Homework 5

Rohan Krishnan

2024-03-05

Problem 1

No submission required

Problem 2

ISLR Chapter 6 Conceptual Exercise 2

(a) iii. is correct because lasso adds a penalty term to the OLS procedure, increasing bias. This increase in bias is done with the hope that the (hopefully small) increase in bias effectively filters out variables and drastically reduces the variance of the model, allowing it to perform similarly across different training sets.

(b) iii. is correct because, similar to lasso, ridge regression adds a (different) penalty term to the OLS procedure. This increases the bias with the hopes of a significant decrease in variance.

(c) ii. is correct because non-linear models are able to fit more complex relationships in data compared to linear models. For larger and more complex data sets, non-linear models can perform better as long as the increase in variance is not very substantial.

ISLR Chapter 6 Conceptual Exercise 3

(a) iv. As s increases from 0, the training RSS will decrease because the model will have more variables to use to explain the variation, thus becoming more flexible. As s continues to increase, the model will become more flexible so the training RSS will always decrease

(b) ii. At first, as s increases from 0, the model will have variables that can effectively explain variation and while having enough bias to perform well on unseen data, causing a decrease in the test RSS. However, as s continues to increase, the model will begin to over fit the training data and the test RSS will begin to increase again, creating a U shape.

(c) iii. As s increases from 0, more variables will be able to be used in the model, allowing for more flexibility but consequently causing a steady increase in variance.

(d) iv. As s increases from 0, more variables will be able to be used in the model, allowing it to more closely fit the true relationship in the training data and steadily decreasing the bias.

(e) v. The irreducible error will remain unchanged regardless of how many variables are used in the model.

ISLR Chapter 6 Conceptual Exercise 4

(a) iii. As we increase lambda from 0, the penalty term, representing the sum of the squared coefficients will increase exponentially, causing a steady increase in the training RSS.

(b) ii. As we increase lambda from 0, the model will become more biased and less flexible. Initially, the reduction in variance will cause the testing RSS to decrease. However, the bias will eventually increase to the point that the model's performance suffers and the testing RSS will increase, causing an overall U shape.

(c) iv. As we increase lambda from 0, the model will be increasingly penalized for each additional estimator. Thus, in an effort to minimize RSS, it will include only the most necessary estimators and steadily decrease the variance.

(d) iii. As we increase lambda from 0, the model will steadily increase in squared bias as each additional coefficient exponentially increases the RSS.

(e) v. The irreducible error will remain unchanged regardless of how we penalize additional variables in the model.

Problem 3

ISLR Chapter 6 Applied Exercise 9

(a)

```r
#Load ISLR 2 library and dplyr
library(ISLR2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
#Create id column
df <- College %>% mutate(id = row_number())

#Check IDs
head(df$id)
```

```
## [1] 1 2 3 4 5 6
```

```r
#Set seed
set.seed(25)

#Create training set
train <- df %>% sample_frac(.70)

#Create test set
test  <- anti_join(df, train, by = 'id')

#Remove id column
train$id <- NULL
test$id <- NULL
```

(b)

```r
#Create OLS model
ols.mod <- lm(Apps ~ ., train)

#Report test error
ols.mse <- mean((predict(ols.mod, test) - test$Apps)^2); ols.mse
```

```
## [1] 1908377
```

(c)

```
#Load glmnet library
library(glmnet)
```

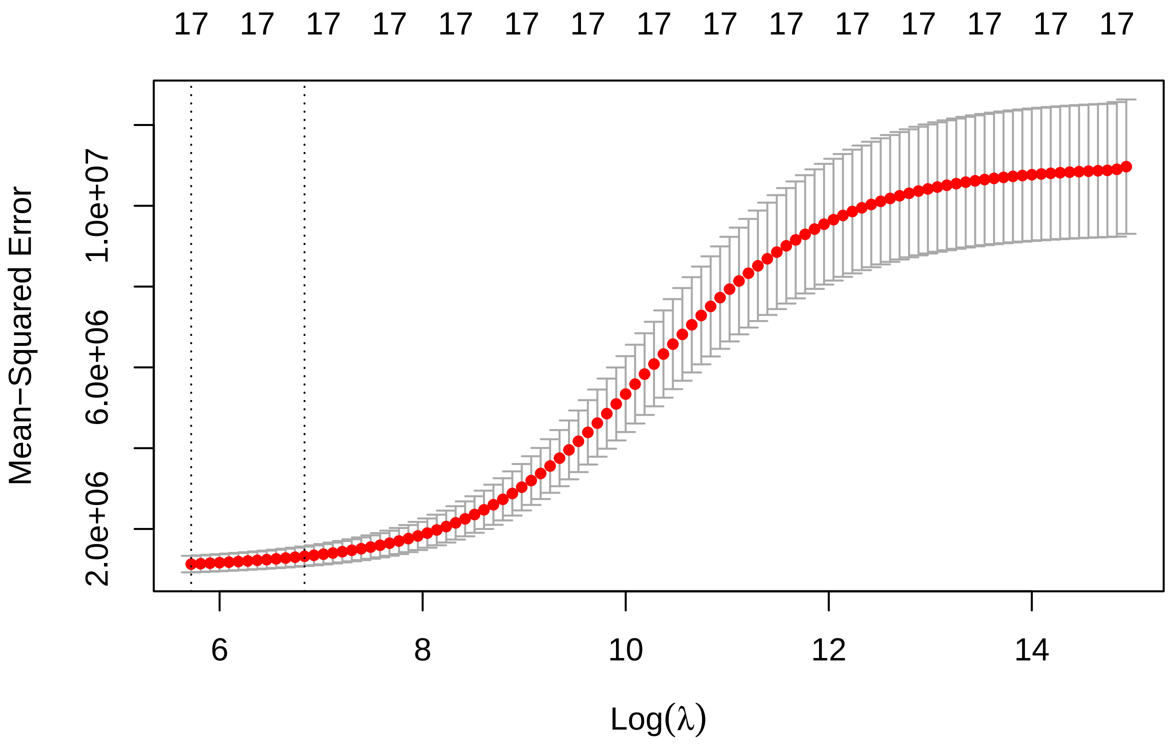
```
## Loading required package: Matrix
```

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

```
#Define predictor and response variables
x.train <- model.matrix(Apps ~ ., train)[,-1]
y.train <- train %>%
  select(Apps) %>%
  unlist() %>%
  as.numeric()
x.test <- model.matrix(Apps ~ ., test)[,-1]
y.test <- test %>%
  select(Apps) %>%
  unlist() %>%
  as.numeric()

#Fit ridge CV regression on training data and recover optimal lambda
cv.ridge.mod <- cv.glmnet(x.train, y.train, alpha = 0)
best.lambda <- cv.ridge.mod$lambda.min; best.lambda
```

```
## [1] 304.904
```

```
#Plot MSE as a function of lambda
plot(cv.ridge.mod)
```



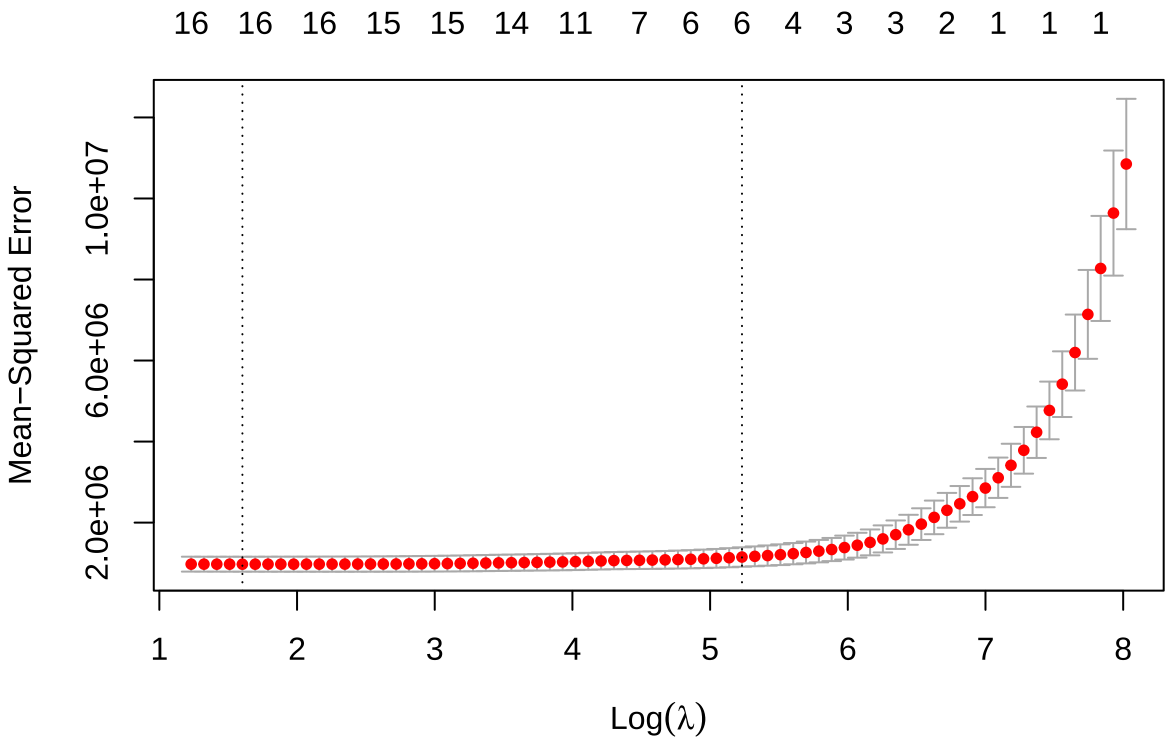
```
#Report test error
ridge.mse <- mean((predict(glmnet(x.train, y.train, alpha = 0), s = best.lambda, newx = x.test) - y.tes
## [1] 3210022
```

(d)

```
#Fit lasso CV regression on training data and recover optimal lambda
cv.lasso.mod <- cv.glmnet(x.train, y.train, alpha = 1)
best.lambda1 <- cv.lasso.mod$lambda.min; best.lambda1
```

```
## [1] 4.969175
```

```
#Plot MSE as a function of lambda
plot(cv.lasso.mod)
```



```
#Report test error
lasso.mse <- mean((
  predict(glmnet(x.train, y.train, alpha = 1), s = best.lambda1, newx = x.test)
                 - y.test)^2
  ); lasso.mse
```

```
## [1] 1893907
```

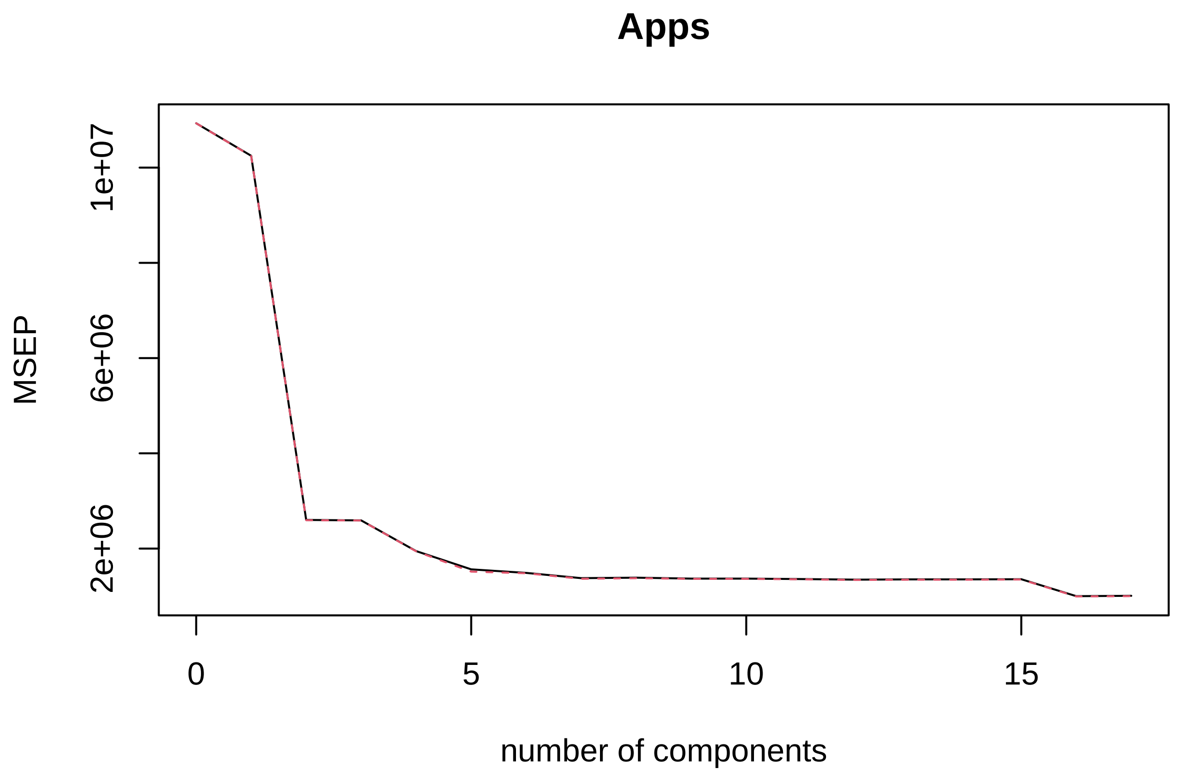
(e)

```
#Load pls library
library(pls)
```

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
##
##     loadings
```

```
#Fit PCR model
pcr.mod <- pcr(Apps ~ .,data = train, scale = TRUE, validation = "CV")
```

```
#Plot MSE as a function of M values and pull ncomp values -- Selected M of 17
validationplot(pcr.mod, val.type = "MSEP")
```



```
pcr.mod$ncomp
```

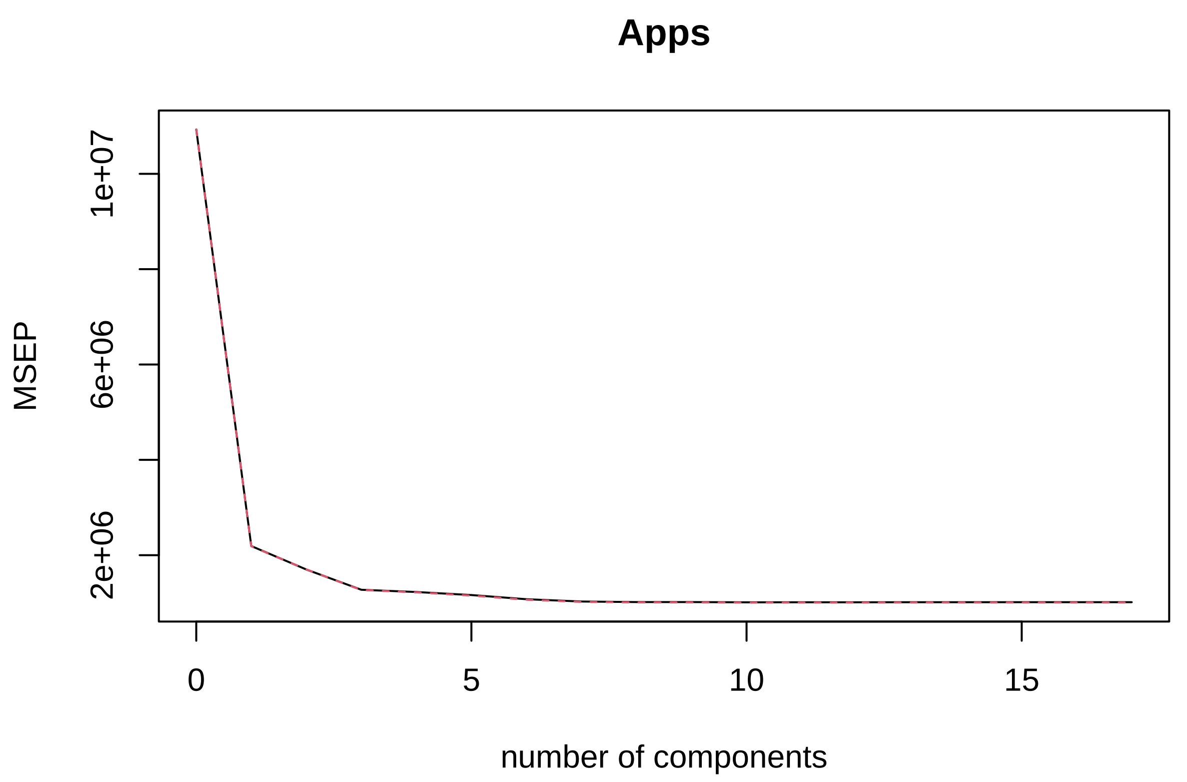
```
## [1] 17
```

```
#Report test error
pcr.mse <- mean((predict(pcr.mod, test, ncomp = 17) - test$Apps)^2); pcr.mse
```

```
## [1] 1908377
```

(f)

```
#Fit PLS model
pls.mod <- plsr(Apps ~., data = train, scale = TRUE, validation = "CV")

#Plot MSE and recover ncomp values -- Selected M of 17
validationplot(pls.mod, val.type = "MSEP")
```



```
pls.mod$ncomp
```

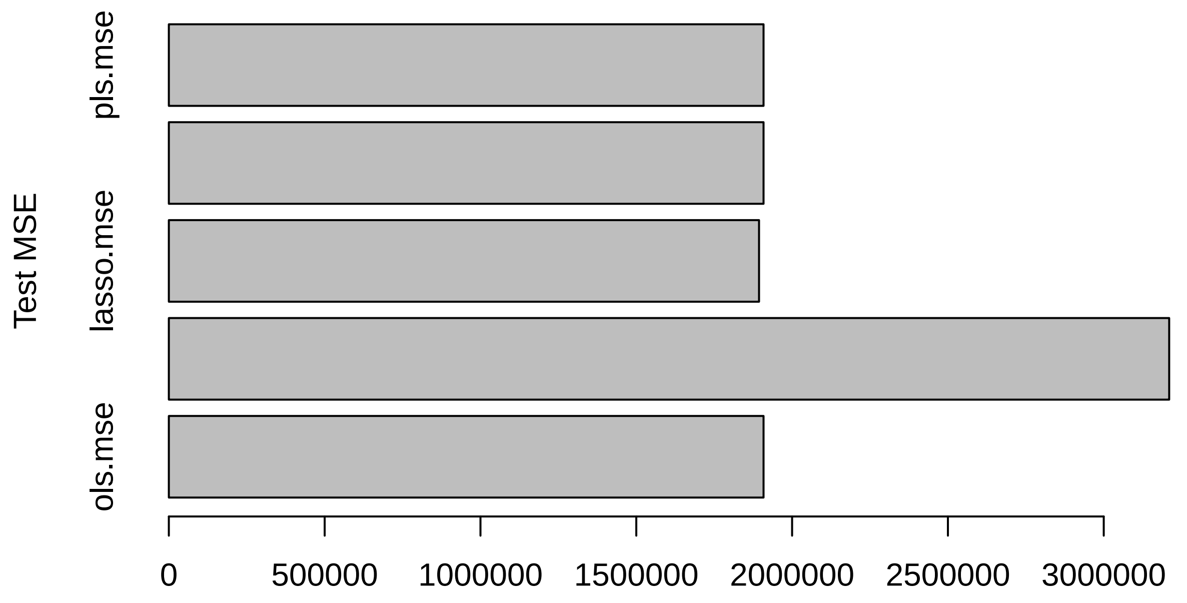
```
## [1] 17
```

```
#Report test error
pls.mse <- mean((predict(pls.mod, test, ncomp = 17) - test$Apps)^2); pls.mse
```

```
## [1] 1908377
```

(g)

```
mse.list <- data.frame(ols.mse, ridge.mse, lasso.mse, pcr.mse, pls.mse)
barplot(unlist(mse.list), ylab = "Test MSE", horiz = TRUE)
```



ISLR Chapter 6 Applied Exercise 10

(a)

```
#Set seed
set.seed(420)

#Create predictors
pred <- matrix(rnorm(1000*20), nrow = 1000)
colnames(pred) <- paste0("x", 1:20)

#Create beta values
beta <- rep(0, 20)
beta[1:6] <- c(4,2,9,7,3,5)

#Create y
y <- colSums((t(pred) * beta)) + rnorm(1000)

#Combine into one data frame
data <- data.frame(pred)
data$y <- y
```

(b)

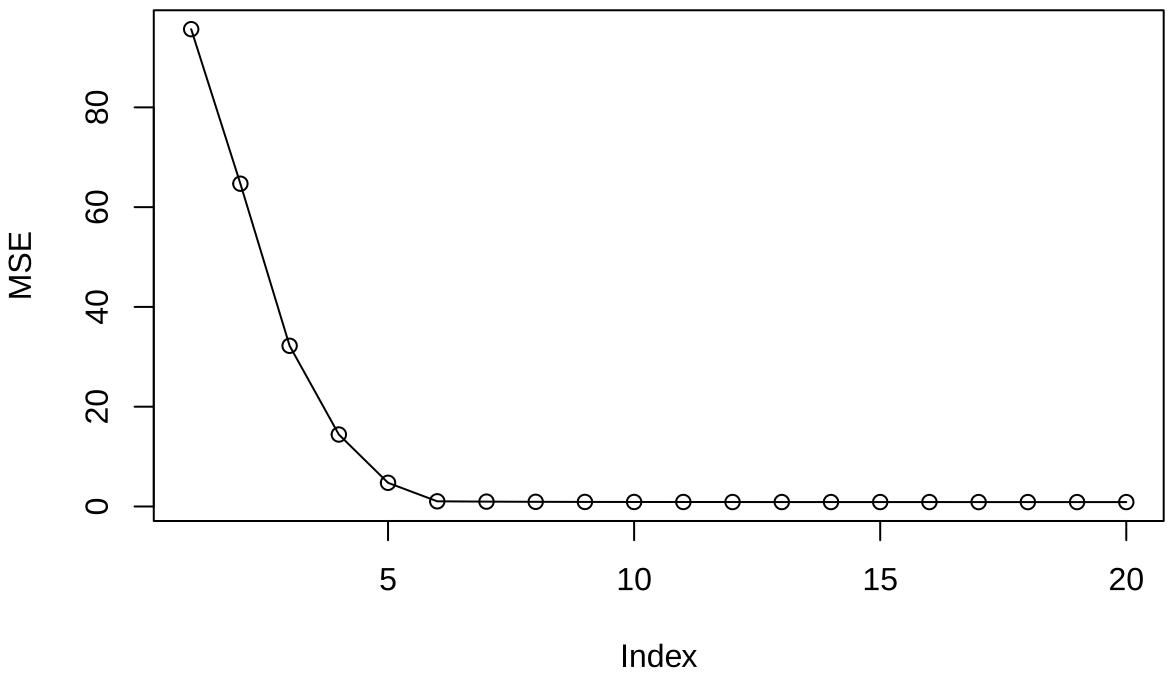
```
#Create train and test sets
train <- data[1:100,]
test <- data[101:1000,]
```

(c)

```
#Load leaps library
library(leaps)

#Perform BSS on training data
bss.fit <- regsubsets(y ~ ., data = train, nvmax = 20)
summary(bss.fit)
```

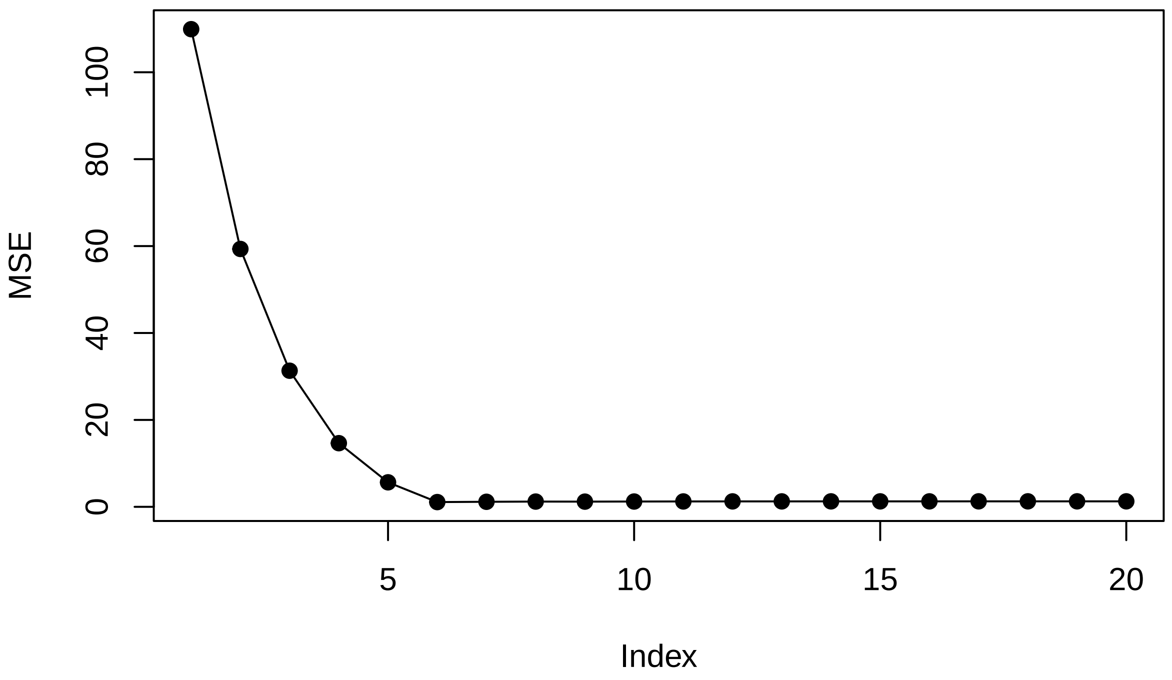
```
## Subset selection object
## Call: regsubsets.formula(y ~ ., data = train, nvmax = 20)
## 20 Variables  (and intercept)
##     Forced in Forced out
## x1      FALSE      FALSE
## x2      FALSE      FALSE
## x3      FALSE      FALSE
## x4      FALSE      FALSE
## x5      FALSE      FALSE
## x6      FALSE      FALSE
## x7      FALSE      FALSE
## x8      FALSE      FALSE
## x9      FALSE      FALSE
## x10     FALSE      FALSE
## x11     FALSE      FALSE
## x12     FALSE      FALSE
## x13     FALSE      FALSE
## x14     FALSE      FALSE
## x15     FALSE      FALSE
## x16     FALSE      FALSE
## x17     FALSE      FALSE
## x18     FALSE      FALSE
## x19     FALSE      FALSE
## x20     FALSE      FALSE
## 1 subsets of each size up to 20
## Selection Algorithm: exhaustive
##           x1  x2  x3  x4  x5  x6  x7  x8  x9  x10 x11 x12 x13 x14 x15 x16 x17
## 1  ( 1 )  " " " " "*" " " " " " " " " " " " " " " " " " " " " " " " " " " " "
## 2  ( 1 )  " " " " "*" "*" " " " " " " " " " " " " " " " " " " " " " " " " " "
## 3  ( 1 )  " " " " "*" "*" " " "*" " " " " " " " " " " " " " " " " " " " " " "
## 4  ( 1 )  "*" " " "*" "*" " " "*" " " " " " " " " " " " " " " " " " " " " " "
## 5  ( 1 )  "*" " " "*" "*" "*" "*" " " " " " " " " " " " " " " " " " " " " " "
## 6  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " " " " " " " " " " " " " " " " " " "
## 7  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " " " " " " " " " " " " " " " " " " "
## 8  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " "*" " " " " " " " " " " " " " " " "
## 9  ( 1 )  "*" "*" "*" "*" "*" "*" "*" " " "*" " " " " " " " " " " " " " " " "
## 10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" " " "*" " " " " " " "*" " " " " " " " "
## 11  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " " " "*" " " " " " " " "
## 12  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " " " "*" "*" " " " " " "
## 13  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " " " "*" "*" " " " " "*"
## 14  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " " " "*" "*" " " " " "*"
## 15  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " " " "*" "*" " " " " "*"
## 16  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " "*" "*" "*" " " " " "*"
## 17  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " "*" "*" "*" " " "*" "*"
## 18  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " "*" "*" "*" " " "*" "*"
## 19  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " "*" "*"
## 20  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
##           x18 x19 x20
## 1  ( 1 )  " " " " " "
## 2  ( 1 )  " " " " " "
## 3  ( 1 )  " " " " " "
## 4  ( 1 )  " " " " " "
## 5  ( 1 )  " " " " " "
## 6  ( 1 )  " " " " " "
## 7  ( 1 )  " " "*" " "
## 8  ( 1 )  " " "*" " "
## 9  ( 1 )  " " "*" " "
## 10  ( 1 ) " " "*" " "
## 11  ( 1 ) " " "*" " "
## 12  ( 1 ) " " "*" " "
## 13  ( 1 ) " " "*" " "
## 14  ( 1 ) " " "*" "*"
## 15  ( 1 ) "*" "*" "*"
## 16  ( 1 ) "*" "*" "*"
## 17  ( 1 ) "*" "*" "*"
## 18  ( 1 ) "*" "*" "*"
## 19  ( 1 ) "*" "*" "*"
## 20  ( 1 ) "*" "*" "*"
```

```r
#Graph training mse
bss.mse <- summary(bss.fit)$rss / nrow(train)
plot(bss.mse, ylab = "MSE", type = "o")
```



(d)

```
#Create function to predict values for each best model
predict.regsubsets <- function(object, newdata, id, ...) {
  #Extract formula
  form <- as.formula(object$call[[2]])
  #Create model matrix with test set
  mat <- model.matrix(form, newdata)
  #Choose coefficients based on number of predictors(id)
  coefi <- coef(object, id = id)
  #Get chosen variables names
  xvars <- names(coefi)
  #Multiply chosen variable values by coefficient values
  mat[, xvars] %*% coefi
}

#Calculate test MSE for models of size 1 to 20
mse <- sapply(1:20, function(i) mean((test$y - predict.regsubsets(bss.fit, test, i))^2))

#Plot test MSE
plot(mse, ylab = "MSE", type = "o", pch = 19)
```



(e) The test MSE takes on its minimum value at a model size of 6. This matches the simulated data we made above, illustrating that BSS evaluated via test MSE can recover an accurate model.

(f)

```
which.min(mse)
```

```
## [1] 6
```

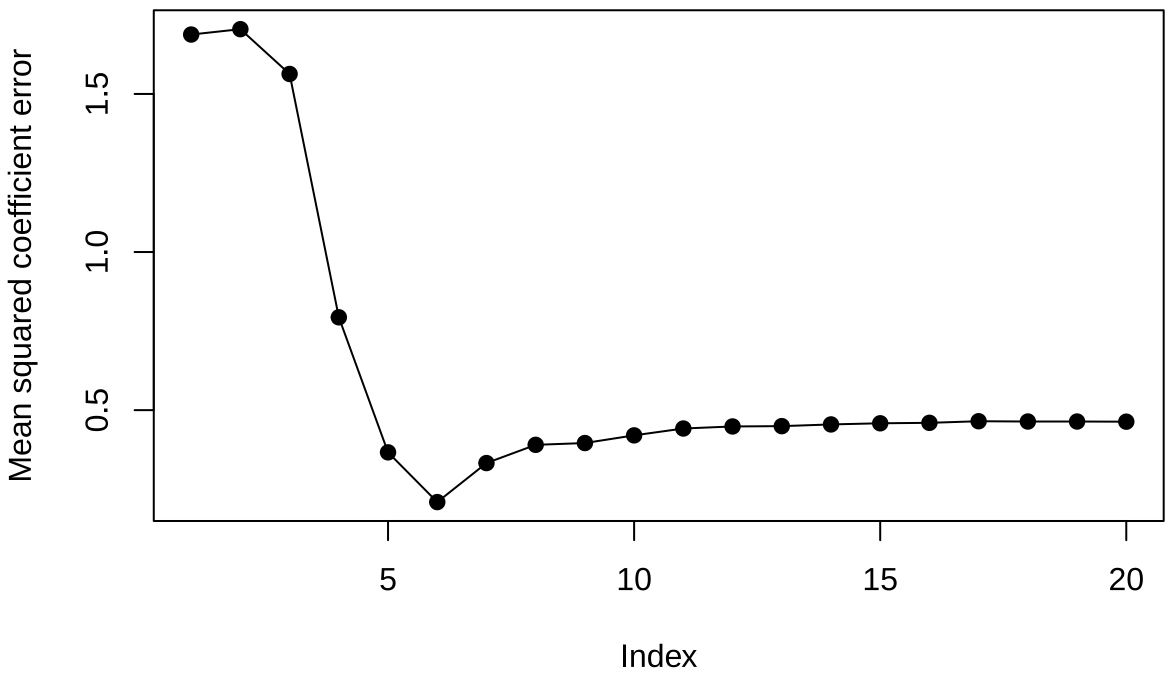
```
coef(bss.fit, id=6)
```

```
## (Intercept)          x1          x2          x3          x4          x5
##   -0.115987    3.873882    2.082003    8.937569    6.936385    2.915940
##          x6
##    5.078210
```

It appears to estimate the simulated coefficients relatively well.

(g)

```
names(beta) <- paste0("x", 1:20)
truevals <- data.frame(id = names(beta), b = beta)

out <- sapply(1:20, function(i) {
  modelvals <- coef(bss.fit, id = i)[-1]
  modelvals <- data.frame(id = names(modelvals), c = modelvals)
  comparevals <- merge(truevals, modelvals)
  sqrt(sum((comparevals$b - comparevals$c)^2))
})
plot(out, ylab = "Mean squared coefficient error", type = "o", pch = 19)
```



Just like with the test MSE plot, the values are minimized at a model size of 6, which matches the true model.

Problem 4

(a)

```r
#Set seed
set.seed(158)

#Generate 100 obs on 10 features
n <- 100
p <- 10
predictors <- matrix(rnorm(n = n*p), nrow = n)
colnames(predictors) <- paste0("x", 1:10)

#Generate betas
betas <- rep(0, 10)
betas[1:5] <- 1

#Generate y
y <- colSums((t(predictors)*betas)) + rnorm(100, mean = 0, sd = 0.5)

#Combine into data frame
train <- data.frame(predictors)
train$y <- y
```

(b)

```r
#Set seed
set.seed(158)

#Generate 10000 obs on 10 features
n <- 10000
p <- 10
```

```r
predictors <- matrix(rnorm(n*p), nrow = n)
colnames(predictors) <- paste0("x", 1:10)

#Generate y
y <- colSums((t(predictors)*betas)) + rnorm(10000, mean = 0, sd = 0.50)

#Combine into data frame
test <- data.frame(predictors)
test$y <- y
```

(c)

```r
#Extract x.train and y.train
x.train <- model.matrix(y ~ ., train)[,-1]
y.train <- train %>%
  select(y) %>%
  unlist() %>%
  as.numeric()
x.test <- model.matrix(y~., test)[,-1]
y.test <- test %>%
  select(y) %>%
  unlist() %>%
  as.numeric()

#Fit lasso CV regression on training data and recover optimal lambda
lasso1 <- cv.glmnet(x.train, y.train, alpha = 1)
best.lambda1 <- lasso1$lambda.min; best.lambda1
```

```
## [1] 0.0389894
```

```r
#Report test error
Err.lasso1 <- mean((
  predict(glmnet(x.train, y.train, alpha = 1),
          s = best.lambda1, newx = x.test)
          - y.test)^2
  ); Err.lasso1
```

```
## [1] 0.2753989
```

(d)

```r
#Recover chosen variables and create new train set
tmp_coef = coef(lasso1, s=best.lambda1)
varnames <- tmp_coef@Dimnames[[1]][tmp_coef@i][-1]
new.train <- train[,varnames]
new.train$y <- train$y

#Fit OLS model
ols1 <- lm(y ~ ., new.train)

#Record test error
Err.ols1 <- mean((predict(ols1, test) - test$y)^2); Err.ols1
```

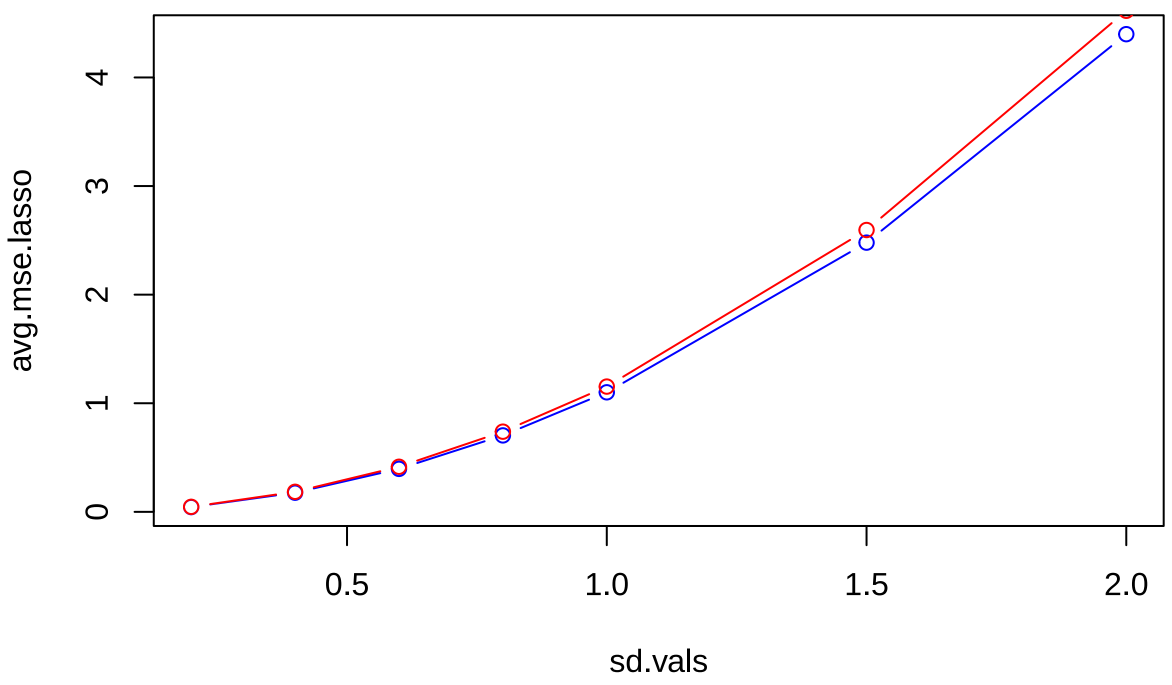
```
## [1] 0.275877
```

(e)

```r
#Create object to hold mse values
mse.lasso <- rep(0,1000)
mse.ols <- rep(0,1000)

#Create test set to use across all loops
#Set seed
set.seed(158)

#Generate 10000 obs on 10 features
predictors <- matrix(rnorm(10000*10), nrow = 10000)
colnames(predictors) <- paste0("x", 1:10)

#Generate betas
b <- rep(0,10)
b[1:5] <- 1

#Generate y
test.response <- colSums((t(predictors)*b)) + rnorm(10000, mean = 0, sd = 0.50)

#Combine into data frame
testing <- data.frame(predictors)
testing$y <- test.response

#Create for loop to iterate over new training set and perform operations
for(i in 1:1000){
  #Create train set
  set.seed(158)
  features <- matrix(rnorm(n = 100*10), nrow = 100)
  colnames(features) <- paste0("x", 1:10)
  y <- colSums((t(features)*b)) + rnorm(100, mean = 0, sd = 0.50)
  training <- data.frame(features)
  training$y <- y

  #Extract x.train, y.train, x.test, and y.test
  x.training <- model.matrix(y ~ ., training)[,-1]
  y.training <- training %>%
    select(y) %>%
    unlist() %>%
    as.numeric()
  x.testing <- model.matrix(y ~ ., testing)[,-1]
  y.testing <- testing %>%
    select(y) %>%
    unlist() %>%
    as.numeric()

  #Fit lasso CV and find best lambda
  lasso <- cv.glmnet(x.training, y.training, alpha = 1)
  best.lambda <- lasso$lambda.min

  #Store test error
  predicted.lasso <- predict(glmnet(x.training, y.training, alpha = 1),
                             s = best.lambda, newx = x.testing)
  mse.lasso[i] <- mean((predicted.lasso - y.testing)^2)
  #Recover chosen variables and make filtered train set
  chosen_coef <- coef(lasso, s = best.lambda)
  varnames <- chosen_coef@Dimnames[[1]][chosen_coef@i][-1]
  new.training <- training[,varnames]
  new.training$y <- training$y

  #Fit after lasso OLS
  ols <- lm(y ~ ., new.training)

  #Store test error
  predicted.ols <- predict(ols, testing)
  mse.ols[i] <- mean((predicted.ols - testing$y)^2)
}

#Calculate averages of mse
ErrLasso <- mean(mse.lasso)
ErrOLS <- mean(mse.ols)
```

(f)

```r
sd.vals <- c(0.2, 0.4, 0.6, 0.8, 1, 1.5, 2)
avg.mse.lasso <- rep(0,length(sd.vals))
avg.mse.ols <- rep(0,length(sd.vals))

for(i in 1:7){
  #Set standard deviation for error and create (reset) total error vectors
  std.dev <- sd.vals[i]
  tot.mse.lasso <- 0
  tot.mse.ols <- 0

  #For loop for each sd
  for(n in 1:10){
    #Create training set
    set.seed(158)
    features <- matrix(rnorm(n = 100*10), nrow = 100)
    colnames(features) <- paste0("x", 1:10)
    b <- c(rep(1,5), rep(0,5))
    y <- colSums((t(features)*b)) + rnorm(100, mean = 0, sd = std.dev)
    training <- data.frame(features)
    training$y <- y

    #Create testing set
    predictors <- matrix(rnorm(10000*10), nrow = 10000)
    colnames(predictors) <- paste0("x", 1:10)
    test.response <- colSums((t(predictors)*b)) + rnorm(10000, mean = 0, sd = std.dev)
    testing <- data.frame(predictors)
    testing$y <- test.response

    #Extract x.train, y.train, x.test, and y.test
    x.training <- model.matrix(y ~ ., training)[,-1]
    y.training <- training %>%
      select(y) %>%
      unlist() %>%
      as.numeric()
    x.testing <- model.matrix(y ~ ., testing)[,-1]
    y.testing <- testing %>%
      select(y) %>%
      unlist() %>%
      as.numeric()

    #Fit lasso CV and find best lambda
    lasso <- cv.glmnet(x.training, y.training, alpha = 1)
    best.lambda <- lasso$lambda.min

    #Store test error
    predicted.lasso <- predict(glmnet(x.training, y.training, alpha = 1),
                         s = best.lambda, newx = x.testing)
    mse.lasso.iter <- mean((predicted.lasso - y.testing)^2)
    tot.mse.lasso <- tot.mse.lasso + mse.lasso.iter

    #Recover chosen variables and make filtered train set
    chosen_coef <- coef(lasso, s = best.lambda)
    varnames <- chosen_coef@Dimnames[[1]][chosen_coef@i][-1]
    new.training <- training[,varnames]
    new.training$y <- training$y

    #Fit after lasso OLS
    ols <- lm(y ~ ., new.training)

    #Store test error
    predicted.ols <- predict(ols, testing)
    mse.ols.iter <- mean((predicted.ols - testing$y)^2)
    tot.mse.ols <- tot.mse.ols + mse.ols.iter
  }

  avg.mse.lasso[i] <- tot.mse.lasso/10
  avg.mse.ols[i] <- tot.mse.ols/10
}

plot(sd.vals, avg.mse.lasso, type = "b", col = "blue")
lines(sd.vals, avg.mse.ols, type = "b", col = "red")
```



Error increases as sigma increases, but the model with more flexibility (OLS after lasso) performs better as SNR increases.

(g) As SNR increases, models with more degrees of freedom that can fit more complex signals perform better. However, in data with low SNRs, models with higher bias and lower variance and degrees of freedom perform better as they don't over fit to the noise.