STAT 1361 - Homework 5

Gordon Lu

3/19/2021

Problem 2)

ISLR Conceptual Exercise 2

2a)

The lasso regression, relative to least squares is less flexible. Hence, will yield improved prediction accuracy when its increase in bias is less than its decrease in variance.

2b)

The ridge regression, relative to least square is less flexible and hence will yield improved prediction accruacy when its increase in bias is less than its decrease in variance.

2c)

The non-linear methods, relative to least squares, is more flexible and hence will yield improved prediction accuracy when its increase in variance is less than its ecrease in bias.

ISLR Conceptual Exercise 3

3a)

As `s` increases from 0, the training RSS will steadily decrease, since increasing `s` leads to restricting the coefficients to a lesser extent, and makes the model more flexible.

3b)

As `s` increases from 0, the test RSS will decrease initially, and then will eventually start increasing into a `U` shape, since the model becomes more flexible and will overestimate/overfit due to the extra parameters being involved.

3c)

As `s` increases from 0, the variance will steadily incres, since a more flexible model will result in higher variance.

3d)

As `s` increases from 0, the squared bias will steadily decrease since the model variance will increase and as variance increases, bias decreases due to the bias-variance tradeoff.

3e)

As `s` increases from 0, the irreducible error will remain constant since the definition of irreducibile error means it will remain independent of the chosen model. It's just some small offset showing that the model isn't the true model.

ISLR Conceptual Exercise 4

4a)

As we increase λ from 0, the training RSS will steadily increase since hte model becomes more flexible as λ increases, naturally leading to a higher RSS.

4b)

As we increase λ from 0, the test RSS will decrease initially, and then eventually start increasing in a `U` shape since the model becomes more flexible, leading to more overestimation/overfitting due to extra parameers being involved.

4c)

As we increase λ from 0, the variance will steadily decrease since the model is becoming less flexible since the coefficients are being restricted more and more, leading to lower variance.

4d)

As we increase λ from 0, the squared bias RSS will steadily increase sice we will be restricting the coefficients to smaller and smaller values, thus the model will become more flexible, leading to higher bias, due to the bias-variance tradeoff.

4e)

As we increase λ from 0, the irreducible error will remain constant since the definition of irreducible error means it will independent of the chosen model.

Problem 3)

ISLR Applied Exercise 9

9a)

```r
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.6.3
```

```r
data(College)
set.seed(1)
train <- sample(1:dim(College)[1], dim(College)[1]/2)
test <- -train
College.train <- College[train,]
College.test <- College[test,]
```

9b)

```r
fit.lm <- lm(Apps ~., data = College.train)
pred.lm <- predict(fit.lm, College.test)
mse <- mean((pred.lm - College.test$Apps)^2)

cat(mse)
```

```
## 1135758
```

The test MSE 1108531.

9c)

```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.6.3
```

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

```
## Loaded glmnet 3.0-2
```

```r
train.mat = model.matrix(Apps ~ ., data = College.train)
test.mat = model.matrix(Apps ~ ., data = College.test)
grid = 10^seq(4, -2, length=100)
fit.ridge = glmnet(train.mat, College.train$Apps, alpha = 0, lambda = grid, thresh = 1e-12)
cv.ridge = cv.glmnet(train.mat, College.train$Apps, alpha = 0, lambda = grid, thresh = 1e-12)
bestlam.ridge = cv.ridge$lambda.min

pred.ridge <- predict(fit.ridge, s = bestlam.ridge, newx = test.mat)
mean((pred.ridge - College.test$Apps)^2)
```

```
## [1] 1135714
```

```
cat(bestlam.ridge)
```

```
## 0.01
```

The test error is: 1108512, which is roughly the same as least squares.

9d)

```
fit.lasso = glmnet(train.mat, College.train$Apps, alpha = 1, lambda = grid, thresh = 1e-12)
cv.lasso = cv.glmnet(train.mat, College.train$Apps, alpha = 1, lambda = grid, thresh = 1e-12)
bestlam.lasso = cv.lasso$lambda.min
pred.lasso = predict(fit.lasso, s = bestlam.lasso, newx = test.mat)
lasso.mse <- mean((pred.lasso - College.test$Apps)^2)
pred.lasso <- predict(fit.lasso, s = bestlam.lasso, type = "coefficients")

pred.lasso <- predict(fit.lasso, s = bestlam.lasso, newx = test.mat)
mean((pred.lasso - College.test$Apps)^2)
```

```
## [1] 1135660
```

```
cat(lasso.mse)
```

```
## 1135660
```

The test error is 1135660, which is slightly lower than least squares and ridge regression. The number of non-zero coefficients is 15.

9e)

```
library(analogue)
```

```
## Warning: package 'analogue' was built under R version 3.6.3
```

```
## Loading required package: vegan
```

```
## Warning: package 'vegan' was built under R version 3.6.3
```

```
## Loading required package: permute
```

```
## Warning: package 'permute' was built under R version 3.6.3
```

```
## Loading required package: lattice
```

```
## This is vegan 2.5-7
```

```
## analogue version 0.17-5
```

```
library(pls)
```

```
## Warning: package 'pls' was built under R version 3.6.3
```

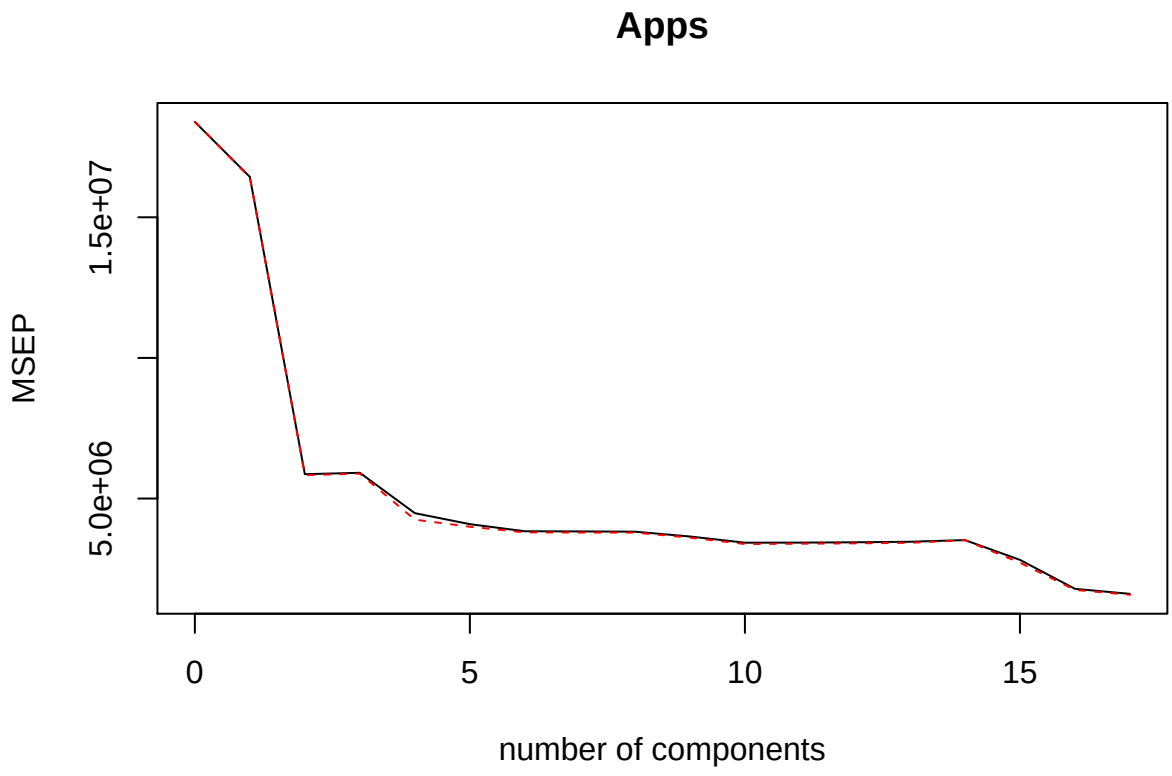
```
##
## Attaching package: 'pls'
```

```
## The following objects are masked from 'package:analogue':
##
##     crossval, pcr, RMSEP
```

```
## The following object is masked from 'package:vegan':
##
##     scores
```

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

```
fit.pcr = pcr(Apps ~ ., data = College.train, scale = TRUE, validation = "CV")
validationplot(fit.pcr, val.type = "MSEP")
```



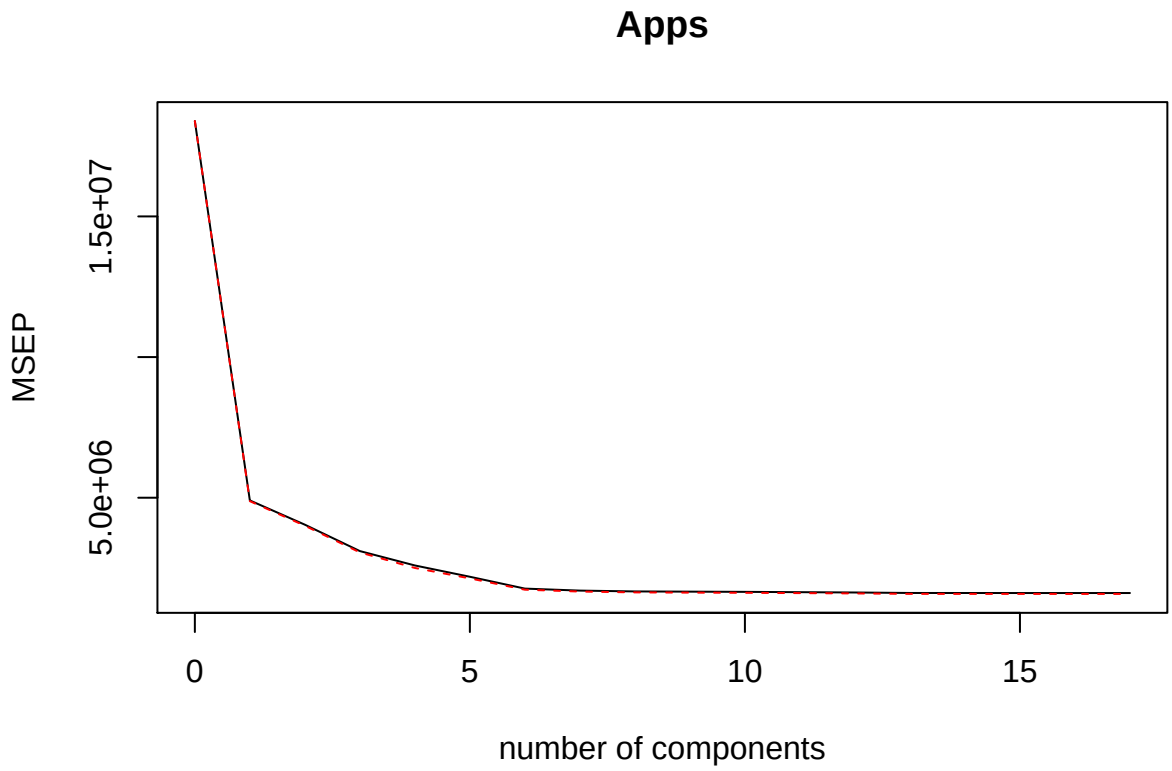
```
pred.pcr <- predict(fit.pcr, College.test, ncomp = 10)
pcr.mse <- mean((pred.pcr - College.test$Apps)^2)
cat(pcr.mse)
```

```
## 1723100
```

The test MSE is 1723100, which is higher than the other models observed thus far. The best value of M found through cross-validation is 17.

9f)

```
library(pls)
fit.pls = plsr(Apps ~ ., data = College.train, scale = TRUE, validation = "CV")
validationplot(fit.pls, val.type = "MSEP")
```




```
pred.pls = predict(fit.pls, College.test, ncomp = 10)
pls.mse <- mean((pred.pls - College.test$Apps)^2)
cat(pls.mse)
```

```
## 1131661
```

The test MSE is 1131661, which is similar to the other models. The best value for M found through Cross-Validation is 17, which is what we also found using PCR.

9g)

```
test.avg = mean(College.test$Apps)
lm.r2= 1 - mean((pred.lm - College.test$Apps)^2)/mean((test.avg-College.test$Apps)^2)
ridge.r2 = 1 - mean((pred.ridge - College.test$Apps)^2)/mean((test.avg-College.test$Apps)^2)
lasso.r2 = 1 - mean((pred.lasso - College.test$Apps)^2)/mean((test.avg-College.test$Apps)^2)
pcr.r2 = 1 - mean((pred.pcr - College.test$Apps)^2)/mean((test.avg-College.test$Apps)^2)
pls.r2 = 1 - mean((pred.pls - College.test$Apps)^2)/mean((test.avg-College.test$Apps)^2)
cat(lm.r2)
```

```
## 0.9015413
```

```
cat(ridge.r2)
```

```
## 0.9015452
```

```
cat(lasso.r2)
```

```
## 0.9015499
```

```
cat(pcr.r2)
```

```
## 0.8506248
```

```
cat(pls.r2)
```

```
## 0.9018965
```

The above calculates the correlation for each of the models (lm, ridge, lasso, pcr, pls in that order) with the predictions. Clearly, PCR has the lowest correlation by far out of all the models, although it's decent at 0.8127. However, all four other models have an r^2 value around 0.9, so they can predict college applications pretty accurately. None of the errors were too terrible different from each other (except for PCR), so we can't really say it's the most accurate distinguishing factor of these models; all would work perfectly fine in general.

ISLR Applied Exercise 10

10a)

```
set.seed(1)
x = matrix(rnorm(1000*20), 1000, 20)
b = rnorm(20)
b[3] = 0
b[4] = 0
b[9] = 0
b[10] = 0
b[19] = 0
eps = rnorm(1000)
y = x%*%b + eps
```

10b)

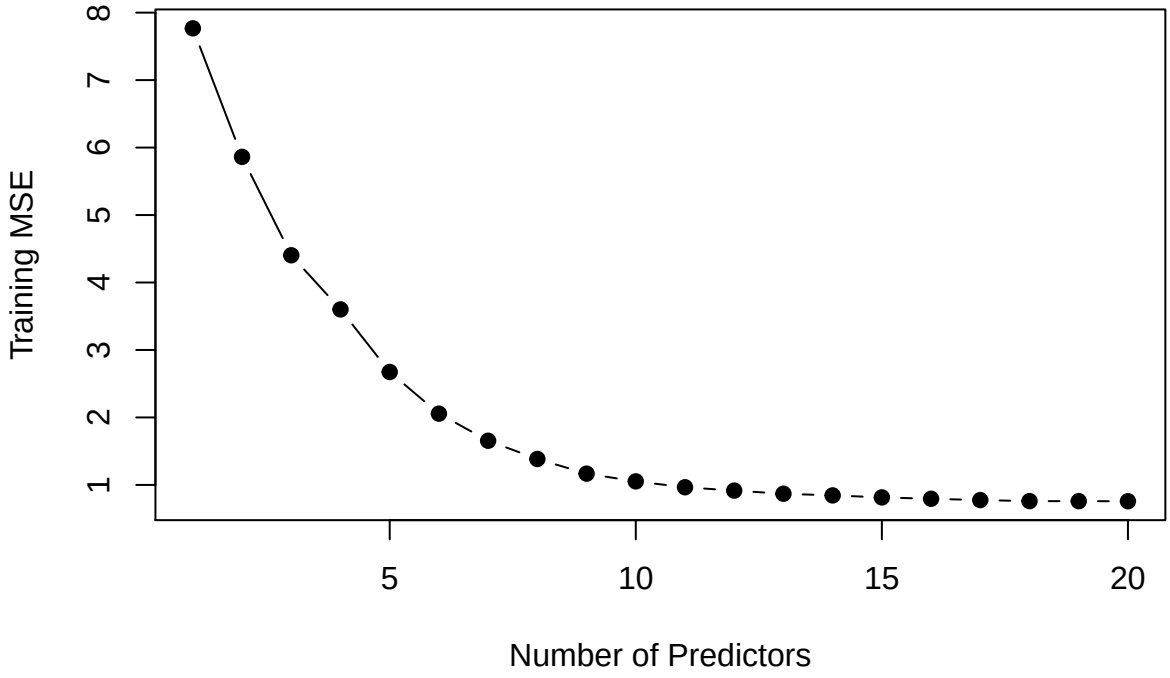
```
train = sample(seq(1000), 100, replace = FALSE)
test <-- train
x.train = x[train, ]
x.test = x[test, ]
y.train = y[train]
y.test = y[test]
```

10c)

```
library(leaps)
```

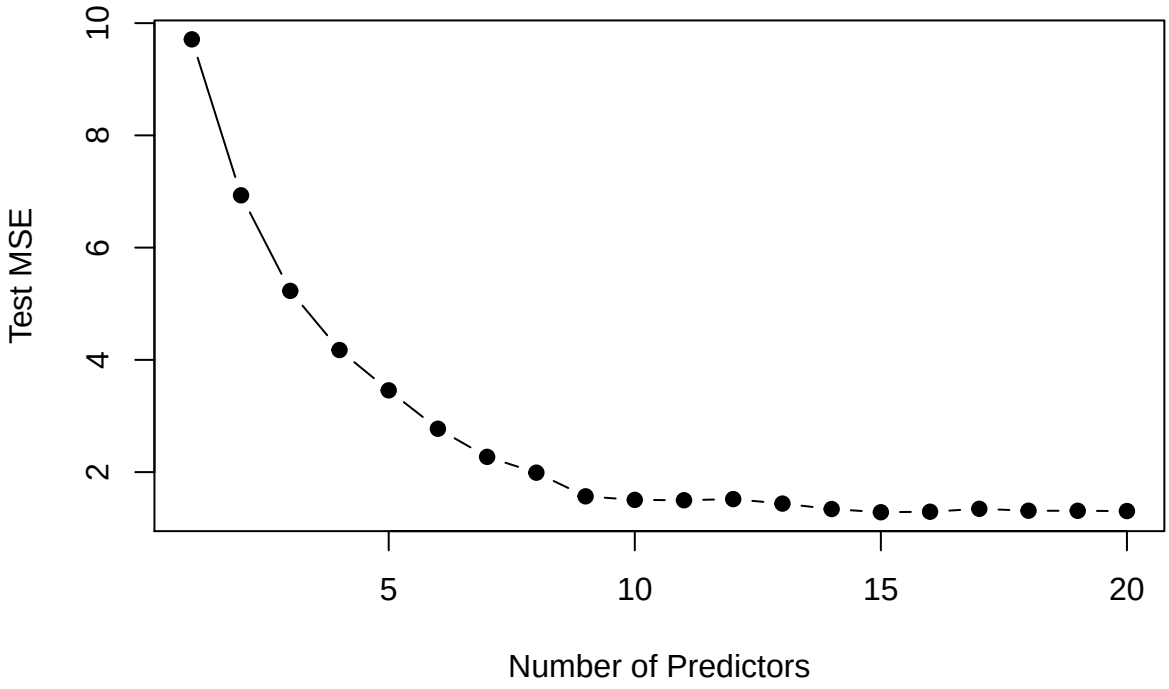
```
## Warning: package 'leaps' was built under R version 3.6.3
```

```
data.train = data.frame(y = y.train, x = x.train)
regfit.full = regsubsets(y ~ ., data = data.train, nvmax = 20)
train.mat = model.matrix(y ~ ., data = data.train, nvmax = 20)
val.errors = rep(NA, 20)
for(i in 1:20){
  coefi = coef(regfit.full, id = i)
  pred = train.mat[, names(coefi)]%*%coefi
  val.errors[i] = mean((pred - y.train)^2)
}
plot(val.errors, xlab = "Number of Predictors", ylab = "Training MSE", pch = 19, type = "b")
```



10d)

```
data.test = data.frame(y = y.test, x = x.test)
test.mat = model.matrix(y ~ ., data = data.test, nvmax = 20)
val.errors = rep(NA, 20)
for(i in 1:20){
  coefi = coef(regfit.full, id = i)
  pred = test.mat[, names(coefi)]%*%coefi
  val.errors[i] = mean((pred - y.test)^2)
}
plot(val.errors, xlab = "Number of Predictors", ylab = "Test MSE", pch = 19, type = "b")
```



10e)

```
which.min(val.errors)
```

```
## [1] 15
```

The model with the smallest test MSE is with 14 predictors. .

10f)

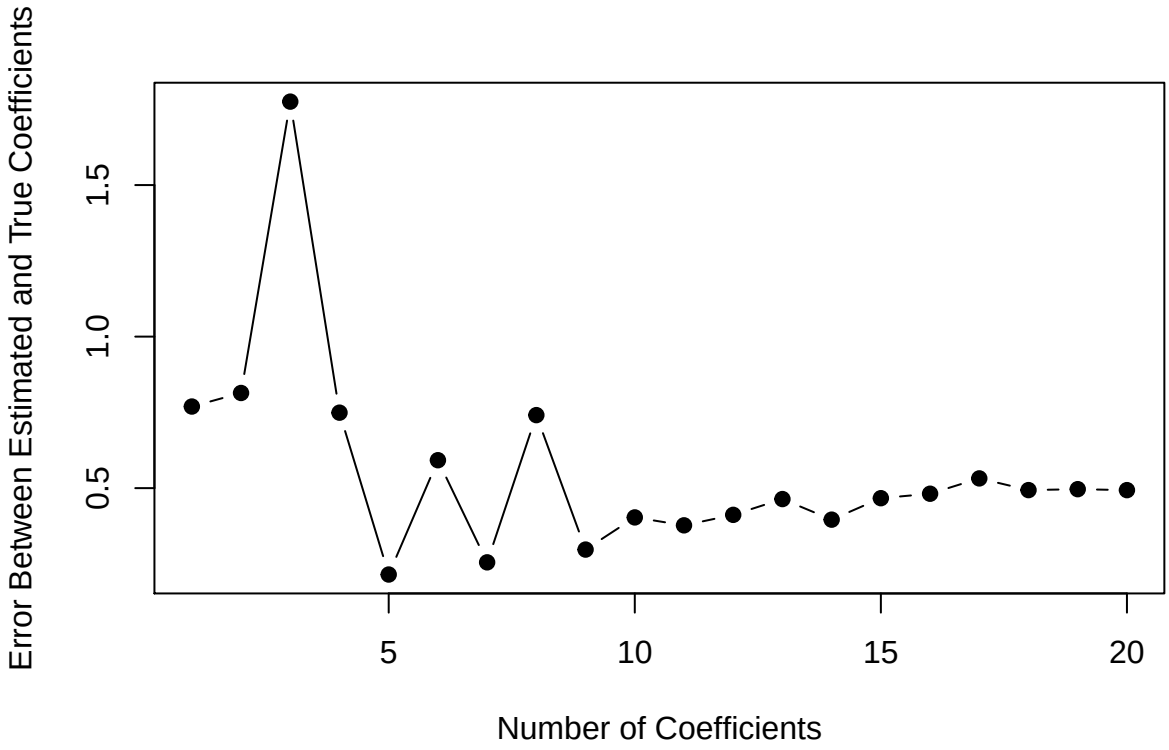
```
coef(regfit.full, which.min(val.errors))
```

```
##  (Intercept)          x.2          x.4          x.5          x.6          x.7
## -0.003933937  0.359127426  0.202707344  1.036265913 -0.253843053 -1.282753293
##          x.8         x.11         x.12         x.13         x.14         x.15
##  0.691581077  0.895769881  0.526887865 -0.207638251 -0.507929833 -0.892604795
##         x.16         x.17         x.18         x.20
## -0.343062241  0.184479252  1.646950451 -1.060191640
```

The true model for the sum of squares is minimized with the above coefficients. In part a), we generate the coefficients based on a normal distribution. If you observe the coefficient values of this model, we can see they're roughly normal-ish, so this model seems to be close to the true model. It's not terribly far off, but it's definitely not perfect.

10g)

```
val.errors = rep(NA,20)
x_cols = colnames(x,do.NULL = FALSE, prefix = 'x.')
for(i in 1:20){
    coefi = coef(regfit.full, id=i)
    val.errors[i]= sqrt(sum((b[x_cols%in%names(coefi)]-coefi[names(coefi)%in%x_cols])^2) + sum(b[!(x_co
}
plot(val.errors, xlab = "Number of Coefficients", ylab = "Error Between Estimated and True Coefficients
```



Compared to the test MSE graph in part d), the graphs look significantly different. In fact, we can observe when the model has 3 or 7 coefficients, it's performing the most optimally. Regardless, once we reach a large number of coefficients (i.e. 15+), the error seems to level out and there begins to be no more additional benefit in making the model more complex (this appears in both charts). As such, it may be worth investigating a model with 4 or 7 coefficients/features since the graph in part d) generally indicates "more features is less test MSE". However, with that being said, after analyzing both charts, a model with 7 predictors proves to be a good tradeoff between model complexity, fit, and interpretability. It has the lowest error rate in this chart, and close to the smallest test MSE in the chart in part d).

Problem 4)

4a)

Write code that will generate a dataset with n = 100 observations on p = 10 features. That is, the first 5 features should have a coefficient of 1 and the rest set equal to 0. The error term ε should be independently sampled from a normal distribution with mean 0 and variance σ^2. For now, you can choose any reasonable value of σ that you like. The dataset created here will be our training set.

```
set.seed(1)
eps <- rnorm(100, mean = 0, sd =  2)
x <- matrix(rnorm(10*100), ncol=10)
betas <- sample(-5:5, 10, replace=TRUE)
betas[seq(1,5)] <- 1
betas[seq(6,10)] <- 0


y <- x %*% betas + eps
training_set = data.frame(y,x)
```

4b)

Generate a large test set – let's say with 10,000 observations – in the same fashion asin part (a).

```
set.seed(1)
test_eps <- rnorm(10000, mean = 0, sd = 2)
test_x <- matrix(rnorm(10*10000), ncol=10)
test_betas <- sample(-5:5, 10, replace=TRUE)
test_betas[seq(1,5)] <- 1
test_betas[seq(6,10)] <- 0


test_y <- test_x %*% test_betas + test_eps
test_set = data.frame(test_y,test_x)
```

4c)

```
library(glmnet)
train.mat = model.matrix(y ~ . ,data = training_set)
test.mat = model.matrix(test_y ~ ., data = test_set)
grid = 10 ^ seq(4, -2, length=100)
mod.lasso <- cv.glmnet(train.mat, training_set[, 'y'], alpha = 1, lambda = grid, thresh = 1e12)
lambda.best <- mod.lasso$lambda.min
lasso.pred <- predict(mod.lasso, newx = test.mat, s=lambda.best)
error_lasso.1 <- mean((test_set[, 'test_y'] - lasso.pred)^2)
coef.lasso <- predict(mod.lasso, type="coefficients", s=lambda.best)


coef.list <- c()
for (i in 1:10){
  curr_str = paste0("X", toString(i))
  if(coef.lasso[curr_str,]!=0){
```

```
      coef.list <- c(coef.list,curr_str)
  }
}
coef.list
```

```
## [1] "X1"  "X2"  "X3"  "X4"  "X5"  "X6"  "X8"  "X9"  "X10"
```

4d)

```
fit.ols <- lm(y ~ ., data = training_set[c(coef.list, 'y')])
pred.ols <- predict(fit.ols, test_set[c(coef.list, 'test_y')])
err_ols.1 <- mean((test_set[,'test_y'] - pred.ols)^2)
err_ols.1
```

```
## [1] 4.393858
```

4e)

```
err_lasso <- c()
err_ols <- c()
for (i in 1:1000){
  #part a:
  eps <- rnorm(100, mean = 0, sd = 2)
  x <- matrix(rnorm(10*100, mean = 0, sd = 2), ncol=10)
  betas <- sample(-5:5, 10, replace=TRUE)
  betas[seq(1,5)] <- 1
  betas[seq(6,10)] <- 0

  y <- x %*% betas + eps
  training_set = data.frame(y,x)

  #part c:
  train.mat = model.matrix(y ~ . ,data = training_set)
  test.mat = model.matrix(test_y ~ ., data = test_set)
  grid = 10 ^ seq(4, -2, length=100)
  mod.lasso <- cv.glmnet(train.mat, training_set[, 'y'], alpha = 1, lambda = grid, thresh = 1e12)
  lambda.best <- mod.lasso$lambda.min
  lasso.pred <- predict(mod.lasso, newx = test.mat, s=lambda.best)
  error_lasso.1 <- mean((test_set[, 'test_y'] - lasso.pred)^2)
  err_lasso <- c(err_lasso, error_lasso.1)
  coef.lasso <- predict(mod.lasso, type="coefficients", s=lambda.best)


  coef.list <- c()
  for (i in 1:10){
    curr_str = paste0("X", toString(i))
    if(coef.lasso[curr_str,]!=0){
        coef.list <- c(coef.list,curr_str)
    }
```

```
  }

  #part d:
  fit.ols <- lm(y ~ ., data = training_set[c(coef.list, 'y')])
  pred.ols <- predict(fit.ols, test_set[c(coef.list, 'test_y')])
  err_ols.1 <- mean((test_set[,'test_y'] - pred.ols)^2)

  err_ols <- c(err_ols, err_ols.1)
}
err_ols_avg <- mean(err_ols)
err_lasso_avg <- mean(err_lasso)
```

4f)

```
#choose sigma from 0.015 to 2 in increments of 0.10 (small sigma)
err_lasso_small_sigma <- c()
err_ols_small_sigma <- c()
err_lasso_small_sigma_avg <- c()
err_ols_small_sigma_avg <- c()

for (sigma in seq(from=0.10,to=2,by=0.10)){
  err_curr_lasso_small <- c()
  err_curr_ols_small <- c()
  #part e:
  for (i in 1:1000){
    #part a:
    eps <- rnorm(100, mean = 0, sd = sigma)
    x <- matrix(rnorm(10*100), ncol=10)
    betas <- sample(-5:5, 10, replace=TRUE)
    betas[seq(1,5)] <- 1
    betas[seq(6,10)] <- 0

    y <- x %*% betas + eps
    training_set = data.frame(y,x)

    #part b:
    test_eps <- rnorm(10000, mean = 0, sd = sigma)
    test_x <- matrix(rnorm(10*10000), ncol=10)
    test_betas <- sample(-5:5, 10, replace=TRUE)
    test_betas[seq(1,5)] <- 1
    test_betas[seq(6,10)] <- 0


    test_y <- test_x %*% test_betas + test_eps
    test_set = data.frame(test_y,test_x)

    #part c:
    train.mat = model.matrix(y ~ . ,data = training_set)
    test.mat = model.matrix(test_y ~ ., data = test_set)
    grid = 10 ^ seq(4, -2, length=100)
    mod.lasso <- cv.glmnet(train.mat, training_set[, 'y'], alpha = 1, lambda = grid, thresh = 1e12)
```

```
    lambda.best <- mod.lasso$lambda.min
    lasso.pred <- predict(mod.lasso, newx = test.mat, s=lambda.best)
    error_lasso.1 <- mean((test_set[, 'test_y'] - lasso.pred)^2)
    err_curr_lasso_small <- c(err_curr_lasso_small, error_lasso.1)
    coef.lasso <- predict(mod.lasso, type="coefficients", s=lambda.best)

    coef.list <- c()
    for (i in 1:10){
      curr_str = paste0("X", toString(i))
      if(coef.lasso[curr_str,]!=0){
          coef.list <- c(coef.list,curr_str)
      }
    }

    #part d:
    fit.ols <- lm(y ~ ., data = training_set[c(coef.list, 'y')])
    pred.ols <- predict(fit.ols, test_set[c(coef.list, 'test_y')])
    err_ols.1 <- mean((test_set[,'test_y'] - pred.ols)^2)

    err_curr_ols_small <- c(err_curr_ols_small, err_ols.1)
  }
  err_ols_small_sigma_avg <- mean(err_curr_ols_small)
  err_lasso_small_sigma_avg <- mean(err_curr_lasso_small)
  err_lasso_small_sigma <- c(err_lasso_small_sigma, err_lasso_small_sigma_avg)
  err_ols_small_sigma <- c(err_ols_small_sigma, err_ols_small_sigma_avg)
}
err_ols_small_sigma
```

```
##  [1] 0.01108230 0.04486767 0.10107077 0.17983242 0.28000392 0.40384356
##  [7] 0.54982819 0.71652147 0.90829293 1.11963827 1.35494599 1.61260659
## [13] 1.89531768 2.19082608 2.52633375 2.86814960 3.23955036 3.62789916
## [19] 4.04771406 4.47189741
```

```
err_lasso_small_sigma
```

```
##  [1] 0.01136864 0.04441703 0.09999478 0.17777552 0.27721429 0.39969298
##  [7] 0.54534403 0.70994096 0.90086322 1.11120833 1.34407773 1.59815033
## [13] 1.88040900 2.17393550 2.50467838 2.84874969 3.21285747 3.59733642
## [19] 4.01651504 4.44189973
```

```
#choose sigma from 2 to 6 in increments of 0.5 (large sigma)
err_lasso_large_sigma <- c()
err_ols_large_sigma <- c()
err_lasso_large_sigma_avg <- c()
err_ols_large_sigma_avg <- c()

for (sigma in seq(from=2,to=6,by=0.5)){
  err_curr_lasso_large <- c()
  err_curr_ols_large <- c()
  #part e:
  for (i in 1:1000){
    #part a:
```

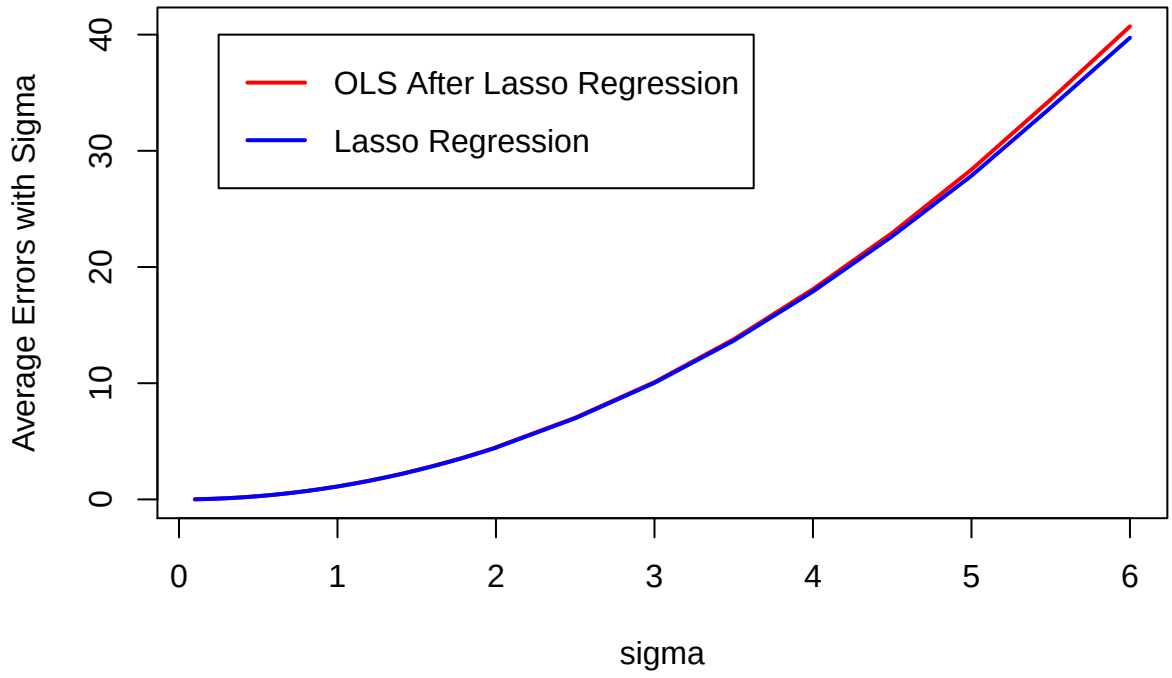
```
    eps <- rnorm(100, mean = 0, sd = sigma)
    x <- matrix(rnorm(10*100), ncol=10)
    betas <- sample(-5:5, 10, replace=TRUE)
    betas[seq(1,5)] <- 1
    betas[seq(6,10)] <- 0

    y <- x %*% betas + eps
    training_set = data.frame(y,x)

    #part b:
    test_eps <- rnorm(10000, mean = 0, sd = sigma)
    test_x <- matrix(rnorm(10*10000), ncol=10)
    test_betas <- sample(-5:5, 10, replace=TRUE)
    test_betas[seq(1,5)] <- 1
    test_betas[seq(6,10)] <- 0


    test_y <- test_x %*% test_betas + test_eps
    test_set = data.frame(test_y,test_x)

    #part c:
    train.mat = model.matrix(y ~ . ,data = training_set)
    test.mat = model.matrix(test_y ~ ., data = test_set)
    grid = 10 ^ seq(4, -2, length=100)
    mod.lasso <- cv.glmnet(train.mat, training_set[, 'y'], alpha = 1, lambda = grid, thresh = 1e12)
    lambda.best <- mod.lasso$lambda.min
    lasso.pred <- predict(mod.lasso, newx = test.mat, s=lambda.best)
    error_lasso.1 <- mean((test_set[, 'test_y'] - lasso.pred)^2)
    err_curr_lasso_large <- c(err_curr_lasso_large, error_lasso.1)
    coef.lasso <- predict(mod.lasso, type="coefficients", s=lambda.best)

    coef.list <- c()
    for (i in 1:10){
      curr_str = paste0("X", toString(i))
      if(coef.lasso[curr_str,]!=0){
          coef.list <- c(coef.list,curr_str)
      }
    }

    #part d:
    fit.ols <- lm(y ~ ., data = training_set[c(coef.list, 'y')])
    pred.ols <- predict(fit.ols, test_set[c(coef.list, 'test_y')])
    err_ols.1 <- mean((test_set[,'test_y'] - pred.ols)^2)

    err_curr_ols_large <- c(err_curr_ols_large, err_ols.1)
  }
  err_ols_large_sigma_avg <- mean(err_curr_ols_large)
  err_lasso_large_sigma_avg <- mean(err_curr_lasso_large)
  err_lasso_large_sigma <- c(err_lasso_large_sigma, err_lasso_large_sigma_avg)
  err_ols_large_sigma <- c(err_ols_large_sigma, err_ols_large_sigma_avg)
}
err_ols_large_sigma
```

```
## [1]  4.491929  7.032116 10.099339 13.779327 18.095268 22.935408 28.381384
## [8] 34.410088 40.707459
```

```
err_lasso_large_sigma
```

```
## [1]  4.455939  6.980260 10.029035 13.660305 17.893625 22.647426 27.852550
## [8] 33.718874 39.730068
```

```
sigmas <- c(seq(from=0.10,to=2,by=0.10), seq(from=2,to=6,by=0.5))
errors_ols <- c(err_ols_small_sigma, err_ols_large_sigma)
errors_lasso <- c(err_lasso_small_sigma, err_lasso_large_sigma)
plot(sigmas, errors_ols, type = "l", col = "red", lwd = 2, xlab = "sigma", ylab = "Average Errors with S
lines(sigmas, errors_lasso, col="blue", lwd=2)
legend(0.25, 40, c("OLS After Lasso Regression","Lasso Regression"), lwd=c(2,2), col=c("red","blue"), y
```



For small values of σ, based on what we know from SNRs and DoFs, Lasso has few DoF, while OLS after Lasso will have higher DoF. As a result, at low SNRs, as we can see from the plot, it becomes difficult to see the differences in performances between OLS after lasso and Lasso regression, however as σ increases, at higher SNRs, it will be more clear to see which model will perform better, as it becomes more apparent when the model is overfitting. In our case, we can see the slight divergence with OLS after Lasso and Lasso. This gap will continually grow for larger σ. Notably, at low SNRs, we can see that procedures with highe DoFs tend to do poorly. All methods begin to do well with procedures that have higher DoFs doing the best at high SNRs. At low SNRs, it's hard to see the real signal, and thus high DoF procedures are almost guaranteed to overfit, no matter how it's validated. Since regularized procedures like Lasso are designed to increase bias/stability, they excel in these settings.