OptiTree: Advanced Ensemble Strategies for Predictive Accuracy

Preethi Sree Allam

2023-04-17

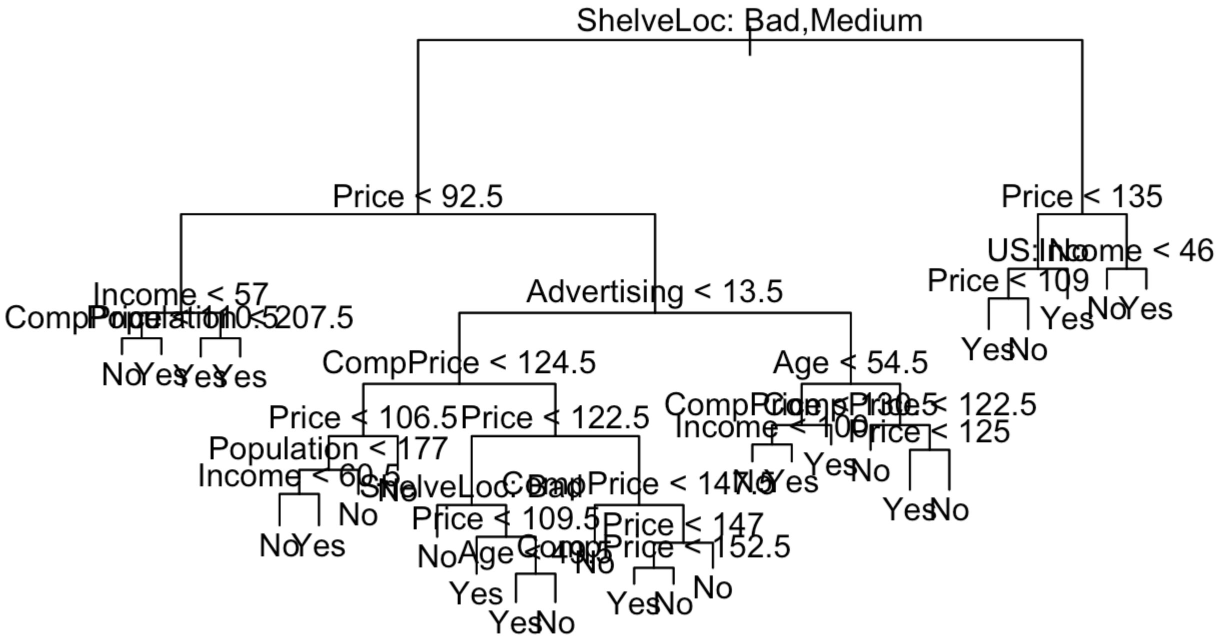
Decision Trees

Fitting Classification Trees

```
library(tree)
library(ISLR2)
attach(Carseats)
High <- factor(ifelse(Sales <= 8, "No", "Yes"))
Carseats <- data.frame(Carseats, High)
tree.carseats <- tree(High ~ . - Sales, Carseats)
summary(tree.carseats)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

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



```
tree.carseats
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 400 541.500 No ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407 Yes ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
##          21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680 Yes ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702 No ( 0.90909 0.09091 ) *
##              85) ShelveLoc: Medium 40  52.930 Yes ( 0.37500 0.62500 )
##               170) Price < 109.5 16   7.481 Yes ( 0.06250 0.93750 ) *
##               171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
##                 342) Age < 49.5 13  16.050 Yes ( 0.30769 0.69231 ) *
##                 343) Age > 49.5 11   6.702 No ( 0.90909 0.09091 ) *
##            43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##              86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##              87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##               174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
##                 348) CompPrice < 152.5 7   5.742 Yes ( 0.14286 0.85714 ) *
##                 349) CompPrice > 152.5 5   5.004 No ( 0.80000 0.20000 ) *
##               175) Price > 147 7   0.000 No ( 1.00000 0.00000 ) *
##        11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
##          22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
```

```
##             44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
##               88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##               89) Income > 100 5   0.000 Yes ( 0.00000 1.00000 ) *
##             45) CompPrice > 130.5 11   0.000 Yes ( 0.00000 1.00000 ) *
##           23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##             46) CompPrice < 122.5 10   0.000 No ( 1.00000 0.00000 ) *
##             47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##               94) Price < 125 5   0.000 Yes ( 0.00000 1.00000 ) *
##               95) Price > 125 5   0.000 No ( 1.00000 0.00000 ) *
##      3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
##        6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
##         12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
##           24) Price < 109 8   0.000 Yes ( 0.00000 1.00000 ) *
##           25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##         13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
##        7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
##         14) Income < 46 6   0.000 No ( 1.00000 0.00000 ) *
##         15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *
```

```
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train, ]
High.test <- High[-train]
tree.carseats <- tree(High ~ . - Sales, Carseats,
                      subset = train)
tree.pred <- predict(tree.carseats, Carseats.test,
                     type = "class")
table(tree.pred, High.test)
```

```
##            High.test
## tree.pred  No Yes
##       No  104  33
##       Yes  13  50
```

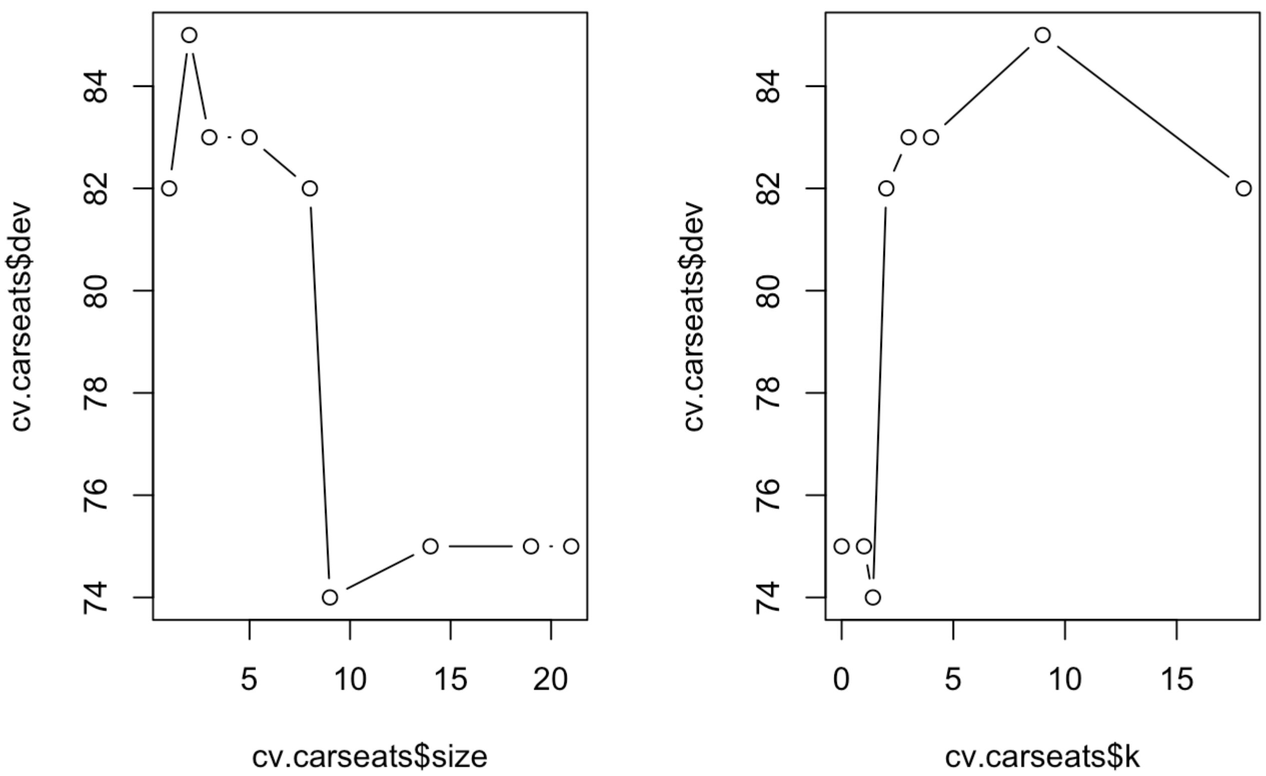
```
set.seed(7)
cv.carseats <- cv.tree(tree.carseats, FUN = prune.misclass)
names(cv.carseats)
```

```
## [1] "size"   "dev"    "k"      "method"
```

```
cv.carseats
```

```
## $size
## [1] 21 19 14  9  8  5  3  2  1
##
## $dev
## [1] 75 75 75 74 82 83 83 85 82
##
## $k
## [1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

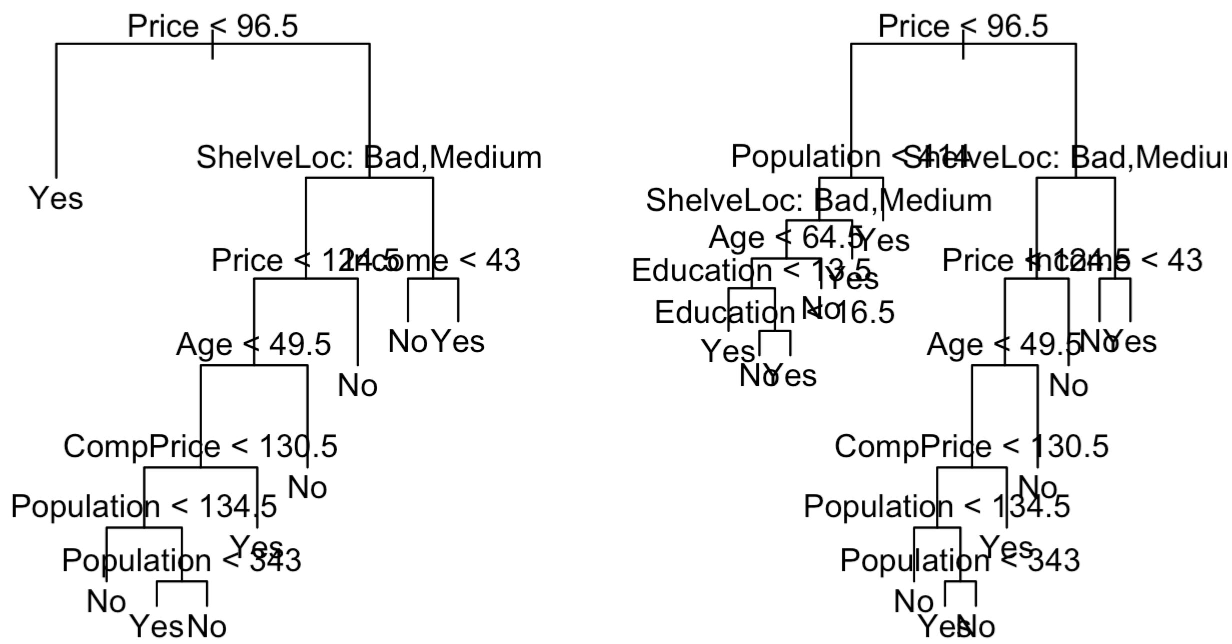
```
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```




```
prune.carseats <- prune.misclass(tree.carseats, best = 9)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
tree.pred <- predict(prune.carseats, Carseats.test,
                     type = "class")
table(tree.pred, High.test)
```

```
##          High.test
## tree.pred No Yes
##       No  97  25
##       Yes 20  58
```

```
prune.carseats <- prune.misclass(tree.carseats, best = 14)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```



```
tree.pred <- predict(prune.carseats, Carseats.test,
                     type = "class")
table(tree.pred, High.test)
```

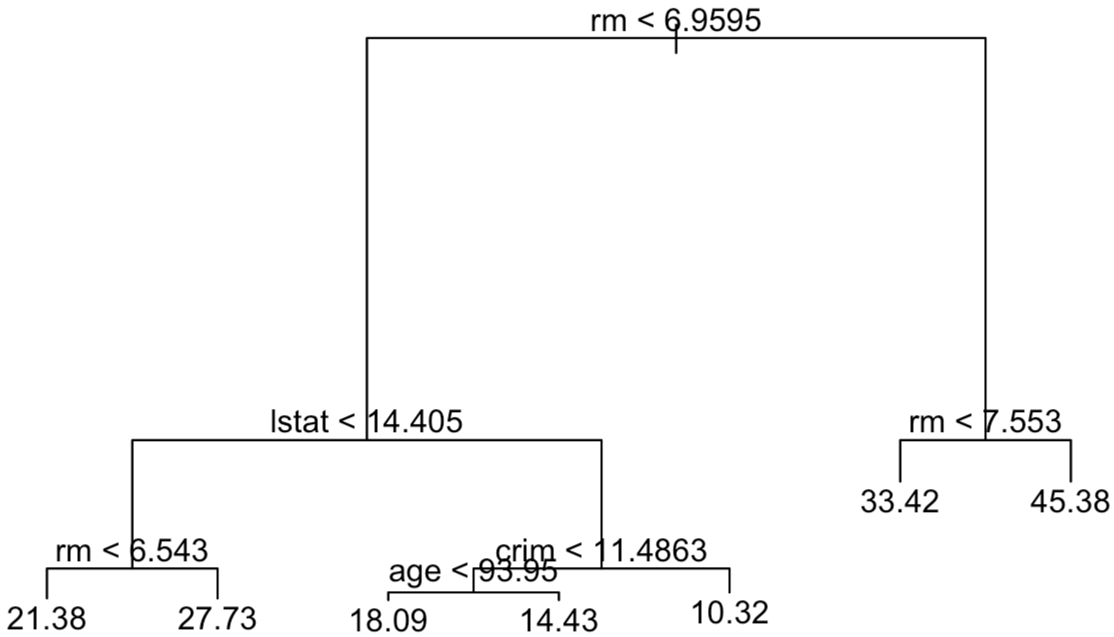
```
##          High.test
## tree.pred  No Yes
##       No  102  31
##       Yes  15  52
```

Fitting Regression Trees

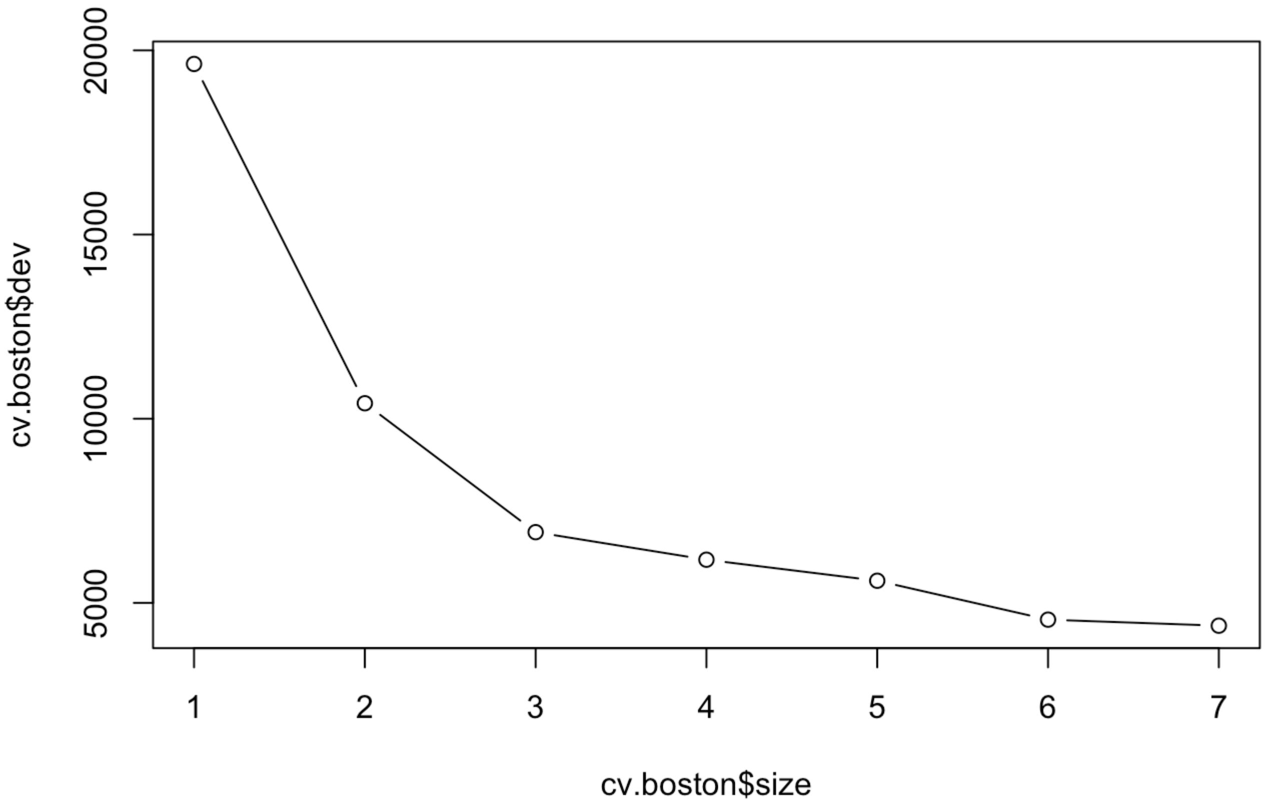
```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston) / 2)
tree.boston <- tree(medv ~ ., Boston, subset = train)
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

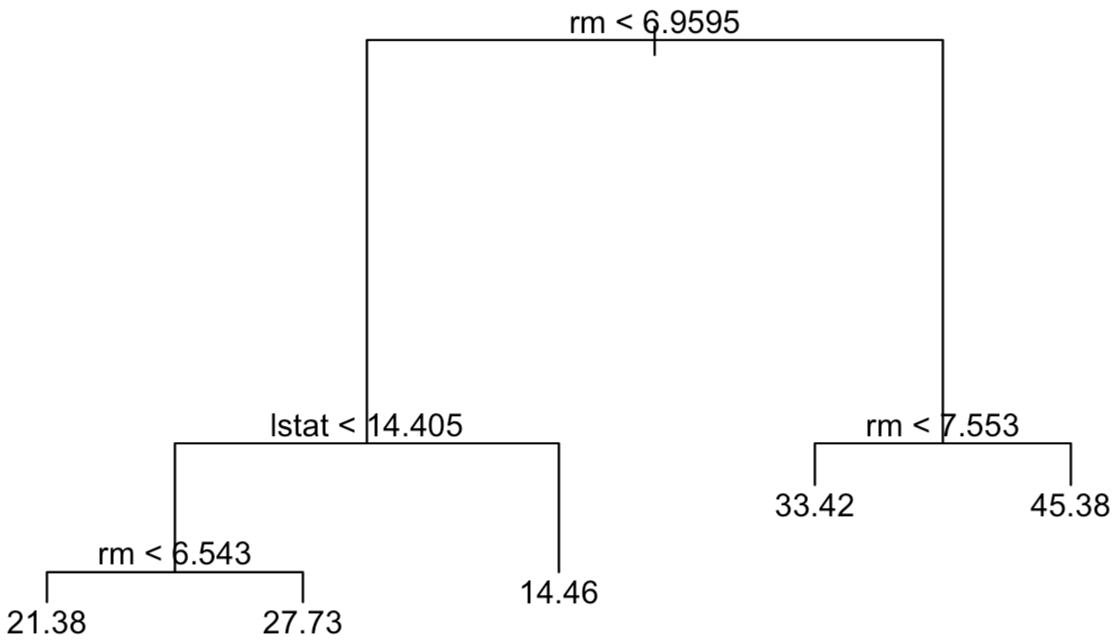
```
plot(tree.boston)
text(tree.boston, pretty = 0)
```



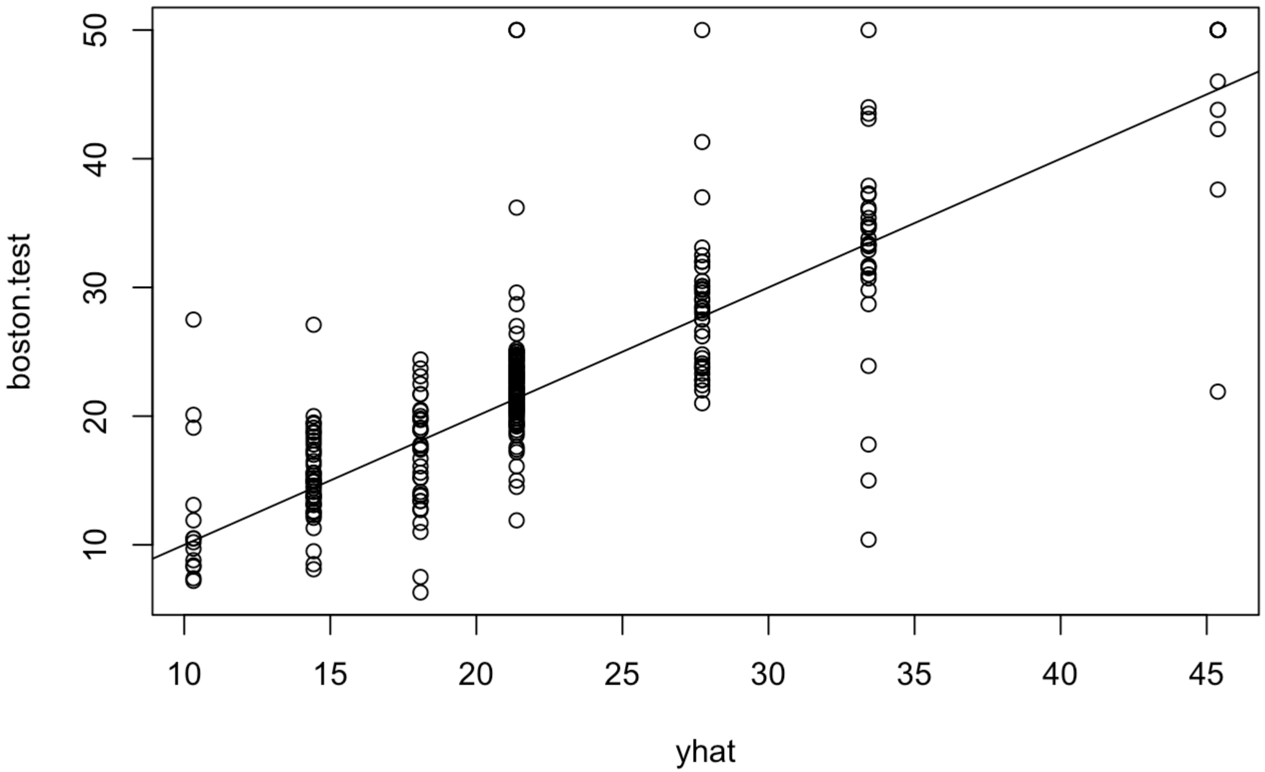

```
cv.boston <- cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type = "b")
```




```
prune.boston <- prune.tree(tree.boston, best = 5)
plot(prune.boston)
text(prune.boston, pretty = 0)
```




```
yhat <- predict(tree.boston, newdata = Boston[-train, ])
boston.test <- Boston[-train, "medv"]
plot(yhat, boston.test)
abline(0, 1)
```




```
mean((yhat - boston.test)^2)
```

```
## [1] 35.28688
```

Bagging and Random Forests

```
library(randomForest)
```

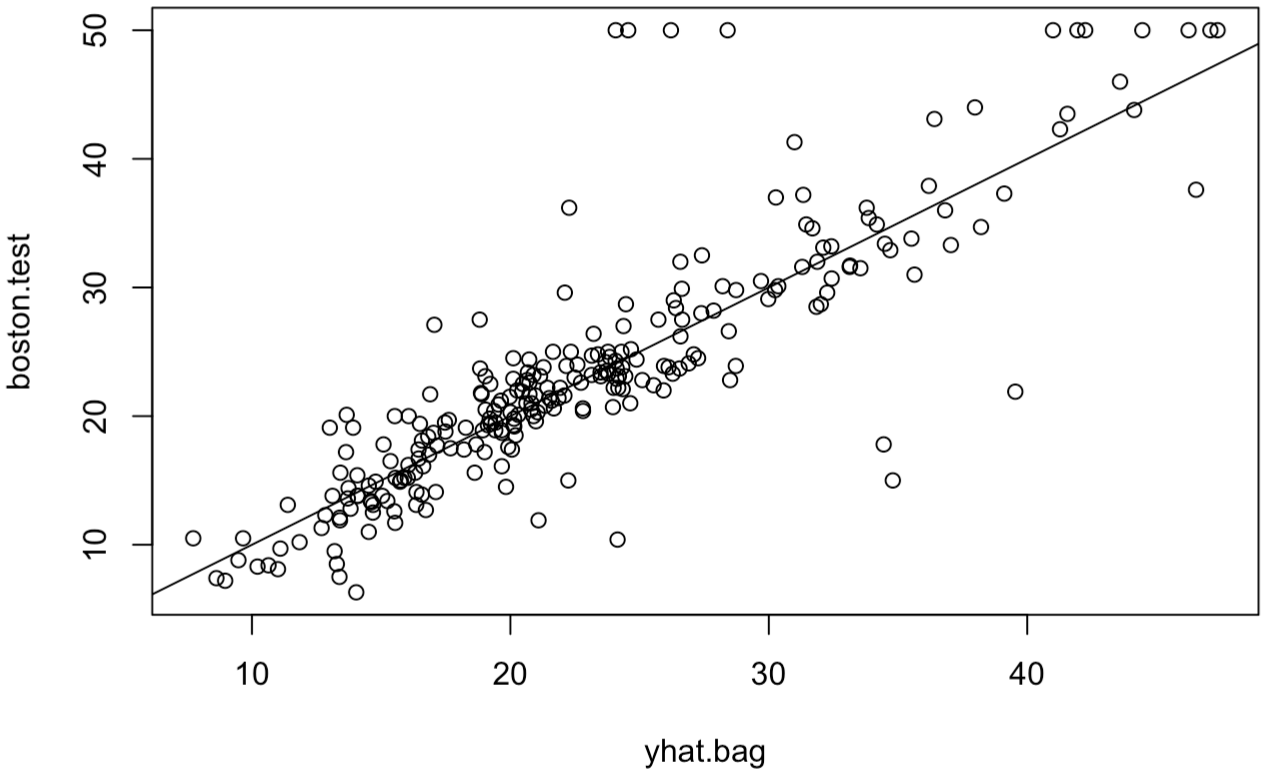
```
## randomForest 4.7-1.1
```

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

```
set.seed(1)
bag.boston <- randomForest(medv ~ ., data = Boston,
subset = train, mtry = 12, importance = TRUE)
bag.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 12, importance = TRUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 12
##
##           Mean of squared residuals: 11.40162
##                     % Var explained: 85.17
```

```
yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
plot(yhat.bag, boston.test)
abline(0, 1)
```




```
mean((yhat.bag - boston.test)^2)
```

```
## [1] 23.41916
```

```
bag.boston <- randomForest(medv ~ ., data = Boston,
subset = train, mtry = 12, ntree = 25)
yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
mean((yhat.bag - boston.test)^2)
```

```
## [1] 25.75055
```

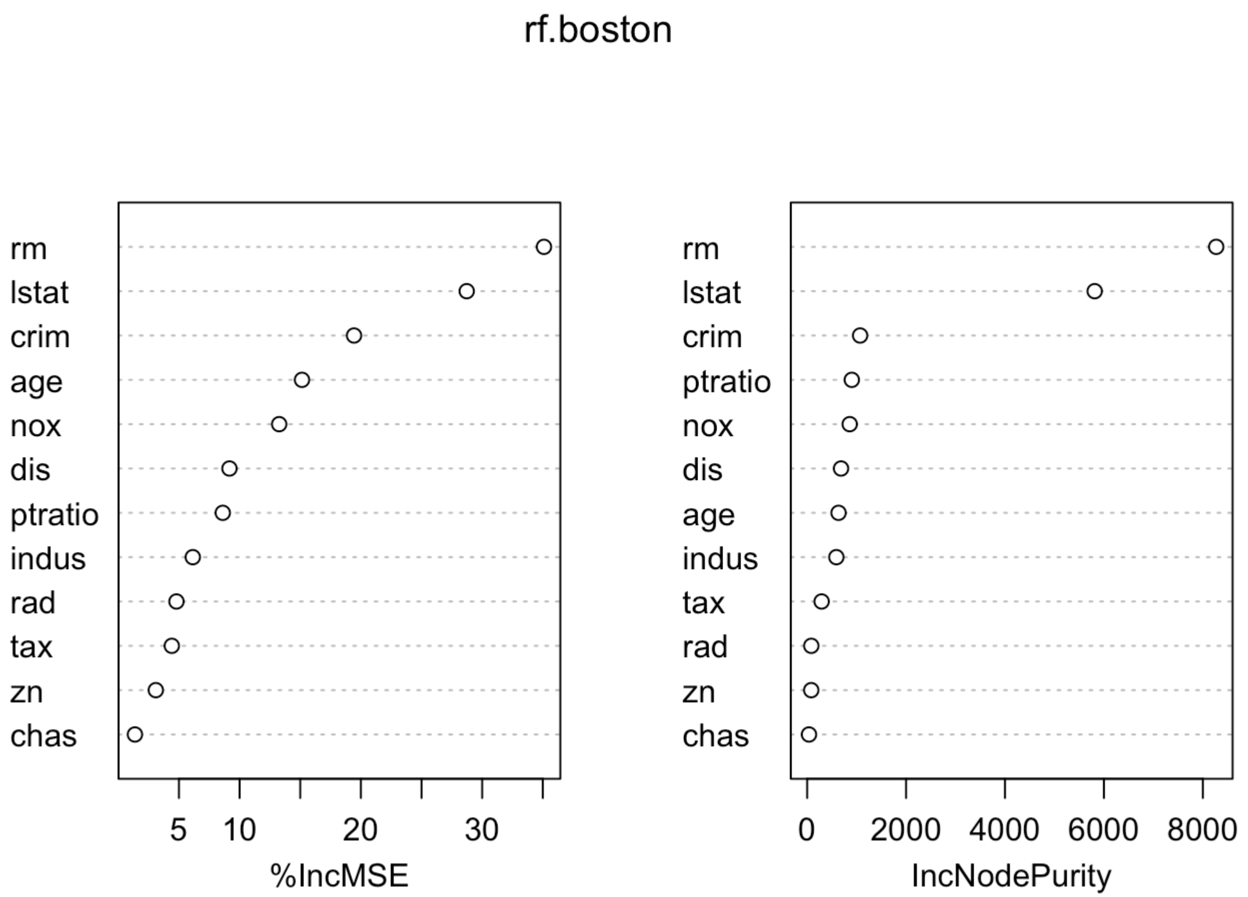
```
set.seed(1)
rf.boston <- randomForest(medv ~ ., data = Boston,
subset = train, mtry = 6, importance = TRUE)
yhat.rf <- predict(rf.boston, newdata = Boston[-train, ])
mean((yhat.rf - boston.test)^2)
```

```
## [1] 20.06644
```

```
importance(rf.boston)
```

```
##           %IncMSE IncNodePurity
## crim    19.435587    1070.42307
## zn       3.091630      82.19257
## indus    6.140529     590.09536
## chas     1.370310      36.70356
## nox     13.263466     859.97091
## rm      35.094741    8270.33906
## age     15.144821     634.31220
## dis      9.163776     684.87953
## rad      4.793720      83.18719
## tax      4.410714     292.20949
## ptratio  8.612780     902.20190
## lstat   28.725343    5813.04833
```

```
varImpPlot(rf.boston)
```



Boosting

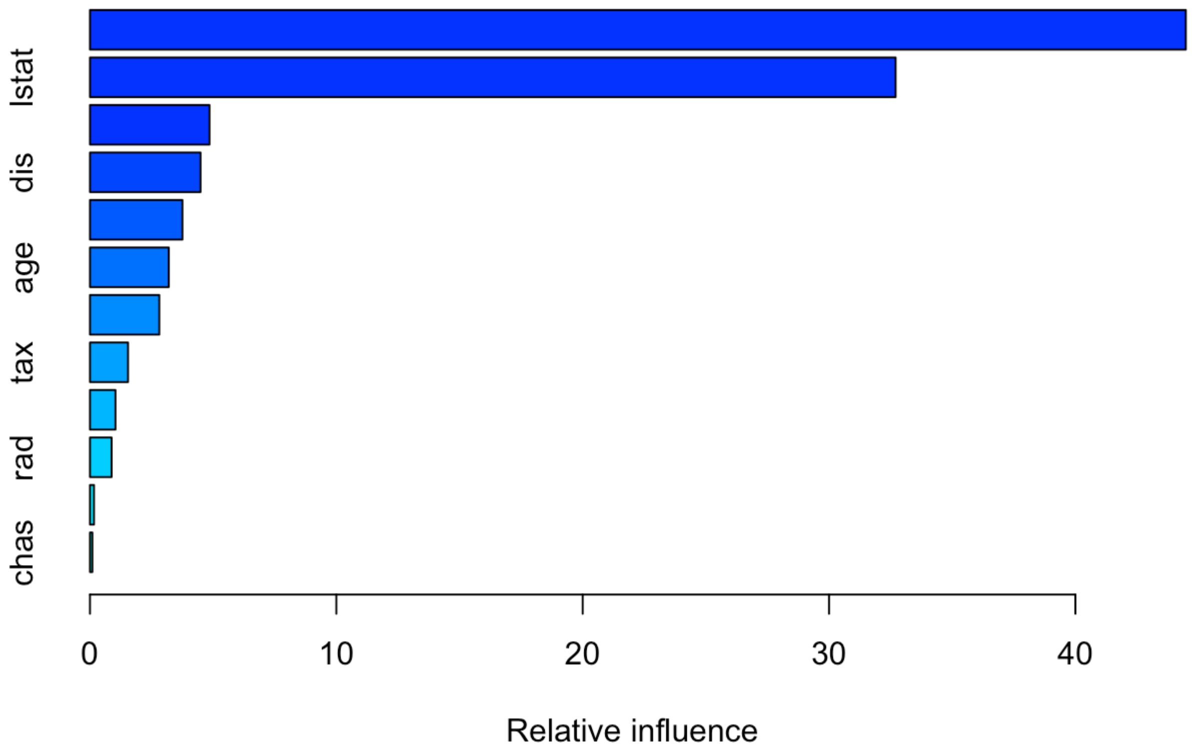
```r
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.3.3
```

```
## Loaded gbm 2.2.2
```

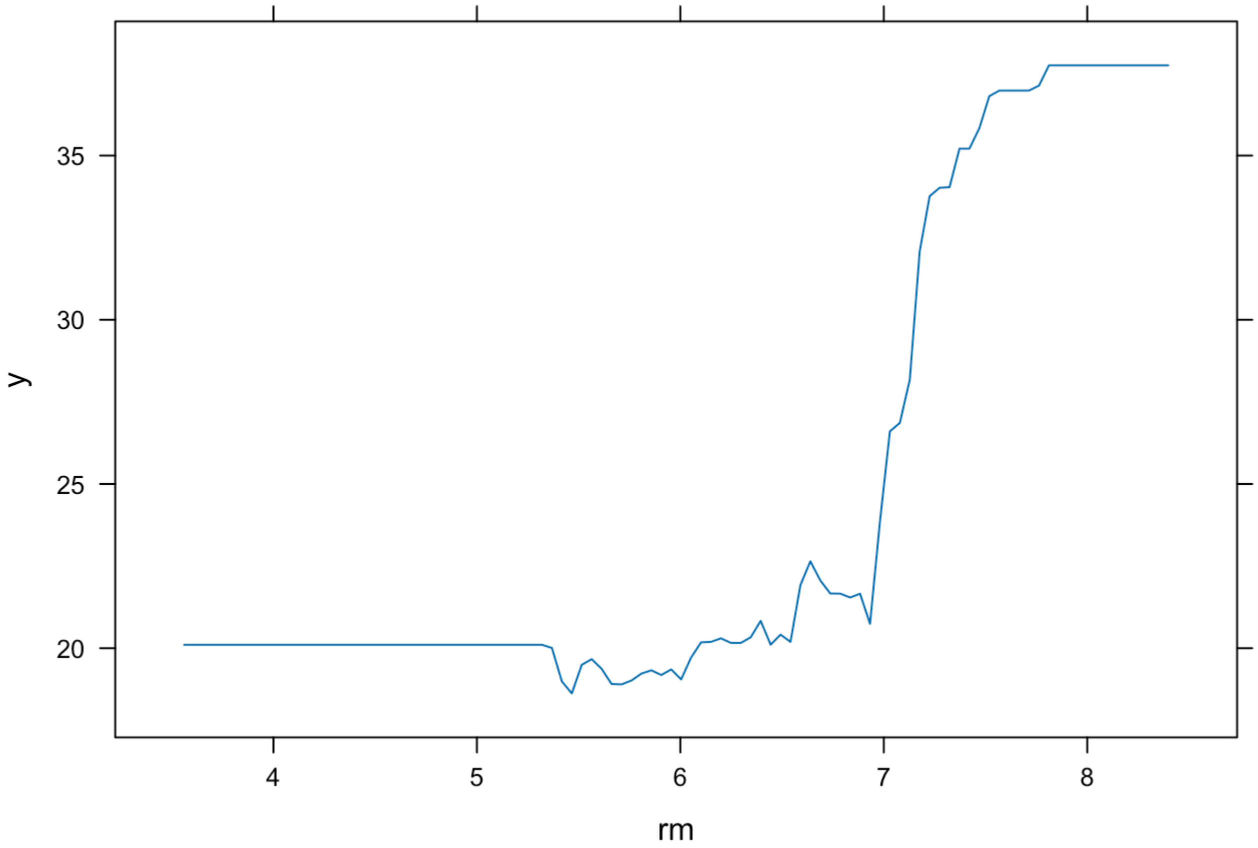
```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.com/gbm-dev
elopers/gbm3
```

```
set.seed(1)
boost.boston <- gbm(medv ~ ., data = Boston[train, ],
distribution = "gaussian", n.trees = 5000,
interaction.depth = 4)
summary(boost.boston)
```

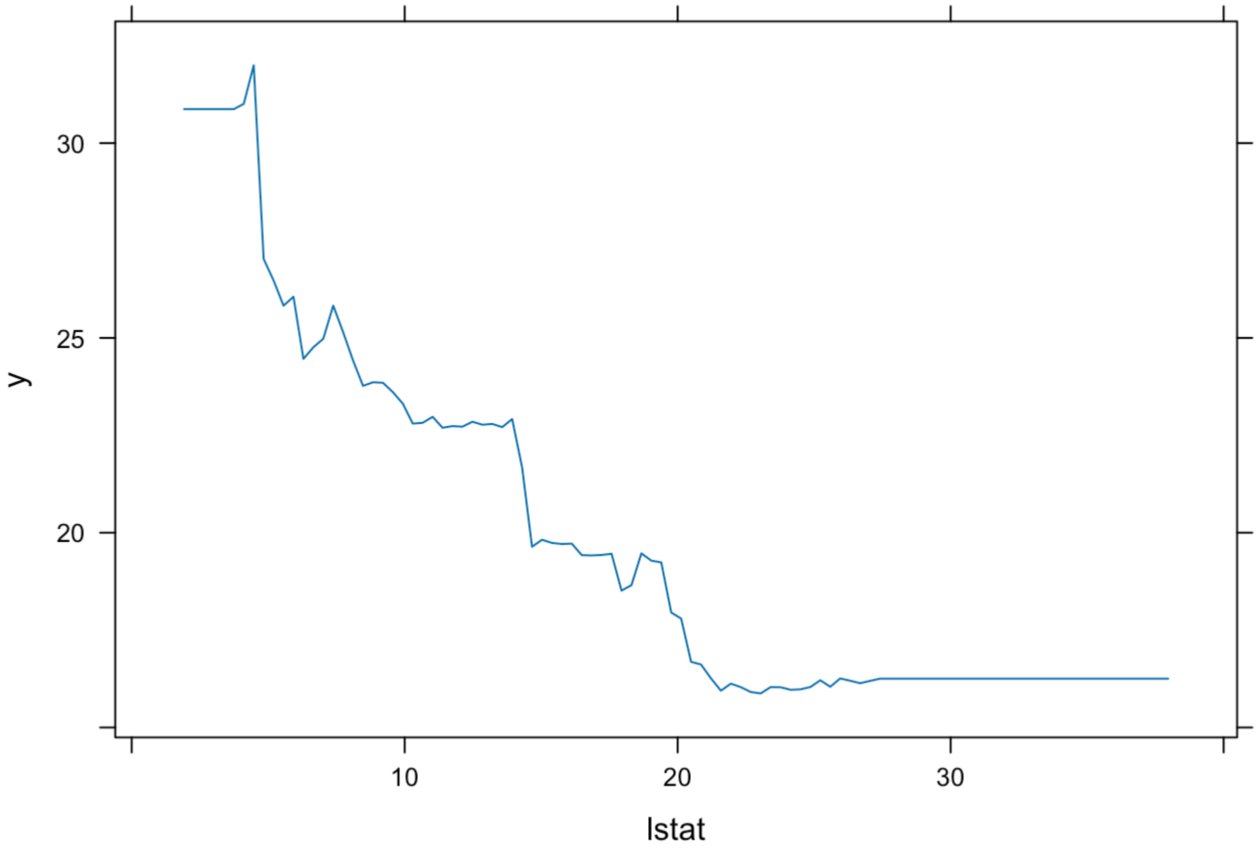


```
##                 var     rel.inf
## rm               rm 44.48249588
## lstat         lstat 32.70281223
## crim           crim  4.85109954
## dis             dis  4.48693083
## nox             nox  3.75222394
## age             age  3.19769210
## ptratio ptratio  2.81354826
## tax             tax  1.54417603
## indus         indus  1.03384666
## rad             rad  0.87625748
## zn               zn  0.16220479
## chas           chas  0.09671228
```

```
plot(boost.boston, i = "rm")
```




```
plot(boost.boston, i = "lstat")
```




```
yhat.boost <- predict(boost.boston,
newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
## [1] 18.39057
```

```
boost.boston <- gbm(medv ~ ., data = Boston[train, ],
distribution = "gaussian", n.trees = 5000,
interaction.depth = 4, shrinkage = 0.2, verbose = F)
yhat.boost <- predict(boost.boston,
newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
## [1] 16.54778
```

Bayesian Additive Regression Trees

```
library(BART)
```

```
## Warning: package 'BART' was built under R version 4.3.3
```

```
## Loading required package: nlme
```

```
## Loading required package: survival
```

```
x <- Boston[, 1:12]
y <- Boston[, "medv"]
xtrain <- x[train, ]
ytrain <- y[train]
xtest <- x[-train, ]
ytest <- y[-train]
set.seed(1)
bartfit <- gbart(xtrain, ytrain, x.test = xtest)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 253, 12, 253
## y1,yn: 0.213439, -5.486561
## x1,x[n*p]: 0.109590, 20.080000
## xp1,xp[np*p]: 0.027310, 7.880000
## *****Number of Trees: 200
## *****Number of Cut Points: 100 ... 100
## *****burn,nd,thin: 100,1000,1
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.795495,3,3.71636,21.7866
## *****sigma: 4.367914
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,12,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 2s
## trcnt,tecnt: 1000,1000
```

```
yhat.bart <- bartfit$yhat.test.mean
mean((ytest - yhat.bart)^2)
```

```
## [1] 15.91912
```

```
ord <- order(bartfit$varcount.mean, decreasing = T)
bartfit$varcount.mean[ord]
```

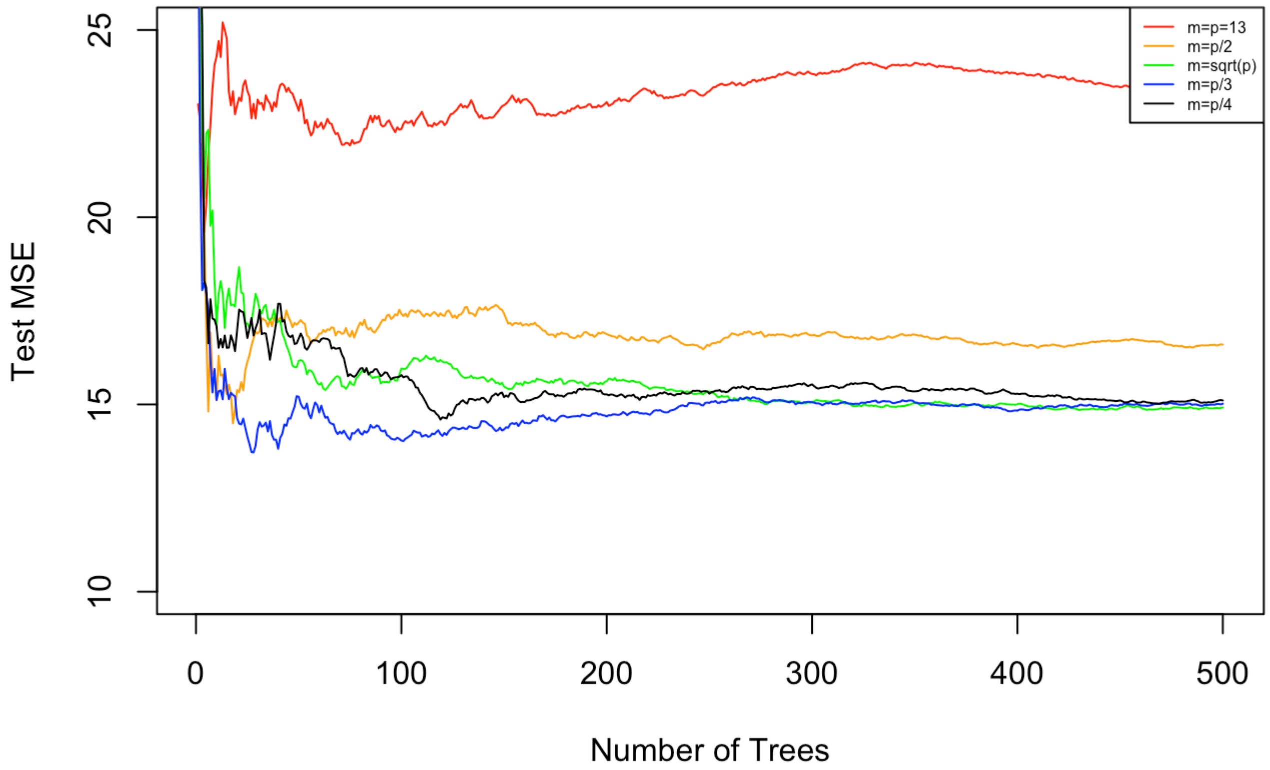
```
##     nox   lstat     rad      rm     tax ptratio    chas     age   indus      zn
##  22.973  21.653  21.638  20.725  20.021  19.615  19.283  19.278  19.073  15.576
##     dis    crim
##  13.800  11.607
```

```
library(tree)
library(randomForest)
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```
set.seed(1)
subset<-sample(1:nrow(Boston),nrow(Boston)*0.7)
Boston.train<-Boston[subset,-14]
Boston.test<-Boston[-subset,-14]
y.train<-Boston[subset,14]
y.test<-Boston[-subset,14]
rfmodel1<-randomForest(Boston.train,y=y.train,xtest = Boston.test,ytest = y.test,ntree=500,mtry=ncol(Boston.trai
n))
rfmodel2<-randomForest(Boston.train,y.train,xtest = Boston.test,ytest = y.test,ntree=500,mtry=(ncol(Boston.trai
n))/2)
rfmodel3<-randomForest(Boston.train,y.train,xtest = Boston.test,ytest = y.test,ntree=500,mtry=(ncol(Boston.trai
n))^(0.5))
rfmodel4<-randomForest(Boston.train,y.train,xtest = Boston.test,ytest = y.test,ntree=500,mtry=(ncol(Boston.trai
n))/3)
rfmodel5<-randomForest(Boston.train,y.train,xtest = Boston.test,ytest = y.test,ntree=500,mtry=(ncol(Boston.trai
n))/4)
plot(1:500,rfmodel1$test$mse,col="red",type="l",xlab = "Number of Trees",ylab = "Test MSE",ylim = c(10,25))
lines(1:500,rfmodel2$test$mse, col="orange",type="l")
lines(1:500,rfmodel3$test$mse, col="green",type="l")
lines(1:500,rfmodel4$test$mse, col="blue",type="l")
lines(1:500,rfmodel5$test$mse, col="black",type="l")
legend("topright",c("m=p=13","m=p/2","m=sqrt(p)","m=p/3","m=p/4"),col=c("red","orange","green","blue","black"),ce
x=0.5,lty=1)
```



We see that Test MSE decreases with the increase in number of trees. It stabilizes after certain number of trees and no further improvement is seen.

```
library(ISLR)
```

```
##
## Attaching package: 'ISLR'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     Carseats
```

```
## The following objects are masked from 'package:ISLR2':
##
##     Auto, Credit
```

```
attach(Carseats)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     High
```

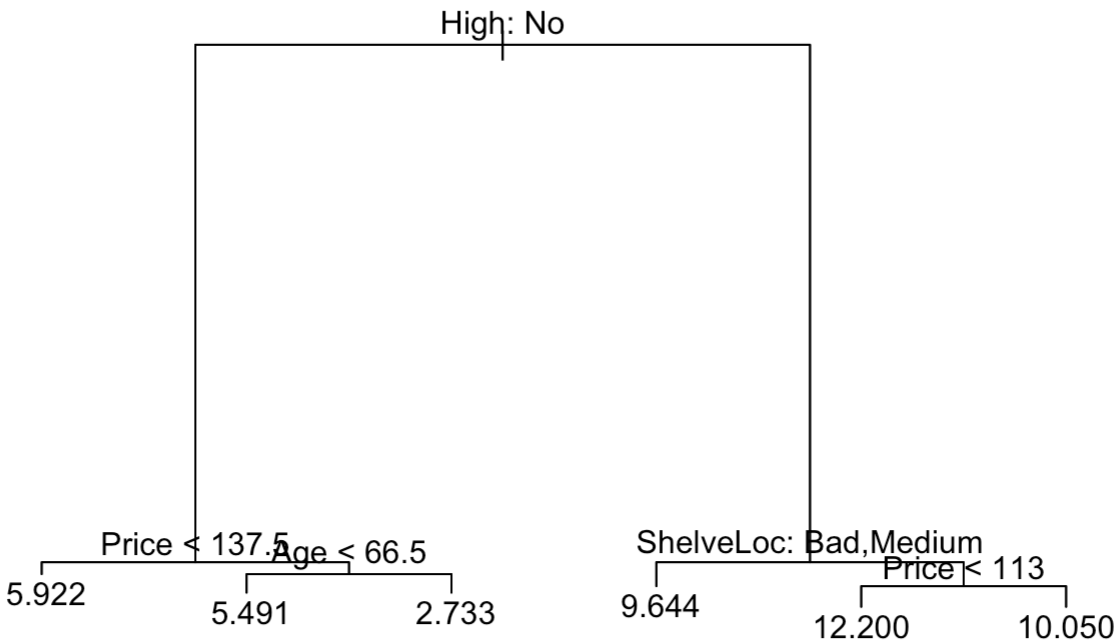
```
## The following objects are masked from Carseats (pos = 10):
##
##     Advertising, Age, CompPrice, Education, Income, Population, Price,
##     Sales, ShelveLoc, Urban, US
```

```
set.seed(1)
subset<-sample(nrow(Carseats),nrow(Carseats)*0.7)
car.train<-Carseats[subset,]
car.test<-Carseats[-subset,]
```

```
library(tree)
car.model.train<-tree(Sales~.,car.train)
summary(car.model.train)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = car.train)
## Variables actually used in tree construction:
## [1] "High"      "Price"     "Age"       "ShelveLoc"
## Number of terminal nodes:  6
## Residual mean deviance:  2.081 = 570.2 / 274
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -4.502  -0.964  -0.037   0.000   1.048   3.716
```

```
plot(car.model.train)
text(car.model.train,pretty=0)
```

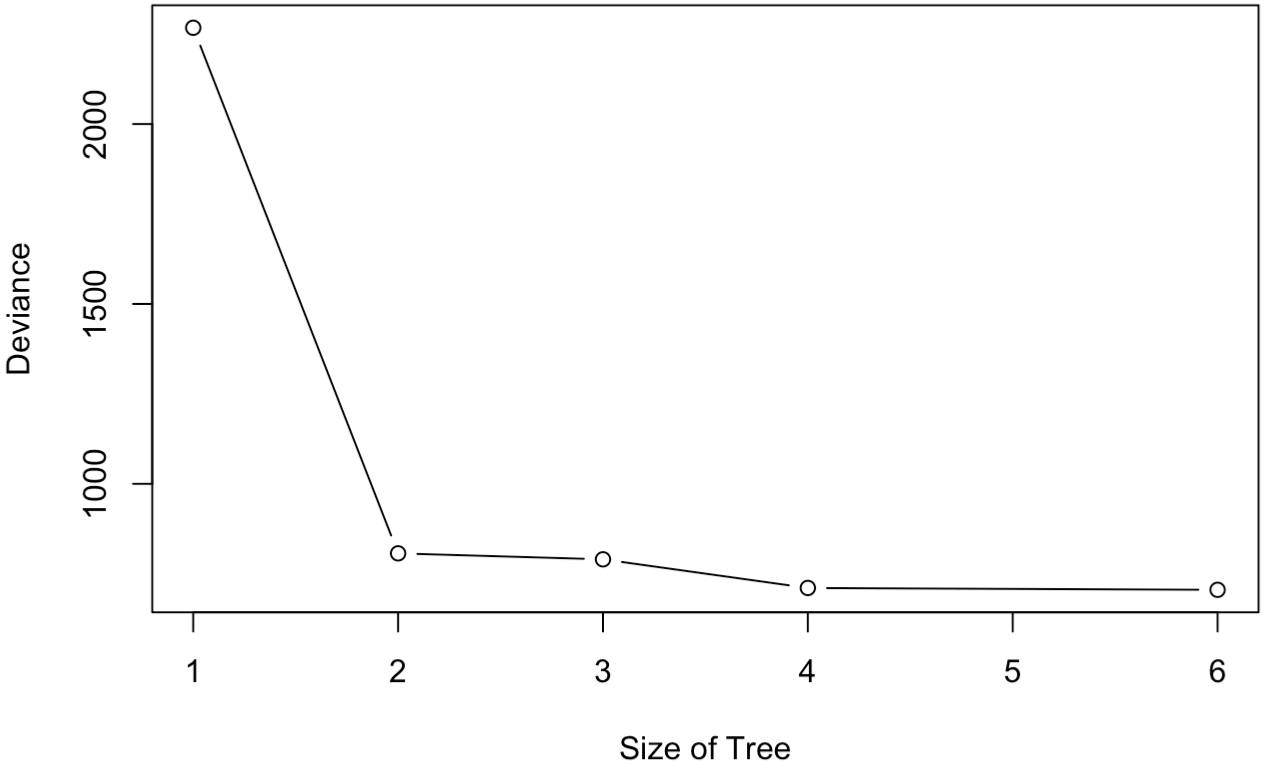



```
tree.prediction<-predict(car.model.train,newdata=car.test)
tree.mse<-mean((car.test$Sales-tree.prediction)^2)
tree.mse
```

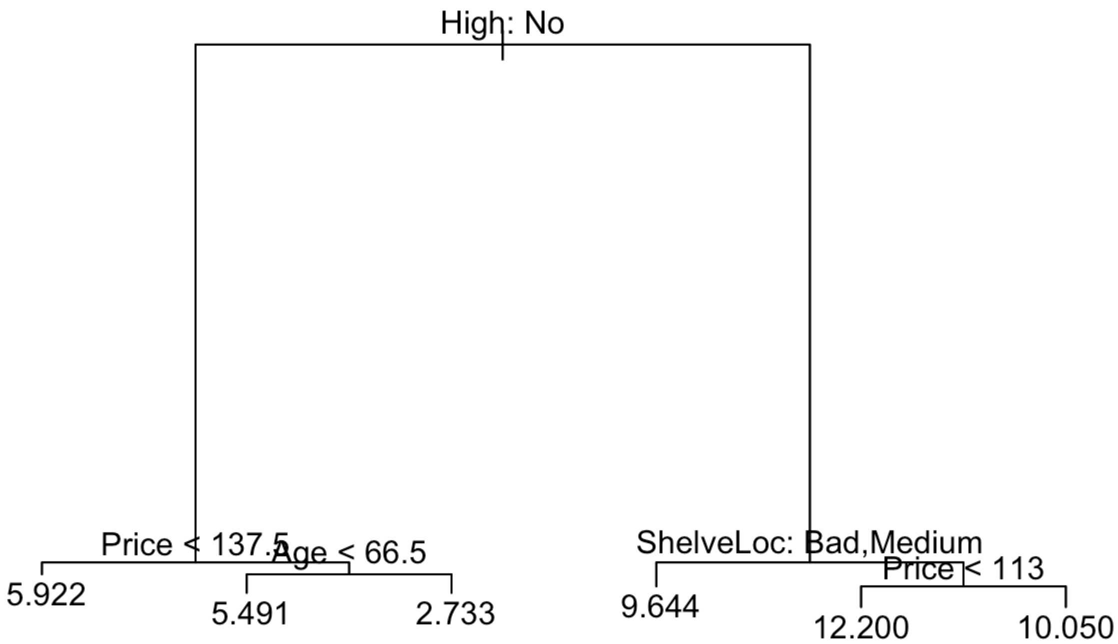
```
## [1] 2.98086
```

The Test MSE for full grown Tree is recorded as 5.288

```
set.seed(1)
cv.car<-cv.tree(car.model.train)
plot(cv.car$size,cv.car$dev,xlab = "Size of Tree",ylab = "Deviance",type = "b")
```




```
prune.car<-prune.tree(car.model.train,best=6)
plot(prune.car)
text(prune.car,pretty=0)
```




```
prune.predict<-predict(prune.car,car.test)
mean((prune.predict-car.test$Sales)^2)
```

```
## [1] 2.98086
```

For the pruned tree we get MSE as 5.454

```
bag.car<-randomForest(Sales~.,car.train,importance=TRUE,mtry=13)
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range
```

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

```
##                 %IncMSE IncNodePurity
## CompPrice    16.9808558     94.501792
## Income       -2.3357235     58.429280
## Advertising   8.0939012     45.420033
## Population   -4.0682805     54.279255
## Price        36.3535209    220.436675
## ShelveLoc    33.8350571    125.105257
## Age          11.4439739    100.602908
## Education    -0.3099822     36.484335
## Urban        -1.3278151      8.833596
## US            1.4225721      6.988115
## High        109.4175231   1470.147822
```

```
bag.car.predict<-predict(bag.car,car.test)
mean((bag.car.predict-car.test$Sales)^2)
```

```
## [1] 2.041283
```

We use randomForest with m=p=13 total number of predictors which is equivalent to bagging,The Test Set MSE obtained is 2.324. It has further reduced compared to single pruned tree.Thus Bagging helped reducing the MSE,We can see that Price & ShelvLoc are the two most important variables chosen by Bagging model

```
rf.car<-randomForest(Sales~.,car.train,importance=TRUE,mtry=sqrt(13))
importance(rf.car)
```

```
##                %IncMSE IncNodePurity
## CompPrice   10.4245307     109.82698
## Income      -2.1469464      91.34584
## Advertising  5.8209621      77.60837
## Population  -2.7845501      72.96286
## Price       23.3821519     310.36233
## ShelveLoc   22.8678599     284.92725
## Age          6.0439606     138.37473
## Education   -0.5646153      55.22101
## Urban       -4.1408498      13.12375
## US           3.2582924      15.01993
## High        69.2591486    1020.45181
```

```
rf.car.predict<-predict(rf.car,car.test)
mean((rf.car.predict-car.test$Sales)^2)
```

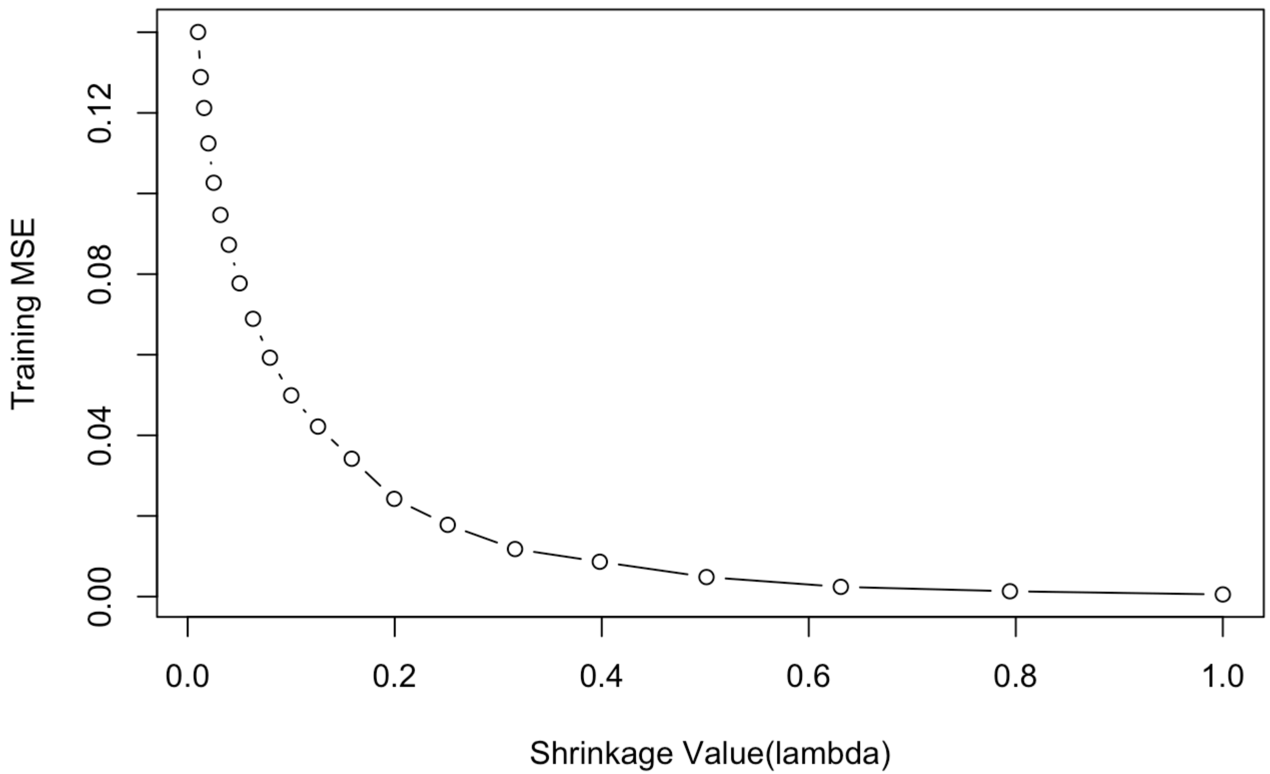
```
## [1] 2.041314
```

Using Random Forest the MSE increased compared to bagging,The important variables chosen by Random Forest are the same as the one chosen by Bagging.Random Forest avoids correlated trees and hence is expected to perform better than Bagging. Here the case is not true.Further tuning of Random Forest model should be tried:Full Grown Tree MSE: 5.288;Pruned Tree MSE: 5.454;Bagging Model MSE: 2.324;Random Forest MSE: 2.518

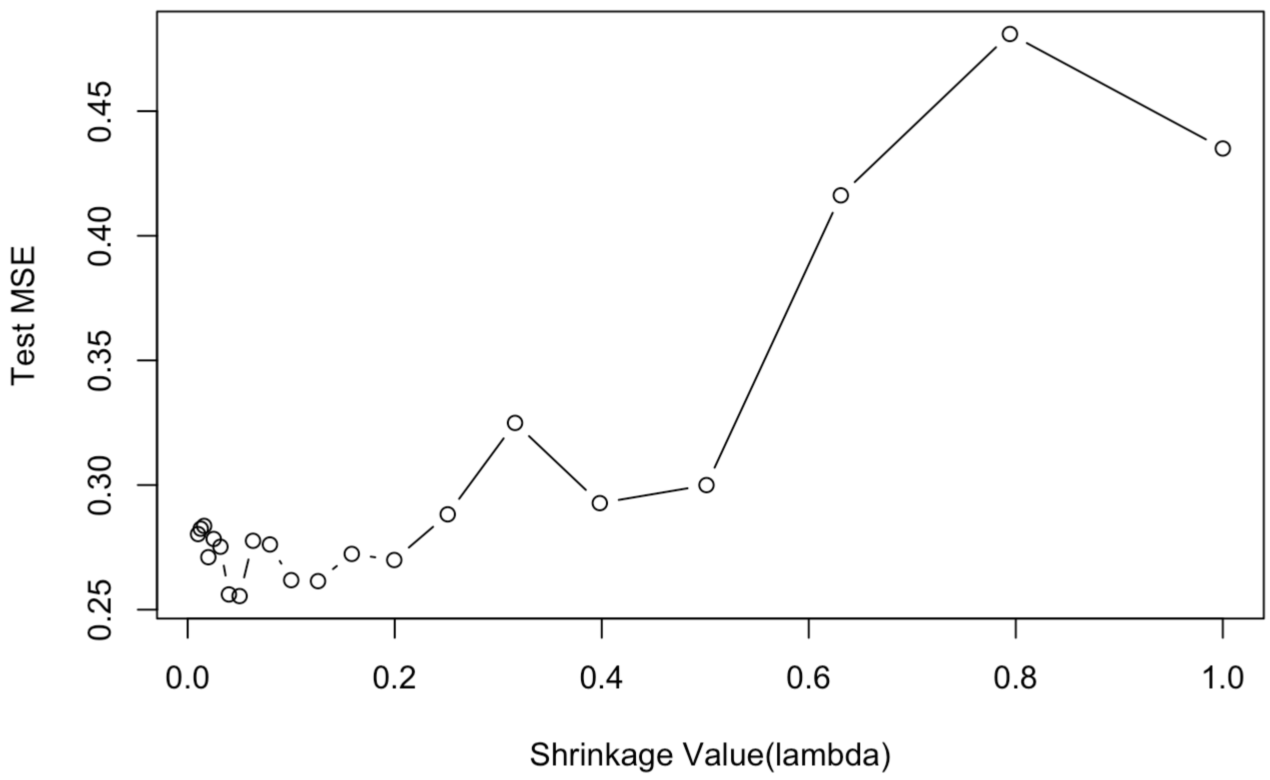
```
attach(Hitters)
Hitters<-na.omit(Hitters)
Hitters$Salary<-log(Hitters$Salary)
```

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

```
library(gbm)
set.seed(1)
powerss<-seq(-2,0,by=0.1)
lambdas<-10^powerss
train.error<-rep(NA,length(lambdas))
for (i in 1:length(lambdas)){
hitters.gbm<-gbm(Salary~.,hitters.train,distribution = "gaussian",n.trees=1000,shrinkage=lambdas[i])
hitters.train.pred<-predict(hitters.gbm,hitters.train,n.trees=1000)
train.error[i]<-mean((hitters.train.pred-hitters.train$Salary)^2)
}
plot(lambdas,train.error,type="b",xlab="Shrinkage Value(lambda)",ylab="Training MSE")
```



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



```
hitters.gbm.testerror<-min(test.error)
hitters.gbm.testerror
```

```
## [1] 0.255455
```

The Minimum Test MSE obtained by boosting is 0.26.

```
library(glmnet)
```

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

```
## Loaded glmnet 4.1-8
```

```
lm<-lm(Salary~.,hitters.train)
hitters.predict.lm<-predict(lm,hitters.test)
hitters.lm.test.mse<-mean((hitters.predict.lm-hitters.test$Salary)^2)
hitters.lm.test.mse
```

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

```
x<-model.matrix(Salary~.,hitters.train)
x.test<-model.matrix(Salary ~ . , hitters.test)
y<-hitters.train$Salary
hitters.ridge<-glmnet(x,y,alpha=0)
hitters.ridge.predict<-predict(hitters.ridge,s=0.01,x.test)
hitters.ridge.test.mse<-mean((hitters.ridge.predict-hitters.test$Salary)^2)
hitters.ridge.test.mse
```

```
## [1] 0.4570283
```

```
x<-model.matrix(Salary~.,hitters.train)
x.test<-model.matrix(Salary ~ . , hitters.test)
y<-hitters.train$Salary
hitters.lasso<-glmnet(x,y,alpha=1)
hitters.lasso.predict<-predict(hitters.lasso,s=0.01,x.test)
hitters.lasso.test.mse<-mean((hitters.lasso.predict-hitters.test$Salary)^2)
hitters.lasso.test.mse
```

```
## [1] 0.4700537
```

We have Test MSE for different methods as summarized below. It can be seen Boosting gives least Test MSE among the 4 models

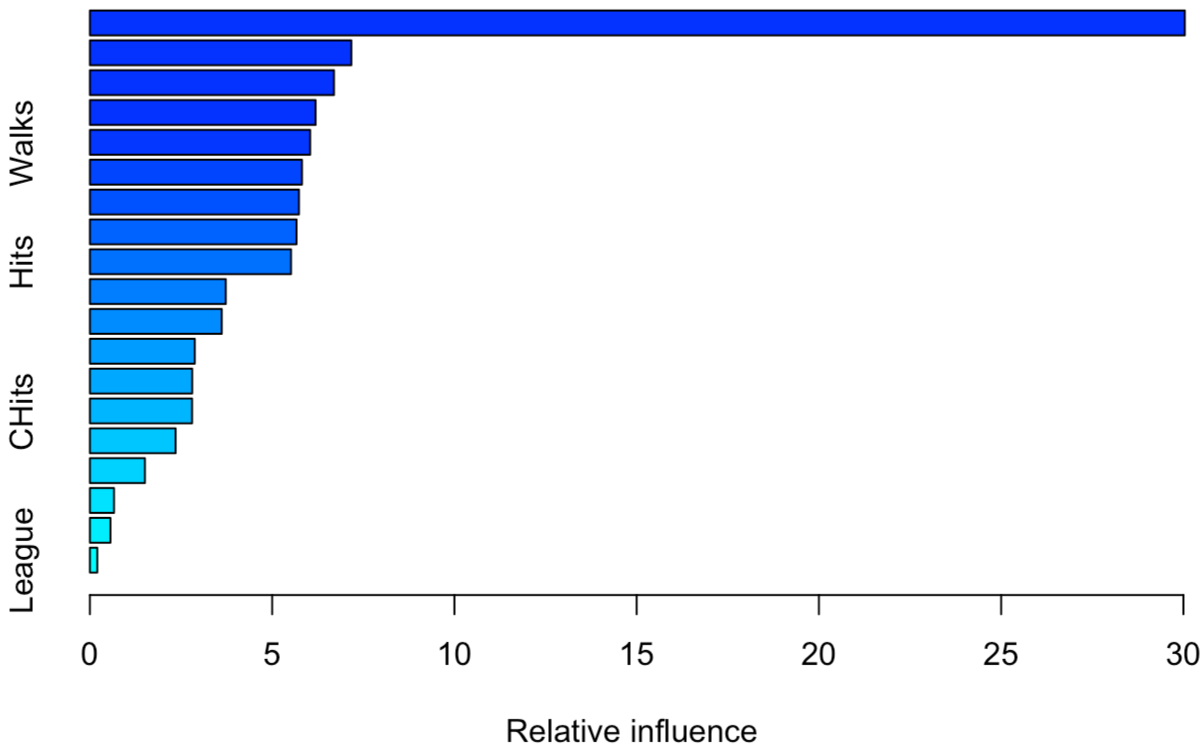
Least Square Regression Full Model Test MSE: 0.49

Ridge Regression Model Test MSE: 0.46

Lasso Regression Model Test MSE: 0.47

```
boost.hitters<-gbm(Salary~.,data=hitters.train,distribution = "gaussian",n.trees = 1000,shrinkage=lambdas[which.m
in(test.error)])

summary(boost.hitters)
```


Walks
Hits
CHits
League
0
5
10
15
20
25
30
Relative influence

```
##                 var    rel.inf
## CAtBat       CAtBat 30.0391455
## Years         Years  7.1722320
## CWalks       CWalks  6.6962390
## PutOuts     PutOuts  6.1919523
## Walks         Walks  6.0430398
## CRuns         CRuns  5.8184446
## CHmRun       CHmRun  5.7355580
## CRBI           CRBI  5.6678930
## Hits           Hits  5.5180489
## HmRun         HmRun  3.7274075
## Assists     Assists  3.6165621
## AtBat         AtBat  2.8770937
## RBI             RBI  2.8062318
## CHits         CHits  2.8030774
## Errors       Errors  2.3509666
## Runs           Runs  1.5093746
## Division   Division  0.6614964
## NewLeague NewLeague  0.5632541
## League       League  0.2019828
```

We find that CAtbat is the most important variable

```
set.seed(1)
hitters.bagging<-randomForest(Salary~.,hitters.train,mtry=19,importance=TRUE)
hitters.bagg.predict<-predict(hitters.bagging,hitters.test)
hitters.bagg.test.mse<-mean((hitters.bagg.predict-hitters.test$Salary)^2)
hitters.bagg.test.mse
```

```
## [1] 0.2301184
```

The Test set MSE for Bagging is 0.23 .This is lower than the Test set MSE obtained for Boosting which was 0.26