stsci4740hw5

Nick Gembs

11/26/2022

p = 50  
N = 100  
set.seed(1)  
X\_train = array(rnorm(p\*N),c(N,p))  
eps\_train = rnorm(N)  
Nte = 10^3  
X\_te = array(rnorm(p\*Nte),c(Nte,p))  
eps\_te = rnorm(Nte)  
grid = 10^seq(10,-2,length = 100)

# Ridge

df = as.data.frame(X\_train)  
#sample(df)  
  
Y\_train = 2\*df$V1 + 2 \* df$V2 + 2\*df$V3 +2\*df$V4 + 2\*df$V5 + eps\_train  
  
  
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.2.2

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 4.2.2

## Loaded glmnet 4.1-4

#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(X\_train, Y\_train, alpha = 0, K=10)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
best\_lambda

## [1] 0.2927512

min(cv\_model$cvm)

## [1] 1.647019

df = as.data.frame(X\_te)  
#sample(df)  
  
Y\_test = 2\*df$V1 + 2 \* df$V2 + 2\*df$V3 +2\*df$V4 + 2\*df$V5 + eps\_train  
  
Y\_pred = predict(cv\_model, newx = X\_te, s = best\_lambda )  
MSE\_ridge = mean((Y\_test-Y\_pred)^2)  
MSE\_ridge

## [1] 2.387636

# Lasso

df = as.data.frame(X\_train)  
#sample(df)  
  
Y\_train = 2\*df$V1 + 2 \* df$V2 + 2\*df$V3 +2\*df$V4 + 2\*df$V5 + eps\_train  
  
  
library(glmnet)  
  
#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(X\_train, Y\_train, alpha = 1, K=10)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
best\_lambda

## [1] 0.03066908

min(cv\_model$cvm)

## [1] 1.231167

df = as.data.frame(X\_te)  
#sample(df)  
  
Y\_test = 2\*df$V1 + 2 \* df$V2 + 2\*df$V3 +2\*df$V4 + 2\*df$V5 + eps\_train  
  
Y\_pred = predict(cv\_model, newx = X\_te, s = best\_lambda )  
MSE\_lasso = mean((Y\_test-Y\_pred)^2)  
MSE\_lasso

## [1] 1.625679

# Test MSE for Lasso is less than ridge in a sparse model

2

# Ridge

df = as.data.frame(X\_train)  
#sample(df)  
  
Y\_train = eps\_train  
  
for (i in df) {  
 Y\_train = Y\_train + .5\*i  
}  
  
library(glmnet)  
  
#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(X\_train, Y\_train, alpha = 0, K=10)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
best\_lambda

## [1] 0.1093254

min(cv\_model$cvm)

## [1] 1.510946

df = as.data.frame(X\_te)  
#sample(df)  
  
Y\_test = eps\_te  
  
for (i in df) {  
 Y\_test= Y\_test + .5\*i  
}  
  
Y\_pred = predict(cv\_model, newx = X\_te, s = best\_lambda )  
MSE\_ridge = mean((Y\_test-Y\_pred)^2)  
MSE\_ridge

## [1] 2.006925

# Lasso

df = as.data.frame(X\_train)  
#sample(df)  
  
Y\_train = eps\_train  
  
for (i in df) {  
 Y\_train = Y\_train + .5\*i  
}  
  
library(glmnet)  
  
#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(X\_train, Y\_train, alpha = 1, K=10)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
best\_lambda

## [1] 0.002584987

min(cv\_model$cvm)

## [1] 2.003724

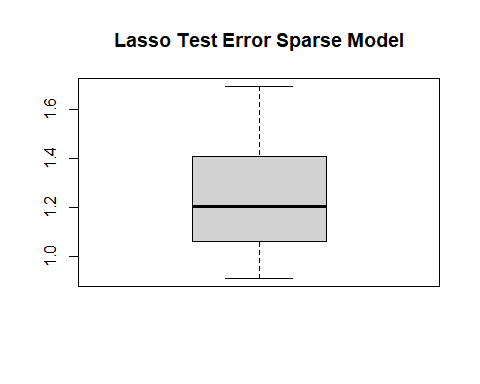
df = as.data.frame(X\_te)  
#sample(df)  
  
Y\_test = eps\_te  
  
for (i in df) {  
 Y\_test= Y\_test + .5\*i  
}  
  
Y\_pred = predict(cv\_model, newx = X\_te, s = best\_lambda )  
MSE\_lasso = mean((Y\_test-Y\_pred)^2)  
MSE\_lasso

## [1] 2.391959

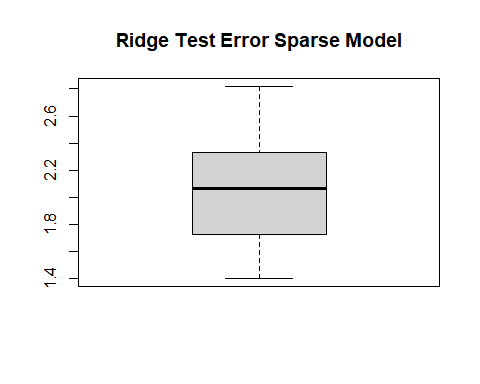
# For non-sparse models, ridge has a lower test MSE than lasso.

test\_errors\_ridge = rep(0,50)  
  
for ( i in 1:50){  
 set.seed(i)  
 X\_train = array(rnorm(p\*N),c(N,p))  
eps\_train = rnorm(N)  
Nte = 10^3  
X\_te = array(rnorm(p\*Nte),c(Nte,p))  
eps\_te = rnorm(Nte)  
grid = 10^seq(10,-2,length = 100)  
  
df = as.data.frame(X\_train)  
#sample(df)  
  
Y\_train = 2\*df$V1 + 2 \* df$V2 + 2\*df$V3 +2\*df$V4 + 2\*df$V5 + eps\_train  
  
  
#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(X\_train, Y\_train, alpha = 0, K=10)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
  
df = as.data.frame(X\_te)  
#sample(df)  
  
Y\_test = 2\*df$V1 + 2 \* df$V2 + 2\*df$V3 +2\*df$V4 + 2\*df$V5 + eps\_train  
  
Y\_pred = predict(cv\_model, newx = X\_te, s = best\_lambda )  
MSE\_ridge = mean((Y\_test-Y\_pred)^2)  
  
test\_errors\_ridge[i] = MSE\_ridge  
  
}  
  
test\_errors\_lasso = rep(0,50)  
  
for ( i in 1:50){  
 set.seed(i)  
 X\_train = array(rnorm(p\*N),c(N,p))  
eps\_train = rnorm(N)  
Nte = 10^3  
X\_te = array(rnorm(p\*Nte),c(Nte,p))  
eps\_te = rnorm(Nte)  
grid = 10^seq(10,-2,length = 100)  
  
df = as.data.frame(X\_train)  
#sample(df)  
  
Y\_train = 2\*df$V1 + 2 \* df$V2 + 2\*df$V3 +2\*df$V4 + 2\*df$V5 + eps\_train  
  
  
#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(X\_train, Y\_train, alpha = 1, K=10)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
  
df = as.data.frame(X\_te)  
#sample(df)  
  
Y\_test = 2\*df$V1 + 2 \* df$V2 + 2\*df$V3 +2\*df$V4 + 2\*df$V5 + eps\_train  
  
Y\_pred = predict(cv\_model, newx = X\_te, s = best\_lambda )  
MSE\_lasso = mean((Y\_test-Y\_pred)^2)  
  
test\_errors\_lasso[i] = MSE\_lasso  
  
}

boxplot(test\_errors\_lasso, main = "Lasso Test Error Sparse Model")

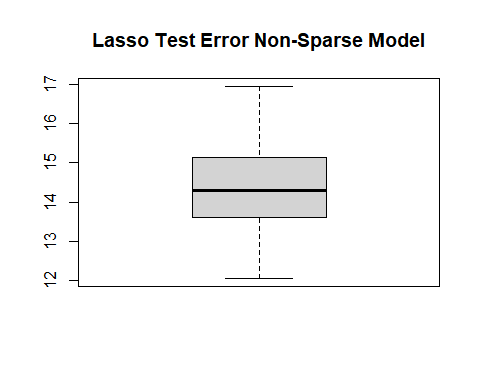


boxplot(test\_errors\_ridge, main = "Ridge Test Error Sparse Model")

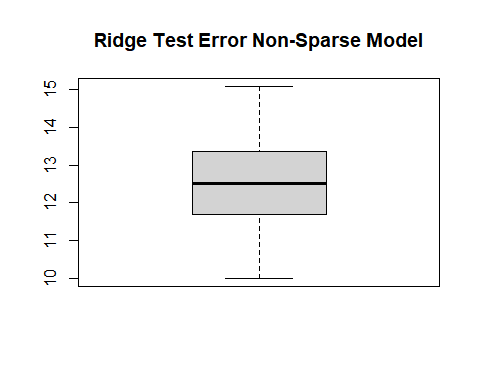
 # Lasso Test MSE is lower on average for the sparse model

test\_errors\_ridge = rep(0,50)  
  
for ( i in 1:50){  
 set.seed(i)  
 X\_train = array(rnorm(p\*N),c(N,p))  
eps\_train = rnorm(N)  
Nte = 10^3  
X\_te = array(rnorm(p\*Nte),c(Nte,p))  
eps\_te = rnorm(Nte)  
grid = 10^seq(10,-2,length = 100)  
  
df = as.data.frame(X\_train)  
#sample(df)  
  
Y\_train = eps\_train  
  
for (q in df) {  
 Y\_train = Y\_train + .5\*q  
}  
  
  
#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(X\_train, Y\_train, alpha = 0, K=10)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
  
df = as.data.frame(X\_te)  
#sample(df)  
  
Y\_test = eps\_train  
  
for (q in df) {  
 Y\_train = Y\_train + .5\*q  
}  
  
Y\_pred = predict(cv\_model, newx = X\_te, s = best\_lambda )  
MSE\_ridge = mean((Y\_test-Y\_pred)^2)  
  
test\_errors\_ridge[i] = MSE\_ridge  
  
}  
  
test\_errors\_lasso = rep(0,50)  
  
for ( i in 1:50){  
 set.seed(i)  
 X\_train = array(rnorm(p\*N),c(N,p))  
eps\_train = rnorm(N)  
Nte = 10^3  
X\_te = array(rnorm(p\*Nte),c(Nte,p))  
eps\_te = rnorm(Nte)  
grid = 10^seq(10,-2,length = 100)  
  
df = as.data.frame(X\_train)  
#sample(df)  
  
Y\_train = eps\_train  
  
for (q in df) {  
 Y\_train = Y\_train + .5\*q  
}  
  
  
#perform k-fold cross-validation to find optimal lambda value  
cv\_model <- cv.glmnet(X\_train, Y\_train, alpha = 1, K=10)  
  
#find optimal lambda value that minimizes test MSE  
best\_lambda <- cv\_model$lambda.min  
  
df = as.data.frame(X\_te)  
#sample(df)  
  
Y\_test = eps\_train  
  
for (q in df) {  
 Y\_train = Y\_train + .5\*q  
}  
  
Y\_pred = predict(cv\_model, newx = X\_te, s = best\_lambda )  
MSE\_lasso = mean((Y\_test-Y\_pred)^2)  
  
test\_errors\_lasso[i] = MSE\_lasso  
  
}

boxplot(test\_errors\_lasso, main = "Lasso Test Error Non-Sparse Model")



boxplot(test\_errors\_ridge, main = "Ridge Test Error Non-Sparse Model")

 # Ridge Test MSE is lower on average for the non-sparse model

1. (Problem 8)

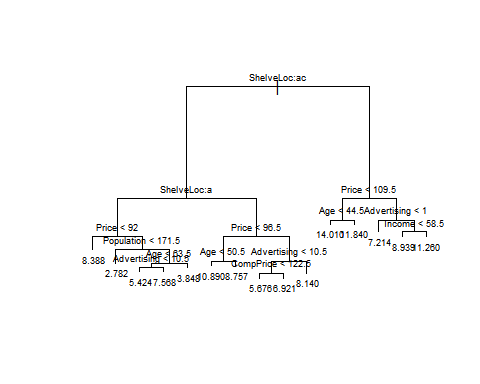
library(tree)

## Warning: package 'tree' was built under R version 4.2.2

library(ISLR)  
  
data = Carseats

# a  
  
set.seed(12)  
  
train = sample(length(data$Sales), length(data$Sales)/2)  
  
training = data[train,]  
testing = data[-train,]

# b  
  
tree <- tree(Sales ~ ., data = training)  
plot(tree)  
text(tree, cex = .56)

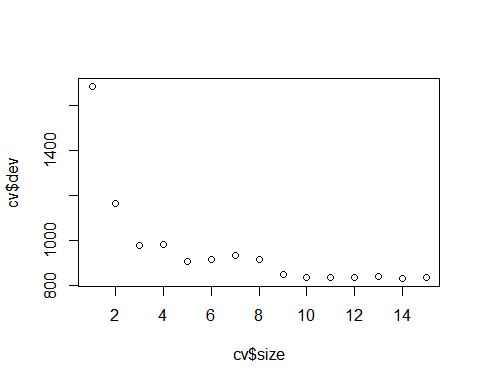


treepred <- predict(tree, newdata = testing)  
MSE = mean((treepred - testing$Sales)^2)  
MSE

## [1] 5.08059

MSE is 5.08059

# c  
  
cv <- cv.tree(tree)  
plot(cv$size, cv$dev)



# min dev appears at size 9  
  
prunedtree <- prune.tree(tree, best = 9)  
prunetreepred <- predict(prunedtree, newdata = testing)  
MSE = mean((prunetreepred - testing$Sales)^2)  
MSE

## [1] 5.184776

The pruned tree had a greater test MSE (5.184776) than the unpruned tree (5.08059). This indicates that the unpruned tree was not overfitting to the training data.

# d  
  
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.2.2

## randomForest 4.7-1.1

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

bagging <- randomForest(Sales ~ ., data = Carseats, subset = train,  
 importance = TRUE)  
bagging

##   
## Call:  
## randomForest(formula = Sales ~ ., data = Carseats, importance = TRUE, subset = train)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 3  
##   
## Mean of squared residuals: 3.111895  
## % Var explained: 62.06

predictbagging <- predict(bagging, newdata = testing)  
MSE = mean((predictbagging - testing$Sales)^2)  
  
MSE

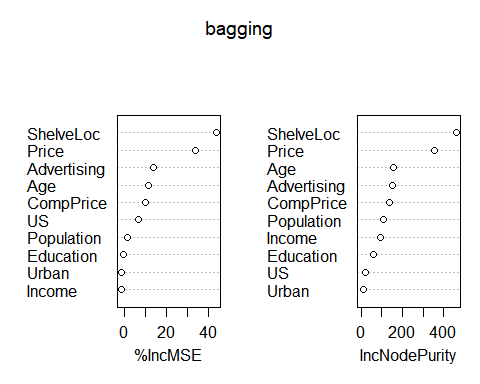
## [1] 2.978925

MSE of bagging is 3.044373, lower than the two trees

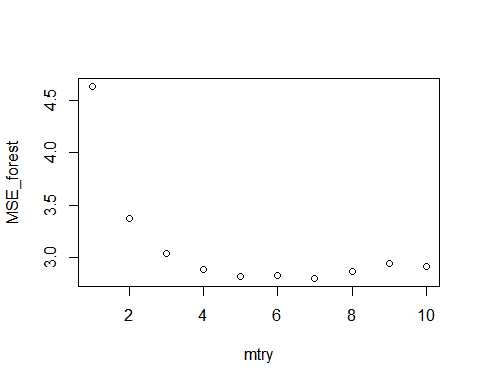
importance(bagging)

## %IncMSE IncNodePurity  
## CompPrice 9.8164095 136.523493  
## Income -1.6603266 92.082628  
## Advertising 13.8431083 152.522949  
## Population 1.5370312 110.047031  
## Price 33.4688136 353.583644  
## ShelveLoc 43.8150852 464.048339  
## Age 11.5547448 155.528390  
## Education -0.3085175 59.650396  
## Urban -1.4363594 9.837111  
## US 6.5590849 23.462175

varImpPlot(bagging)

 Shelf location and price are the most important variables according to bagging.

# e  
  
MSE\_forest = rep(0,10)  
  
for (i in 1:10){  
 randomf <- randomForest(Sales ~ ., data = Carseats, subset = train, mtry = i,  
 importance = TRUE)  
 randomf.pred <- predict(randomf, newdata = testing)  
 MSE\_forest[i] = mean((randomf.pred - testing$Sales)^2)  
}  
  
mtry = 1:10  
  
plot(mtry,MSE\_forest)



MSE\_forest

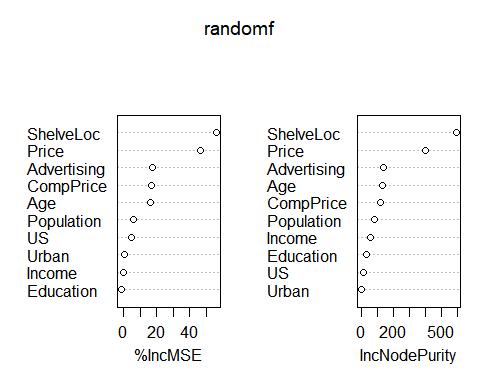
## [1] 4.636720 3.368603 3.033561 2.880979 2.820345 2.829977 2.799821 2.868452  
## [9] 2.940906 2.916626

The value of m with the minimum test MSE is m=7 with an MSE of 2.833899, lower than any other model MSE tried so far

randomf <- randomForest(Sales ~ ., data = Carseats, subset = train, mtry = 7,  
 importance = TRUE)  
importance(randomf)

## %IncMSE IncNodePurity  
## CompPrice 16.62954511 123.28881  
## Income 0.04942483 58.87416  
## Advertising 17.60889946 142.93691  
## Population 6.07750234 84.72412  
## Price 46.53321859 401.49444  
## ShelveLoc 56.49994157 597.31544  
## Age 16.51273087 134.80199  
## Education -1.58469327 34.36782  
## Urban 0.33351371 4.15932  
## US 4.90317994 13.09051

varImpPlot(randomf)

 Random forest also claims that shelf location, followed by price are the most important variables

# f  
  
library(BayesTree)

## Warning: package 'BayesTree' was built under R version 4.2.2

x = subset(training, select = -Sales )  
y = training$Sales  
  
bart = bart(x,y)

##   
##   
## Running BART with numeric y  
##   
## number of trees: 200  
## Prior:  
## k: 2.000000  
## degrees of freedom in sigma prior: 3  
## quantile in sigma prior: 0.900000  
## power and base for tree prior: 2.000000 0.950000  
## use quantiles for rule cut points: 0  
## data:  
## number of training observations: 200  
## number of test observations: 0  
## number of explanatory variables: 12  
##   
##   
## Cutoff rules c in x<=c vs x>c  
## Number of cutoffs: (var: number of possible c):  
## (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)   
## (6: 100) (7: 100) (8: 100) (9: 100) (10: 100)   
## (11: 100) (12: 100)   
##   
##   
## Running mcmc loop:  
## iteration: 100 (of 1100)  
## iteration: 200 (of 1100)  
## iteration: 300 (of 1100)  
## iteration: 400 (of 1100)  
## iteration: 500 (of 1100)  
## iteration: 600 (of 1100)  
## iteration: 700 (of 1100)  
## iteration: 800 (of 1100)  
## iteration: 900 (of 1100)  
## iteration: 1000 (of 1100)  
## iteration: 1100 (of 1100)  
## time for loop: 6  
##   
## Tree sizes, last iteration:  
## 2 2 4 2 2 2 3 2 2 3 4 2 3 2 2 2 2 2 4 3   
## 2 2 3 3 2 2 3 2 2 2 4 2 2 2 3 3 2 2 2 2   
## 3 3 1 2 2 2 2 2 3 2 2 2 2 3 2 1 2 5 2 2   
## 2 2 2 2 2 2 3 3 2 2 2 2 3 2 3 3 2 2 3 3   
## 2 2 1 3 2 2 2 2 2 3 3 3 2 3 2 2 3 2 2 2   
## 2 2 2 1 2 1 2 2 2 2 2 2 2 1 2 3 2 2 4 2   
## 2 1 2 2 2 2 2 2 4 2 2 3 6 3 2 3 2 1 2 2   
## 2 2 2 2 2 2 2 2 2 2 2 3 2 2 4 2 2 2 2 2   
## 2 2 3 2 2 3 3 2 2 3 2 2 3 2 4 5 2 3 4 2   
## 3 3 2 3 2 2 1 4 3 3 2 2 2 2 4 2 3 2 3 4   
## Variable Usage, last iteration (var:count):  
## (1: 21) (2: 22) (3: 23) (4: 23) (5: 34)   
## (6: 24) (7: 13) (8: 25) (9: 24) (10: 18)   
## (11: 22) (12: 20)   
## DONE BART 11-2-2014

MSE = mean((bart$y-testing$Sales)^2)  
MSE

## [1] 16.03221

BART MSE is highest MSE tested