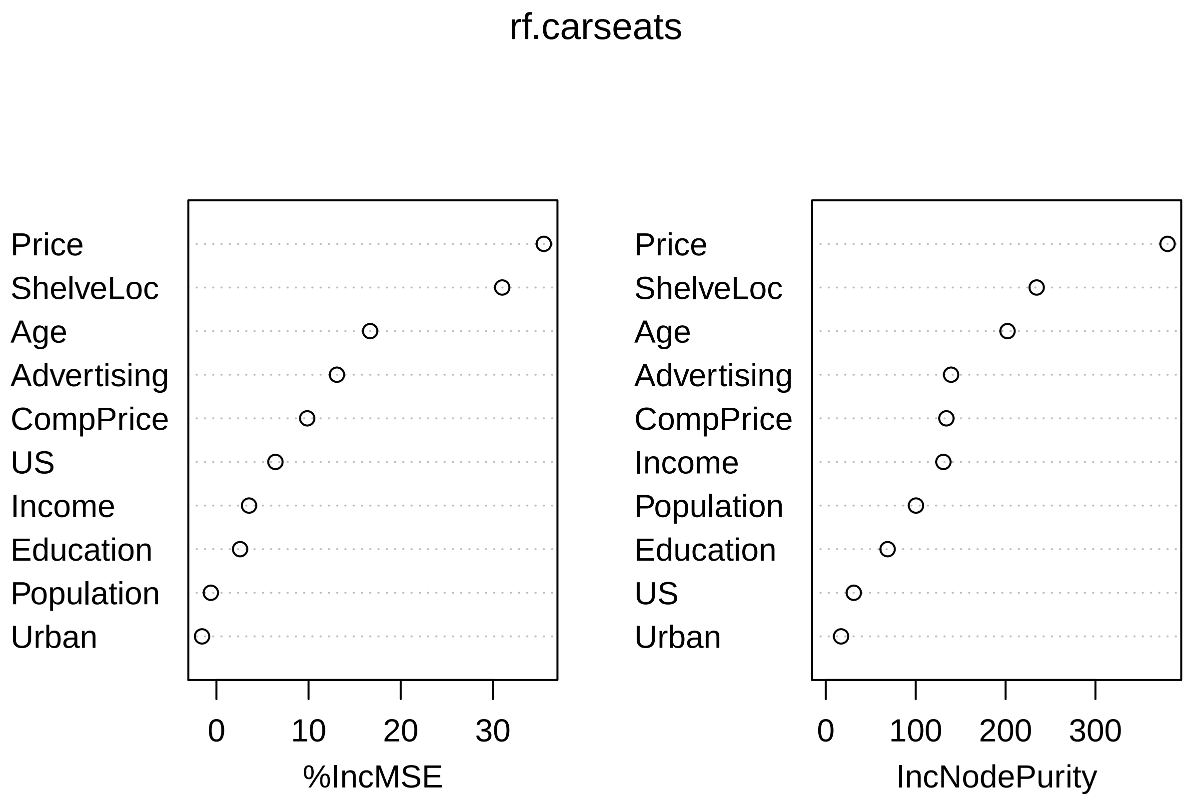
```
set.seed(1)
rf.carseats=randomForest(Sales~.,data=Carseats,subset=train,importance=T)
yhat.rf=predict(rf.carseats,newdata=Carseats.test)
plot(yhat.rf, Carseats.test$Sales)
abline(0,1)
```




```
mean((yhat.rf-Carseats.test$Sales)^2)
```

```
## [1] 3.30763
```

```
varImpPlot(rf.carseats)
```



- Test MSE after doing bagging is 3.3 which is higher than bagging but lower than regression tree
- Price and ShelveLoc are the two most important variables for determining amount of Sales
- We used the default 'm' for random forest that is $sqrt(p)$ i.e $= 3$ variables. Just using 3 variables increased the Test MSE compared to Bagging which used 10 variables.

Q4

- 4a.

```
library(ISLR)
fix(Hitters)
Hitters=na.omit(Hitters)
Hitter=na.omit(Hitters)
Hitter$Salary=log(Hitters$Salary)
```

- 4b.

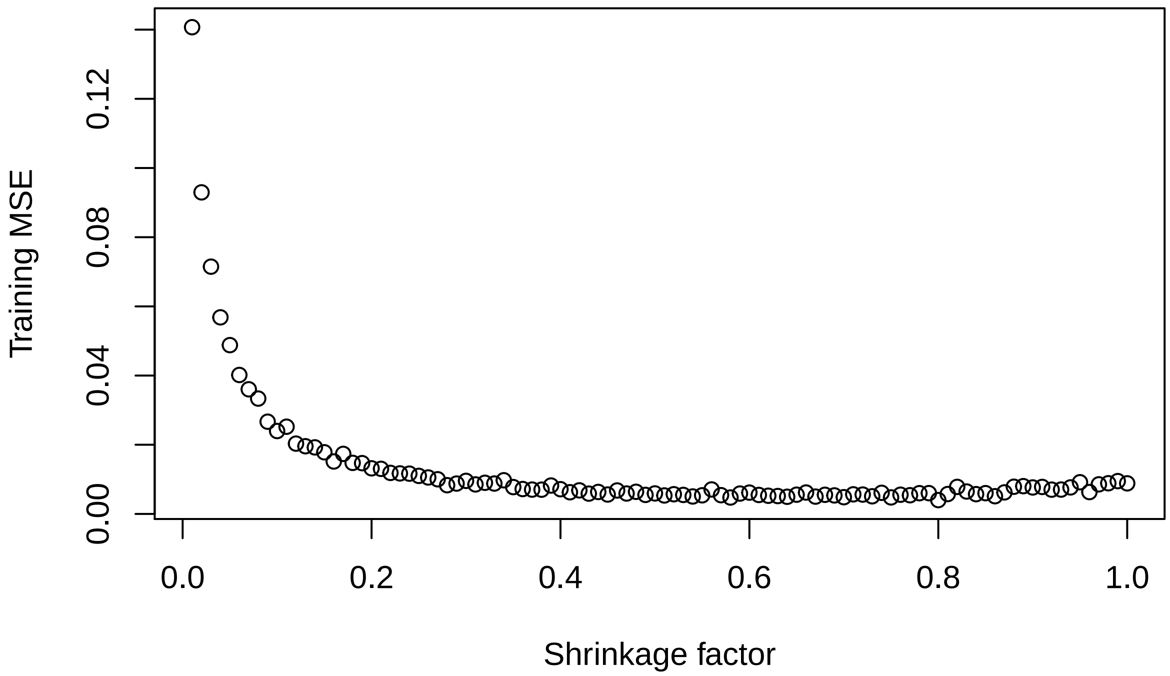
```
train=1:200
test=-train
Hitter.train=Hitter[train,]
Hitter.test=Hitter[-train,]
```

- 4c.

```
library(gbm)
```

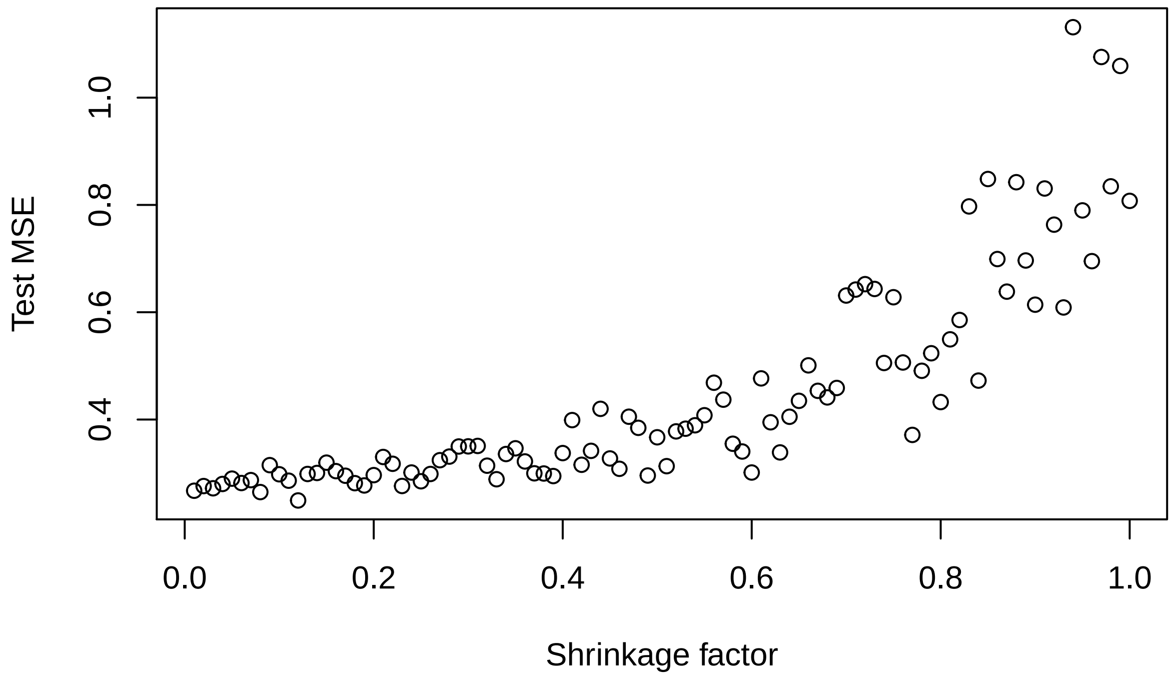
```
## Loading required package: survival
## Loading required package: splines
## Loading required package: lattice
## Loading required package: parallel
## Loaded gbm 2.1.1
```

```
set.seed(1)

train.mse=matrix(data=NA, nrow=100, ncol=2)
test.mse=matrix(data=NA, nrow=100, ncol=2)
for(i in 1:100){
  lambda=i*0.01
  train.mse[i,1]=lambda
  test.mse[i,1]=lambda

  boost.hitter=gbm(Salary~.,data=Hitter.train, distribution="gaussian", n.trees=1000,interaction.de
  train.mse[i,2]=mean(boost.hitter$train.error)

yhat.boost=predict(boost.hitter,newdata=Hitter.test,n.trees=1000)
test.mse[i,2]=mean((yhat.boost-Hitter.test$Salary)^2)
}

plot(train.mse[,1],train.mse[,2], xlab="Shrinkage factor", ylab="Training MSE")
```



- 4d.

```
plot(test.mse[,1],test.mse[,2], xlab="Shrinkage factor", ylab="Test MSE")
```



- 4e.

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

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

```
library(glmnet)
```

```
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-2
```

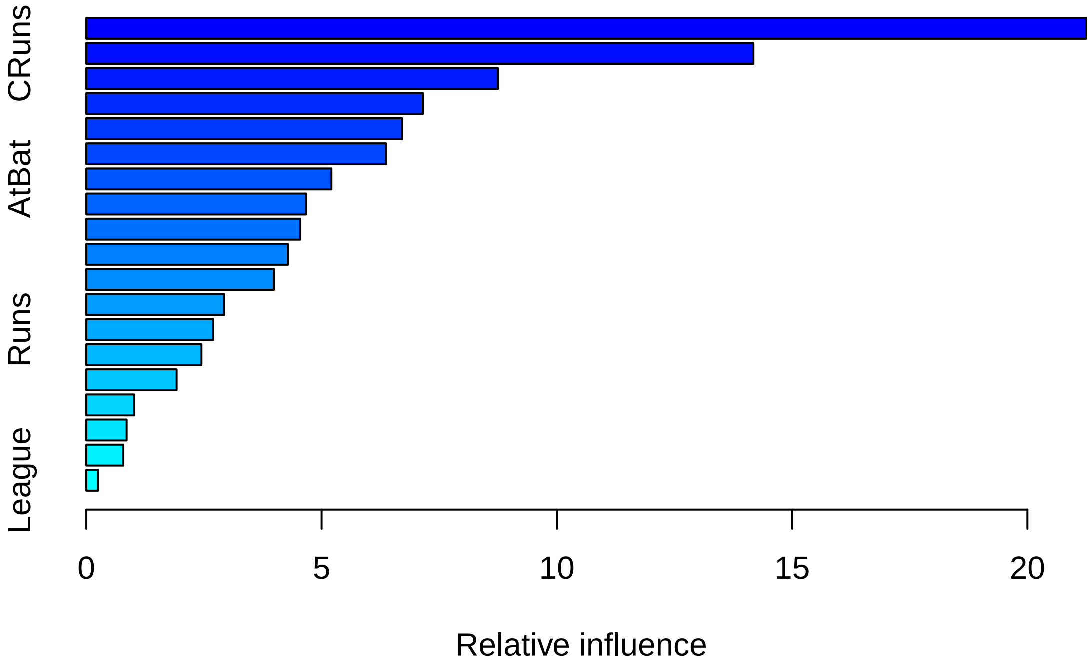
```
set.seed(1)
x=model.matrix(Salary~.,Hitter)
y=Hitter$Salary
y.test=y[test]
grid=10^seq(10,-2,length=100)
ridge.mod=glmnet(x[train,],y[train],alpha=0, lambda=grid, thresh=1e-12)
ridge.pred=predict(ridge.mod,s=212,newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 0.6313728
```

  - Minimum Test MSE: Boosting: 0.249
  - Minimum Test MSE: Linear Model fit: 0.49
  - Minimum Test MSE: Ridge Regression: 0.45

- 4f.

```r
boost.hitter=gbm(Salary~.,data=Hitter.train, distribution="gaussian", n.trees=1000,interaction.dep
 summary(boost.hitter)
```



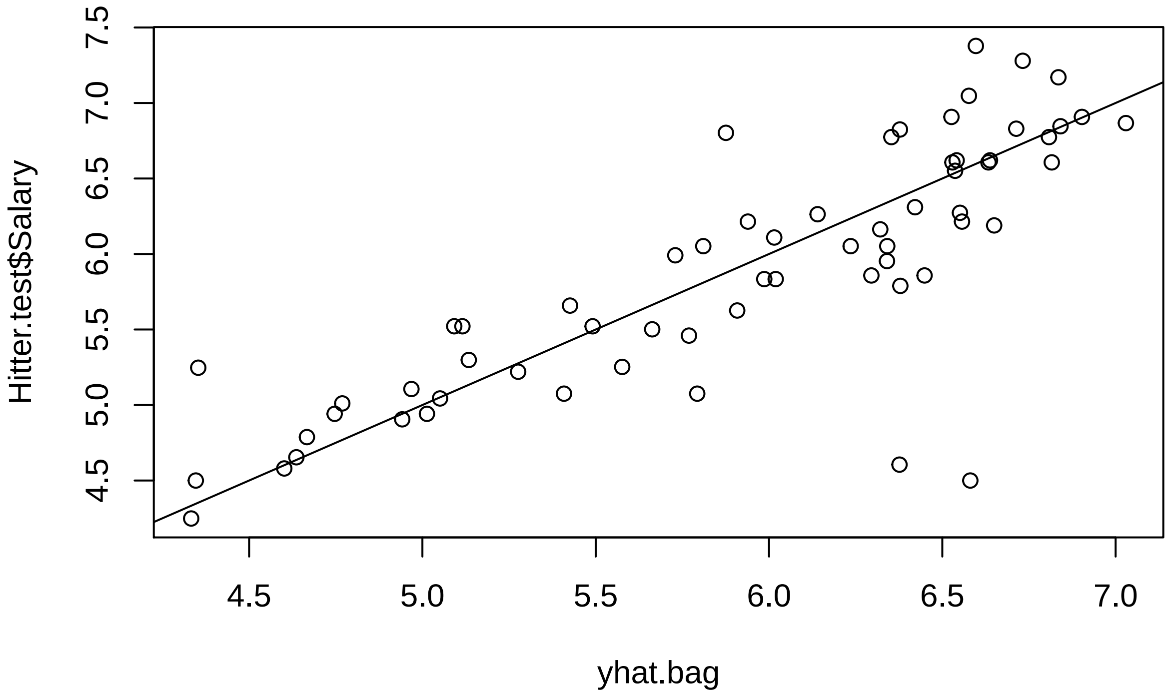

```
##                 var    rel.inf
## CAtBat       CAtBat 21.2500123
## CRuns         CRuns 14.1765669
## CWalks       CWalks  8.7472089
## CRBI           CRBI  7.1509000
## PutOuts     PutOuts  6.7109573
## Walks         Walks  6.3698013
## AtBat         AtBat  5.2080574
## CHmRun       CHmRun  4.6710617
## Years         Years  4.5471386
## RBI             RBI  4.2839575
## Assists     Assists  3.9848415
## Hits           Hits  2.9274869
## Runs           Runs  2.6974009
## HmRun         HmRun  2.4462063
## Errors       Errors  1.9190240
## CHits         CHits  1.0183290
## NewLeague NewLeague  0.8562599
## Division   Division  0.7858884
## League       League  0.2489012
```

  - CAtBat and CRuns are the key variables.

- 4g.

```r
library(randomForest)
set.seed(1)
bag.Hitter=randomForest(Salary~.,data=Hitter.train,mtry=19,importance=T)
yhat.bag=predict(bag.Hitter,newdata=Hitter.test)
```

```r
plot(yhat.bag, Hitter.test$Salary)
abline(0,1)
```




```r
mean((yhat.bag-Hitter.test$Salary)^2)
```

```
## [1] 0.228722
```

- Test MSE using Bagging: 0.23

Q5

```r
library(kernlab)
library(e1071)
set.seed(1)
x=matrix(rnorm(100*2),ncol=2)
x[1:25,]=x[1:25,]+2
x[26:50,]=x[26:50,]-2

y=c(rep(1,60),rep(2,40))
df=data.frame(x=x, y=as.factor(y))

plot(x,col=y)
```

```
![](HW7_files/figure-latex/unnamed-chunk-14-1.pdf)
```

```r
train=sample(100,75)
```

````
test=-train

df.train=df[train,]
df.test=df[test,]

svmfit=svm(y~.,data=df[train,],kernel="radial", gamma=1,cost=1,scale=F)
plot(svmfit,df[train,])
```

![](HW7_files/figure-latex/unnamed-chunk-14-2.pdf)

```r
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = df[train, ], kernel = "radial", gamma = 1,
##     cost = 1, scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  52
##
##  ( 25 27 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

```r
ytrain=predict(svmfit, df[train,])
table(predict=ytrain, truth=df[train,"y"])
```

```
##        truth
## predict  1  2
##       1 36  0
##       2 10 29
```

```r
ytest=predict(svmfit, df[test,])
table(predict=ytest, truth=df[test,"y"])
````

```

```
truth
predict 1 2
1 11 1
2 3 10
```

```r
#Linear filter
svmfit=svm(y~.,data=df[train,],kernel="linear", cost=0.1,scale=F)
plot(svmfit,df[train,])
```

`![](HW7_files/figure-latex/unnamed-chunk-14-3.pdf)`

```r
summary(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = df[train, ], kernel = "linear", cost = 0.1,
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##       gamma:  0.5
##
## Number of Support Vectors:  60
##
##  ( 29 31 )
##
##
## Number of Classes:  2
##
## Levels:
##  1 2
```

```r
ytrain=predict(svmfit, df[train,])
table(predict=ytrain, truth=df[train,"y"])
```

```
##        truth
## predict  1  2
##       1 46 29

```
##       2  0  0
```

```r
ytest=predict(svmfit, df[test,])
table(predict=ytest, truth=df[test,"y"])
```

```
##        truth
## predict  1  2
##       1 14 11
##       2  0  0
```

- With the Radial Kernel, 65 out 75 points have been assigned the correct class in the training data and 21 out of 25 points have been assigned the correct class in the test data.
- With the Linear kernel, only 46 out of 75 points are correctly assigned the correct class in training data and 14 out of 25 points have been assigned the correct class in the test data.

Q6

```r
library(ISLR)
median(Auto$mpg)
```

```
## [1] 22.75
```

```r
mileage=ifelse(Auto$mpg>=median(Auto$mpg),1,0)

df=data.frame(cylinder=Auto$cylinders,displacement= Auto$displacement, horsepower= Auto$horsepower,

set.seed(1)
tune.out=tune(svm,mileage~.,data=df, kernel="linear",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.09179487
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.13288462 0.05551120
## 2 1e-02 0.09179487 0.05812971
```

```
## 3 1e-01 0.09692308 0.06369443
## 4 1e+00 0.09179487 0.04543280
## 5 5e+00 0.10198718 0.04338864
## 6 1e+01 0.11480769 0.05828005
## 7 1e+02 0.11730769 0.06521821
```

```r
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = mileage ~ ., data = df, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.003215434
##
## Number of Support Vectors:  118
##
##  ( 52 66 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
#radial
set.seed(1)
tune.out=tune(svm,mileage~.,data=df, kernel="radial",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100),g
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     1
##
## - best performance: 0.07647436
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03   0.5 0.56115385 0.04344202
## 2  1e-02   0.5 0.56115385 0.04344202
## 3  1e-01   0.5 0.08923077 0.05559893
## 4  1e+00   0.5 0.07897436 0.05443042
```

```
## 5   5e+00   0.5 0.08147436 0.06565669
## 6   1e+01   0.5 0.08910256 0.06377480
## 7   1e+02   0.5 0.08910256 0.06377480
## 8   1e-03   1.0 0.56115385 0.04344202
## 9   1e-02   1.0 0.56115385 0.04344202
## 10  1e-01   1.0 0.56115385 0.04344202
## 11  1e+00   1.0 0.07647436 0.05657355
## 12  5e+00   1.0 0.08160256 0.06250579
## 13  1e+01   1.0 0.08416667 0.06164376
## 14  1e+02   1.0 0.08416667 0.06164376
## 15  1e-03   2.0 0.56115385 0.04344202
## 16  1e-02   2.0 0.56115385 0.04344202
## 17  1e-01   2.0 0.56115385 0.04344202
## 18  1e+00   2.0 0.11724359 0.04962645
## 19  5e+00   2.0 0.11467949 0.05251931
## 20  1e+01   2.0 0.11467949 0.05251931
## 21  1e+02   2.0 0.11467949 0.05251931
## 22  1e-03   3.0 0.56115385 0.04344202
## 23  1e-02   3.0 0.56115385 0.04344202
## 24  1e-01   3.0 0.56115385 0.04344202
## 25  1e+00   3.0 0.33410256 0.15766416
## 26  5e+00   3.0 0.29570513 0.13702698
## 27  1e+01   3.0 0.29570513 0.13702698
## 28  1e+02   3.0 0.29570513 0.13702698
## 29  1e-03   4.0 0.56115385 0.04344202
## 30  1e-02   4.0 0.56115385 0.04344202
## 31  1e-01   4.0 0.56115385 0.04344202
## 32  1e+00   4.0 0.45391026 0.09910221
## 33  5e+00   4.0 0.46160256 0.07848745
## 34  1e+01   4.0 0.46160256 0.07848745
## 35  1e+02   4.0 0.46160256 0.07848745
```

```r
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = mileage ~ ., data = df, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  376
##
##  ( 187 189 )
##
##
## Number of Classes:  2
##
```

```
## Levels:
##  0 1
```

```r
#polynomial
set.seed(1)
tune.out=tune(svm,mileage~.,data=df, kernel="polynomial",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,10
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.07634615
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03   0.5 0.25544872 0.09514919
## 2  1e-02   0.5 0.09185897 0.06420082
## 3  1e-01   0.5 0.08160256 0.06250579
## 4  1e+00   0.5 0.07634615 0.04933672
## 5  5e+00   0.5 0.09166667 0.05108581
## 6  1e+01   0.5 0.09166667 0.04500388
## 7  1e+02   0.5 0.09179487 0.04200857
## 8  1e-03   1.0 0.09442308 0.06846018
## 9  1e-02   1.0 0.08666667 0.06734113
## 10 1e-01   1.0 0.07634615 0.04933672
## 11 1e+00   1.0 0.09166667 0.05108581
## 12 5e+00   1.0 0.09179487 0.04200857
## 13 1e+01   1.0 0.09179487 0.04200857
## 14 1e+02   1.0 0.09179487 0.04200857
## 15 1e-03   2.0 0.08923077 0.06521387
## 16 1e-02   2.0 0.07634615 0.04933672
## 17 1e-01   2.0 0.09166667 0.05108581
## 18 1e+00   2.0 0.09179487 0.04200857
## 19 5e+00   2.0 0.09179487 0.04200857
## 20 1e+01   2.0 0.09179487 0.04200857
## 21 1e+02   2.0 0.09179487 0.04200857
## 22 1e-03   3.0 0.08410256 0.05257348
## 23 1e-02   3.0 0.08660256 0.05261832
## 24 1e-01   3.0 0.09179487 0.04200857
## 25 1e+00   3.0 0.09179487 0.04200857
## 26 5e+00   3.0 0.09179487 0.04200857
## 27 1e+01   3.0 0.09179487 0.04200857
## 28 1e+02   3.0 0.09179487 0.04200857
## 29 1e-03   4.0 0.07891026 0.05003145
## 30 1e-02   4.0 0.09166667 0.05108581
## 31 1e-01   4.0 0.09179487 0.04200857
## 32 1e+00   4.0 0.09179487 0.04200857
## 33 5e+00   4.0 0.09179487 0.04200857
```

```
## 34 1e+01   4.0 0.09179487 0.04200857
## 35 1e+02   4.0 0.09179487 0.04200857
```

```r
bestmod=tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = mileage ~ ., data = df, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5, 1, 2, 3, 4)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##       gamma:  0.5
##      coef.0:  0
##
## Number of Support Vectors:  119
##
##  ( 58 61 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Q7