DATA 558 Homework 3

Shrusti Ghela

5/11/2022

On this assignment, some of the problems involve random number generation. Be sure to set a random seed (using the command set.seed()) before you begin.

1. In this problem, we'll see a (very!!) simple simulated example where a least squares linear model is "too flexible".

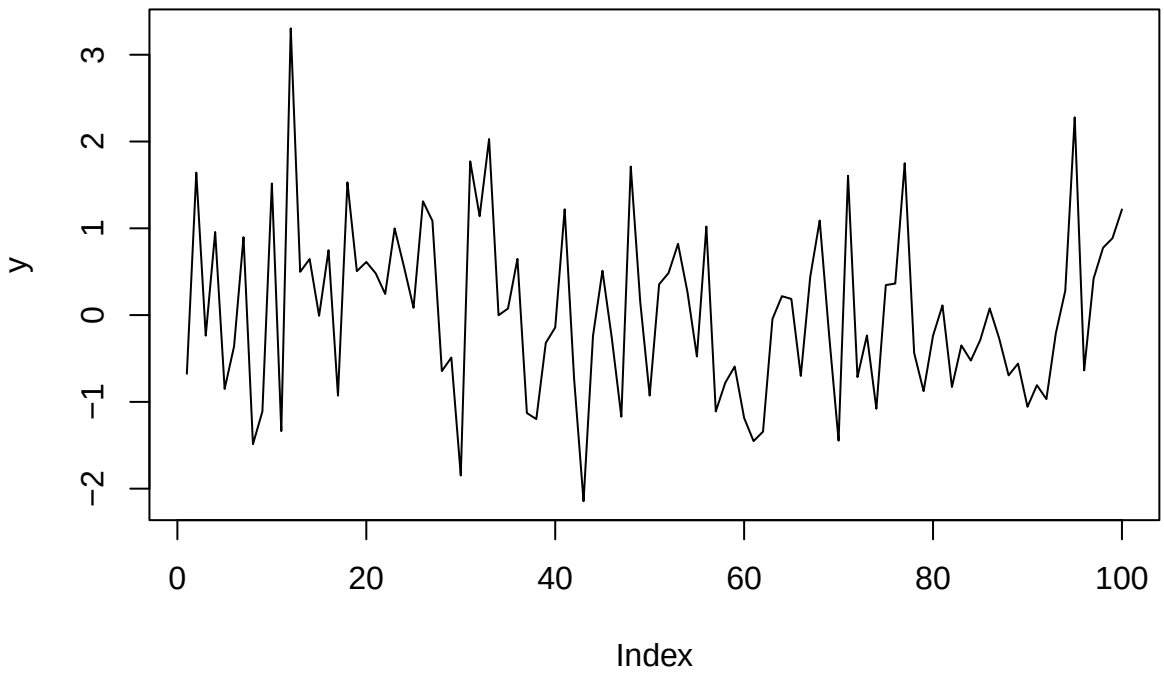
a. First, generate some data with n = 100 and p = 10,000 features, and a quantitative response, using the following R commands:

```
set.seed(1023)
y <- rnorm(100)
x <- matrix(rnorm(10000*100), ncol=10000)
```

Write out an expression for the model corresponding to this data generation procedure. For instance, it might look something like $Y = 2X_1 + 3X_2 + \epsilon, \epsilon \sim \mathcal{N}(0, 1)$

```
plot(y, type='line')
```

```
## Warning in plot.xy(xy, type, ...): plot type 'line' will be truncated to first
## character
```



For the above data generation procedure, we know that $y_1, y_2, ..., y_{100}$ i.i.d $\mathcal{N}(0,1)$ and $x_1, x_2, ..., x_{10000}$ are unrelated to y

So, the corresponding model for this data generating procedure would look like this: $Y = \epsilon$ where $\epsilon \sim \mathcal{N}(0,1)$

b. What is the value of the irreducible error?

$\mathbb{E}[y_0 - \hat{f}(X_0)]^2 = var(\epsilon) + var(\hat{f}(X_0)) + Bias^2(\hat{f}(X_0))$

$reducible\ error = var(\hat{f}(X_0)) + Bias^2(\hat{f}(X_0))$

$irreducible\ error = var(\epsilon) = 1\ (\because \epsilon \sim \mathcal{N}(0,1))$

c. Consider a very simple model-fitting procedure that just predicts 0 for every observation. That is, $\hat{f}(x) = 0$ for all x.

i. What is the bias of this procedure?

Here, for $Y = f(X) + \epsilon$ So, $f(X) = 0 \cdot X$

Hence,

$Bias^2(\hat{f}(X_0)) = (f(X_0) - \mathbb{E}\hat{f}(X_0))^2 = (0 - 0) = 0$

$Bias = 0$

ii. What is the variance of this procedure?

$var(\hat{f}(X_0)) = \mathbb{E}(\mathbb{E}\hat{f}(X_0) - \hat{f}(X_0))^2 = \mathbb{E}(0-0)^2 = 0$

Answering both i. and ii., theoretical bias & variance is 0 if the underlying sampling distribution has a conditional variance of 0 (conditional with respect to your predictors).

iii. What is the expected prediction error of this procedure?

Expected prediction error $= \mathbb{E}[y_0 - \hat{f}(X_0)]^2 = var(\epsilon) + var(\hat{f}(X_0)) + Bias^2(\hat{f}(X_0))$

From i. and ii.,

$\mathbb{E}[y_0 - \hat{f}(X_0)]^2 = var(\epsilon) + 0 + 0$

$\mathbb{E}[y_0 - \hat{f}(X_0)]^2 = 1$

iv. Use the validation set approach to estimate the test error of this procedure. What answer do you get?

```
X <- as.data.frame(x)
data <- cbind(X, y)

library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
set.seed(123)
training.samples <- data$y %>%
  createDataPartition(p = 0.8, list = FALSE)

train.data.1  <- data[training.samples, ]
test.data.1 <- data[-training.samples, ]

predi <- c(seq(0,0,length.out=20))

mean((test.data.1$y-predi)^2)
```

```
## [1] 0.784222
```

v. Comment on your answers to (iii) and (iv). Do your answers agree with each other? Explain.

Theoretical expected prediction error (iii) = 1. Estimated Test error (iv) = 0.784222. This slight difference between expected and estimated error is because we are trying to estimate the test error with our validation data. This is not the actual value. We are just trying to estimate it. The estimate won't always represent true case.

For our case here, due to random sampling of data, the estimated error varies. If I set some other seed, I get a different estimate of test error.

```
set.seed(1235)
training.samples.c <- data$y %>%
  createDataPartition(p = 0.8, list = FALSE)

train.data.1c  <- data[training.samples.c, ]
test.data.1c <- data[-training.samples.c, ]

predic <- c(seq(0,0,length.out=20))

mean((test.data.1c$y-predic)^2)
```

```
## [1] 0.9431362
```

Here, for a different seed, I get estimated test error much closer to our expected prediction error. However, for any random seed, this approach is a decent way to estimate test error.

d. Now use the validation set approach to estimate the test error of a least squares linear model using $X_1, ..., X_{10000}$ to predict Y . What is the estimated test error?

Hint: If you fit a least squares linear model to predict Y using $X_1, ..., X_p$ where $p \geq n$, then only the first $n - 1$ coefficients will be assigned values. The rest will show up as NA because those coefficients aren't needed to obtain a perfect (i.e. zero) residual sum of squares on the training data. You can see all of the coefficient values by applying the coef() command to the output of the linear model.

```
lm.1d <- lm(y ~ ., data =  train.data.1)
#summary(lm.1d)
```

```r
pred.1d <- predict.lm(lm.1d, newdata=test.data.1)
```

```
## Warning in predict.lm(lm.1d, newdata = test.data.1): prediction from a rank-
## deficient fit may be misleading
```

```r
pred.1d.train <- predict.lm(lm.1d, newdata=train.data.1)
```

```
## Warning in predict.lm(lm.1d, newdata = train.data.1): prediction from a rank-
## deficient fit may be misleading
```

```r
#pred.30
#pred.1d
mean((train.data.1$y-pred.1d.train)^2)
```

```
## [1] 5.378148e-28
```

```r
mean((test.data.1$y-pred.1d)^2)
```

```
## [1] 772.4449
```

Training Error = 5.378148e-28 Estimated Test Error = 772.4449

e. Comment on your answers to (c) and (d). Which of the two procedures has a smaller estimated test error? higher bias? higher variance? In answering this question, be sure to think carefully about how the data were generated.

For (d.), Here, we see that the training error is extremely low, almost 0. But our estimated test error is extremely high. This happens due to overfitting. The model tries to fit a model on the predictors and find a relationship between y and X when none exists. Also, one other factor that leads to overfitting is that $p \gg n$. When there are more variables than data points, the problem may not have a unique solution unless it's further constrained. That is, there may be multiple (perhaps infinitely many) solutions that fit the data equally well. For example, when there are more variables than data points, standard least squares regression has infinitely many solutions that achieve zero error on the training data. Such a model would certainly overfit because it's 'too flexible' for the amount of training data. As model flexibility increases and the amount of training data shrinks, it becomes increasingly likely that the model will be able to achieve a low error by fitting random fluctuations in the training data that don't represent the true, underlying distribution. Performance will therefore be poor when the model is run on future data drawn from the same distribution. This leads to higher variance as compared to (c). Also, we can see that our model assumes linearity but it is not satisfied by our data generating function. This leads to higher bias as compared to (c).

For (c.), we see that the test error is pretty close to the expected prediction error. This tells us that our model is a good representation of the true data generating function.

2. In lecture during Week 5, we discussed "Option 1" and "Option 2": two possible ways to perform the validation set approach for a modeling strategy where you identify the q features most correlated with the response, and then fit a least squares linear model to predict the response using just those q features. If you missed that lecture, then please familiarize yourself with the lecture notes (posted on Canvas) before you continue. Here, we are going to continue to work with the simulated data from the previous problem, in order to illustrate the problem with Option 1.

For reference,

Option 1:

- Identify q features for which $|cor(\vec{x_i}, \vec{y})|$ is largest.
- Split n observations into training and validation sets.
- Fit Least Square model to predict $\vec{y}$ using q features on training set.
- Compute error of that model on validation set.

Option 2:

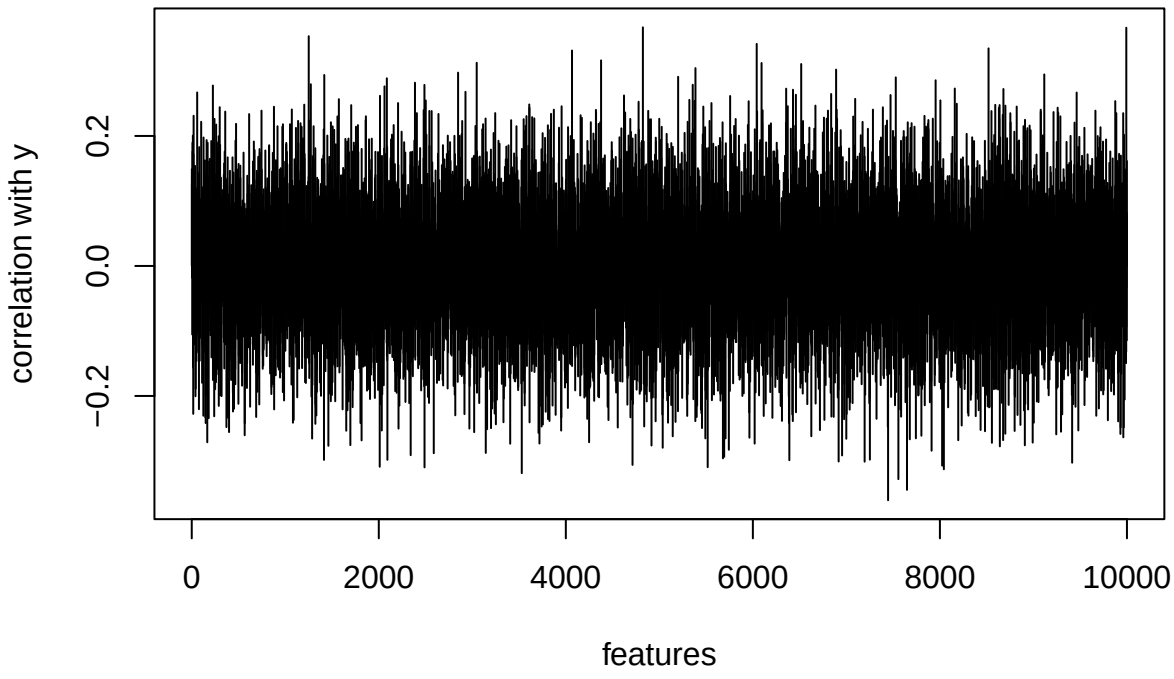
- Split n observations into training and validation sets.
- On training set, Identify q features for which $|cor(x_i^{train}, y^{train})|$ is largest.
- Fit Least Square model to predict $\vec{y}$ using q features on training set.
- Compute error of that model on validation set.

a. Calculate the correlation between each feature and the response. Make a histogram of these correlations. What are the values of the 10 largest absolute correlations?

```
X <- as.data.frame(x)
```

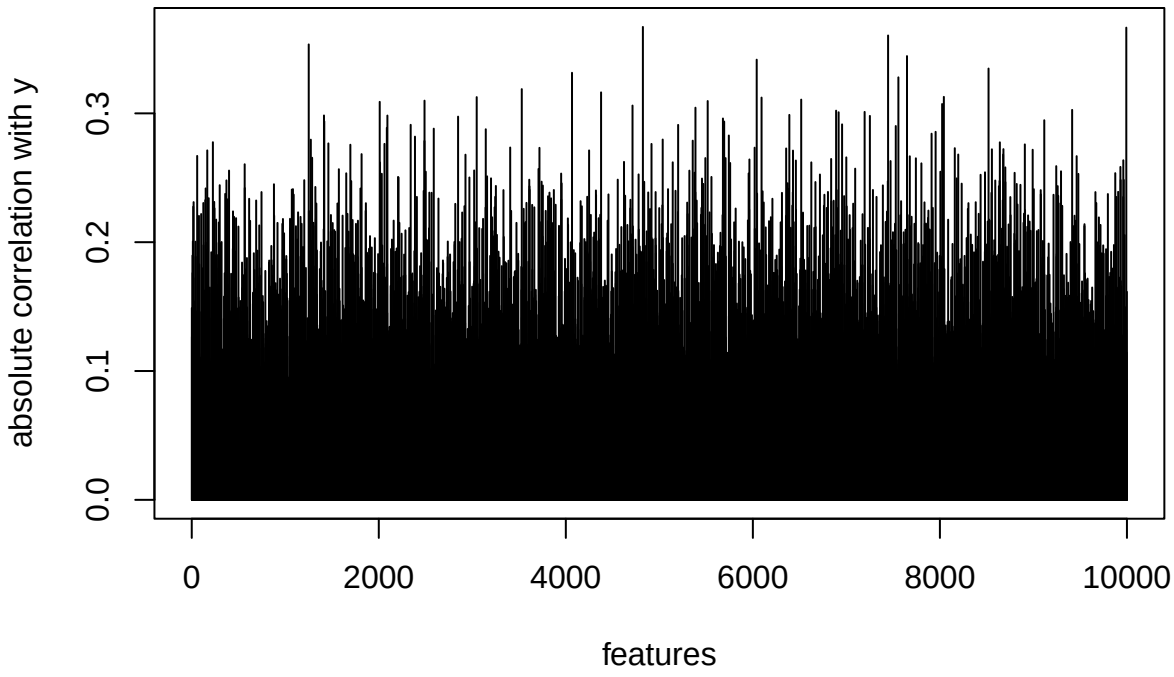
```
corr <- rep(NA, ncol(X))
for(i in 1:ncol(X)){
  corr[i] <- cor(X[ , i], y)
}
```

```
plot(corr, type='h', xlab='features', ylab='correlation with y')
```

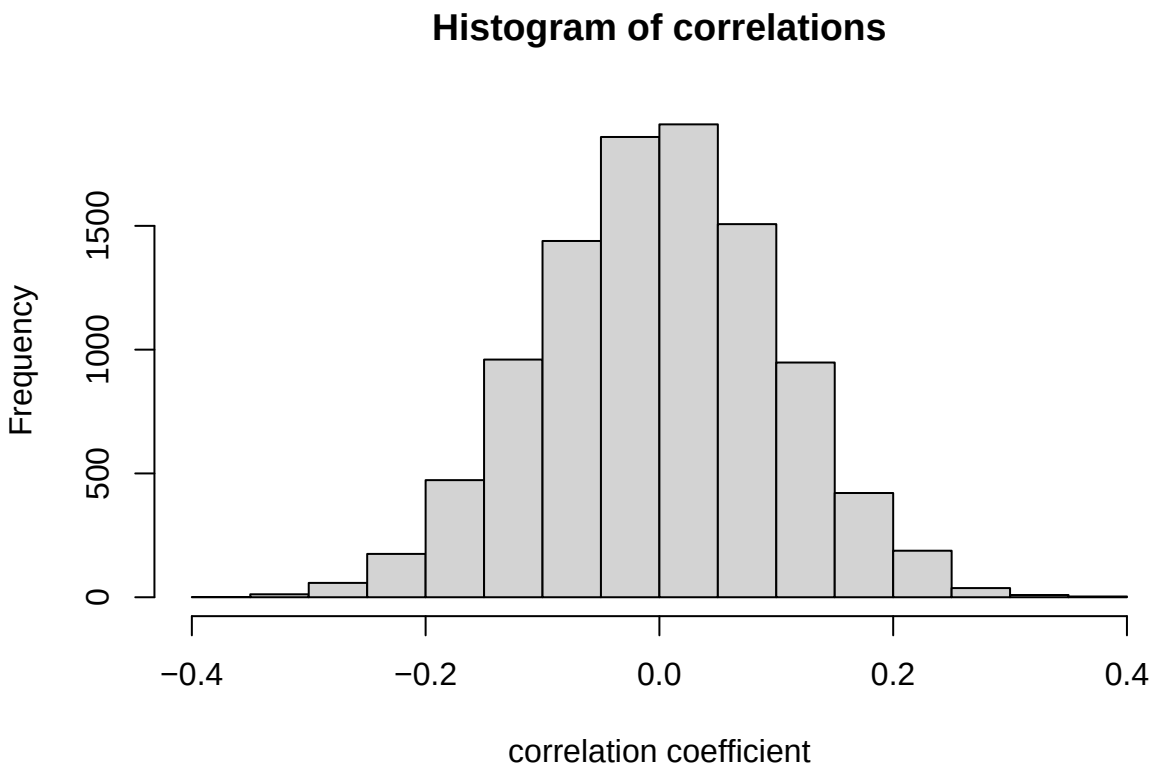


```r
corr1 <- abs(corr)
```

```r
plot(corr1, type='h', xlab='features', ylab='absolute correlation with y')
```




```r
hist(corr, xlab = 'correlation coefficient', main='Histogram of correlations')
```




```
tail(order(corr1), 10)
```

```
##  [1] 3529 7556 4066 8520 6041 7648 1251 7446 9993 4824
```

```
corr1[tail(order(corr1), 10)]
```

```
##  [1] 0.3188246 0.3279577 0.3314899 0.3347798 0.3416814 0.3444441 0.3535151
##  [8] 0.3604447 0.3665314 0.3670895
```

b. Now try out "Option 1" with q = 10. What is the estimated test error?

```
set.seed(123)
training.samples <- data$y %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data.opt1  <- data[training.samples, ]
test.data.opt1 <- data[-training.samples, ]

dim(train.data.opt1)
```

```
## [1]    80 10001
```

```
dim(test.data.opt1)
```

```
## [1]    20 10001
```

```
lm.opt1 <- lm(y ~ V3529 + V7556 + V4066 + V8520 + V6041 + V7648 + V1251 + V7446
              + V9993 + V4824, data =  train.data.opt1)
summary(lm.opt1)
```

```
##
## Call:
## lm(formula = y ~ V3529 + V7556 + V4066 + V8520 + V6041 + V7648 +
##     V1251 + V7446 + V9993 + V4824, data = train.data.opt1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.49714 -0.45817  0.00164  0.37464  2.00884
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.08014    0.07773   1.031 0.306099
## V3529       -0.13658    0.08236  -1.658 0.101773
## V7556       -0.22124    0.10741  -2.060 0.043196 *
## V4066        0.18261    0.08743   2.089 0.040436 *
## V8520        0.26950    0.07710   3.495 0.000832 ***
## V6041        0.24587    0.07612   3.230 0.001897 **
## V7648       -0.07172    0.09007  -0.796 0.428640
## V1251        0.19749    0.08997   2.195 0.031528 *
## V7446       -0.26246    0.09113  -2.880 0.005293 **
## V9993        0.08938    0.08540   1.047 0.298919
## V4824        0.22123    0.09053   2.444 0.017099 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6774 on 69 degrees of freedom
## Multiple R-squared:  0.6132, Adjusted R-squared:  0.5572
## F-statistic: 10.94 on 10 and 69 DF,  p-value: 6.935e-11
```

```
pred.opt1 <- predict(lm.opt1, newdata=test.data.opt1)
mean((test.data.opt1$y-pred.opt1)^2)
```

```
## [1] 0.4245297
```

Estimated Test Error = 0.4245297

c. Now try out "Option 2" with q = 10. What is the estimated test error?

```
set.seed(123)
training.samples.2 <- data$y %>%
  createDataPartition(p = 0.8, list = FALSE)
```

```
train.data.opt2  <- data[training.samples.2, ]
test.data.opt2 <- data[-training.samples.2, ]

X.opt2 <- train.data.opt2 %>% select(1:10000)
y.opt2 <- train.data.opt2 %>% select(10001:10001)
dim(train.data.opt2)
```

```
## [1]    80 10001
```

```
dim(test.data.opt2)
```

```
## [1]    20 10001
```

```
corr.2 <- rep(NA, ncol(X.opt2))
for(i in 1:ncol(X.opt2)) {
  corr.2[i] <- cor(X.opt2[ , i], y.opt2)
}
```

```
corr2 <- abs(corr.2)
```

```
tail(order(corr2), 10)
```

```
##  [1] 7556 6093 2588 4713 8672 9993 6041 1251 6917 7195
```

```
corr1[tail(order(corr2), 10)]
```

```
##  [1] 0.3279577 0.3122429 0.2881375 0.3060509 0.2325032 0.3665314 0.3416814
##  [8] 0.3535151 0.3008648 0.3011374
```

```
lm.opt2 <- lm(y ~ V7556 + V6093 + V2588 +V4713 +V8672 +V9993 +V6041 +V1251
              +V6917 +V7195, data =  train.data.opt2)
summary(lm.opt2)
```

```
##
## Call:
## lm(formula = y ~ V7556 + V6093 + V2588 + V4713 + V8672 + V9993 +
##     V6041 + V1251 + V6917 + V7195, data = train.data.opt2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.52345 -0.36490 -0.02397  0.30317  1.33942
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.08263    0.07325   1.128 0.263206
## V7556       -0.16804    0.09745  -1.724 0.089111 .
## V6093        0.27211    0.08568   3.176 0.002234 **
## V2588       -0.17842    0.08446  -2.113 0.038254 *
## V4713       -0.14584    0.07491  -1.947 0.055616 .
```

```
## V8672        0.14830    0.08166   1.816 0.073716 .
## V9993        0.05244    0.08505   0.617 0.539554
## V6041        0.19993    0.06873   2.909 0.004877 **
## V1251        0.20587    0.08440   2.439 0.017299 *
## V6917       -0.26050    0.07530  -3.460 0.000932 ***
## V7195       -0.17741    0.07320  -2.424 0.017987 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6422 on 69 degrees of freedom
## Multiple R-squared:  0.6523, Adjusted R-squared:  0.6019
## F-statistic: 12.95 on 10 and 69 DF,  p-value: 2.221e-12
```

```
pred.opt2 <- predict(lm.opt2, newdata=test.data.opt2)
mean((test.data.opt2$y-pred.opt2)^2)
```

```
## [1] 0.9650772
```

Estimated Test Error = 0.9650772

d. Comment on your results in (b) and (c). How does this relate to the discussion of Option 1 versus Option 2 from lecture? Explain how you can see that Option 1 gave you a useless (i.e. misleading, inaccurate, wrong) estimate of the test error.

We see that there is a significant difference between the estimation of test errors by option 1 and option 2. Now, why does this happen? There is a very straightforward explanation for this. Option 1 is the WRONG way of estimating the test error. Why? Because in the Option 1, we used all of the examples (data points) to screen the predictors instead of restricting themselves only to the training folds. Since we are working with a wide dataset — number of features (p) $\gg$ (n) number of examples — performing feature selection to narrow down to a subset of genes was a natural choice. This is in fact also a valuable step to prevent the model from over-fitting. But quite often, the screening is performed in the early stages of the project leading people to forget about it by the time they are modeling using cross-validation. If we use all of our examples to select our predictors, the model has "peeked" into the validation set even before predicting on it. Thus, the cross validation accuracy was bound to be much higher than the true model accuracy. Or the cross validation error was bound to be much lower than the true model error.

On the other-hand, for Option 2, if we run the same model, but use only the training folds to screen the predictors, we will have a much better representation of the true error of the model. In this case, we see that the test error is higher than what we estimated using Option 1.

3. In this problem, you will analyze a (real, not simulated) dataset of your choice with a quantitative response Y , and $p \geq 50$ quantitative predictors.

```
df <- read.csv("OnlineNewsPopularity.csv")
data = subset(df, select = -c(url, timedelta))
```

a. Describe the data. Where did you get it from? What is the meaning of the response, and what are the meanings of the predictors?

I downloaded this data from UCI machine learning repository. The link for the source is here

This data set summarizes a heterogeneous set of features about articles published by Mashable (www.mashable.com) over a period of two years. General characteristics of this data set are:

- Data Set Characteristics: Multivariate
- Attribute Characteristics: Integer, Real
- Number of Instances: 39797
- Number of Attributes: 61 (58 predictive attributes, 2 non-predictive, 1 goal field)
- Missing Values: No missing values

The response variable 'shares' refers to the number of shares of an article and the predictors describe some statistics about the article like Number of words in the title, Number of words in the content, Rate of unique words in the content, Rate of non-stop words in the content, Rate of unique non-stop words in the content, Number of links and many more.

b. Fit a least squares linear model to the data, and provide an estimate of the test error. (Explain how you got this estimate.)

I estimated the test error using the validation set approach. I split the data in train and validation set. I do this by randomly choosing a subset of numbers between 1 and n. After I split the data, I fit a least squares model on the training set. I calculated predicted values of shares using the model that I fit on training set. The Mean Squared Error can be calculated by the formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

where Y_i is the observed value and $\hat{Y}_i$ is the predicted value.

```
set.seed(1433)

x <- model.matrix(shares~. , data)[, -1]
y <- data$shares
```

```
set.seed(2)
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
y.test <- y[test]
```

```
lm.fit3b <- lm(y[train]~ x[train, ])
#summary(lm.fit3b)

predictions <- predict(lm.fit3b, newdata=as.data.frame(x[test,]))
```

```
## Warning in predict.lm(lm.fit3b, newdata = as.data.frame(x[test, ])): prediction
## from a rank-deficient fit may be misleading
```

```
mean((y.test-predictions)^2)
```

```
## [1] 170109026
```

There are two reasons this warning may occur:

- Two predictor variables are perfectly correlated.
- More model parameters than observations in the dataset.

Now, we are sure that $n > p$ for our dataset, then this warning would be raised because Two predictor variables might be perfectly correlated.

c. Fit a ridge regression model to the data, with a range of values of the tuning parameter λ. Make a plot like the left-hand panel of Figure 6.4 in the textbook.

I used the same training and test sets that we generated in the question above.

```
library(glmnet)
```

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

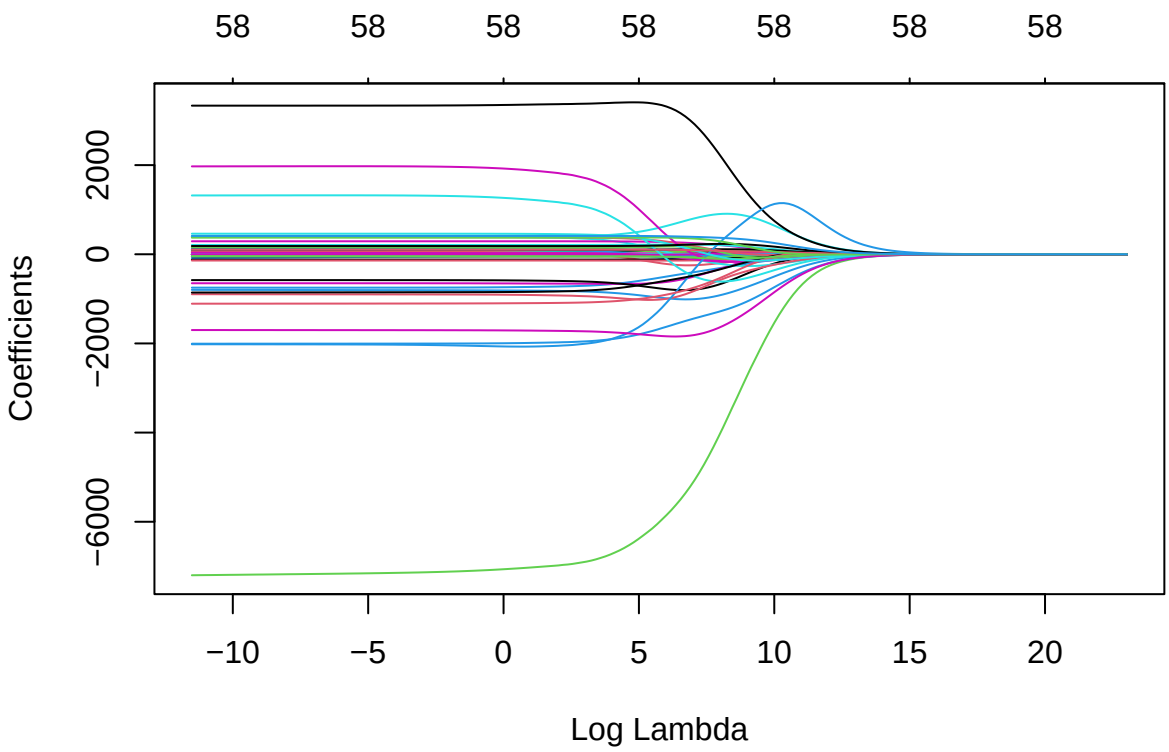
```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-4
```

```
grid <- 10^seq(10, -5, length = 200)
ridge.mod <- glmnet(x[train, ], y[train], alpha = 0, lambda = grid)
```

```
plot(ridge.mod, xvar="lambda")
```



d. What value of λ in the ridge regression model provides the smallest estimated test error? Report this estimate of test error. (Also, explain how you estimated test error.)

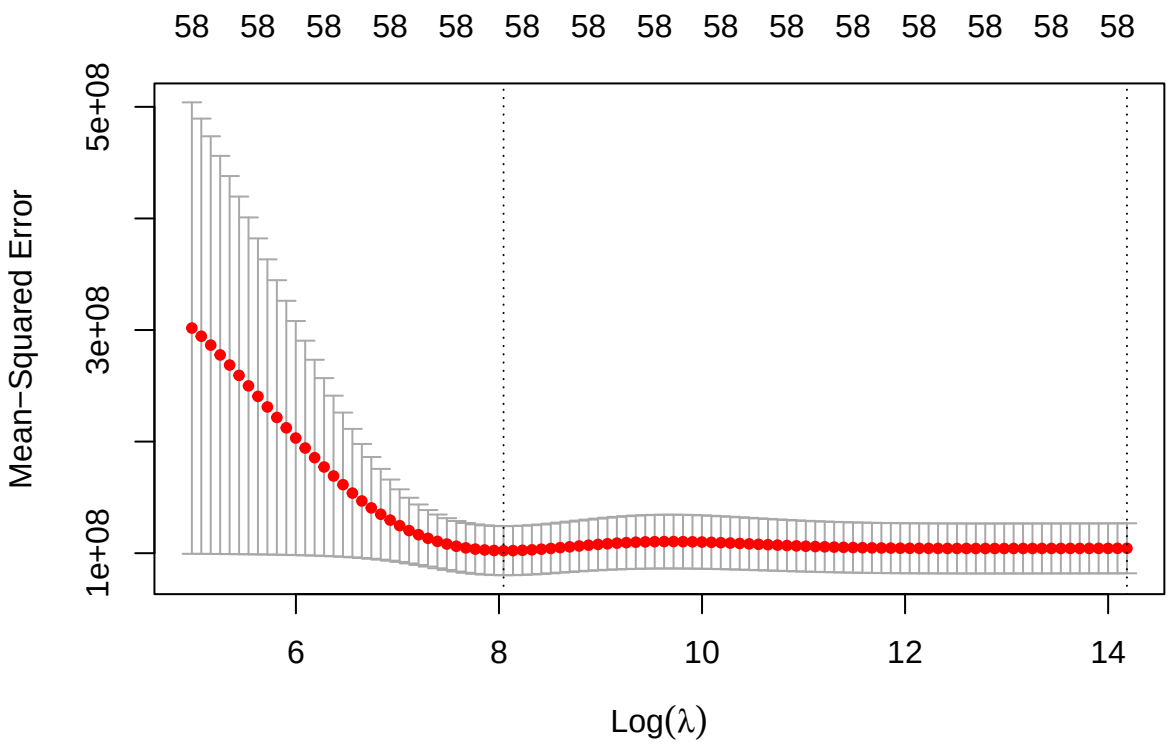
I split the dataset in the previous question to train the model and estimate the test error. Now I use cross-validation to choose the tuning parameter λ such that it provides the smallest estimated test error.

```
set.seed(2)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 0)
```

```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 3118.324
```

```
plot(cv.out)
```



To estimate this test error using the λ that has the least estimated test error.

I used the cross validation approach to estimate the best value of λ i.e, the value that has the least estimated error. I used the validation set appraoch to estimate the test error associated with this λ

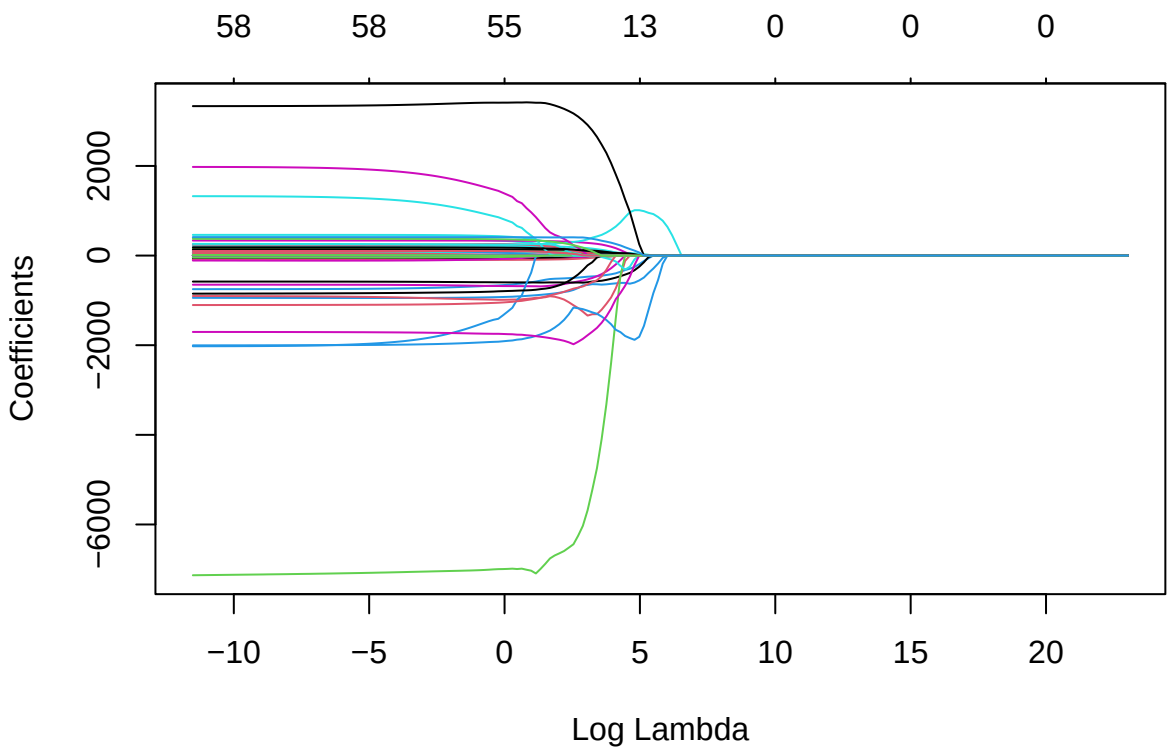
```
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x[test, ])
```

```
mean((ridge.pred - y.test)^2)
```

```
## [1] 163961514
```

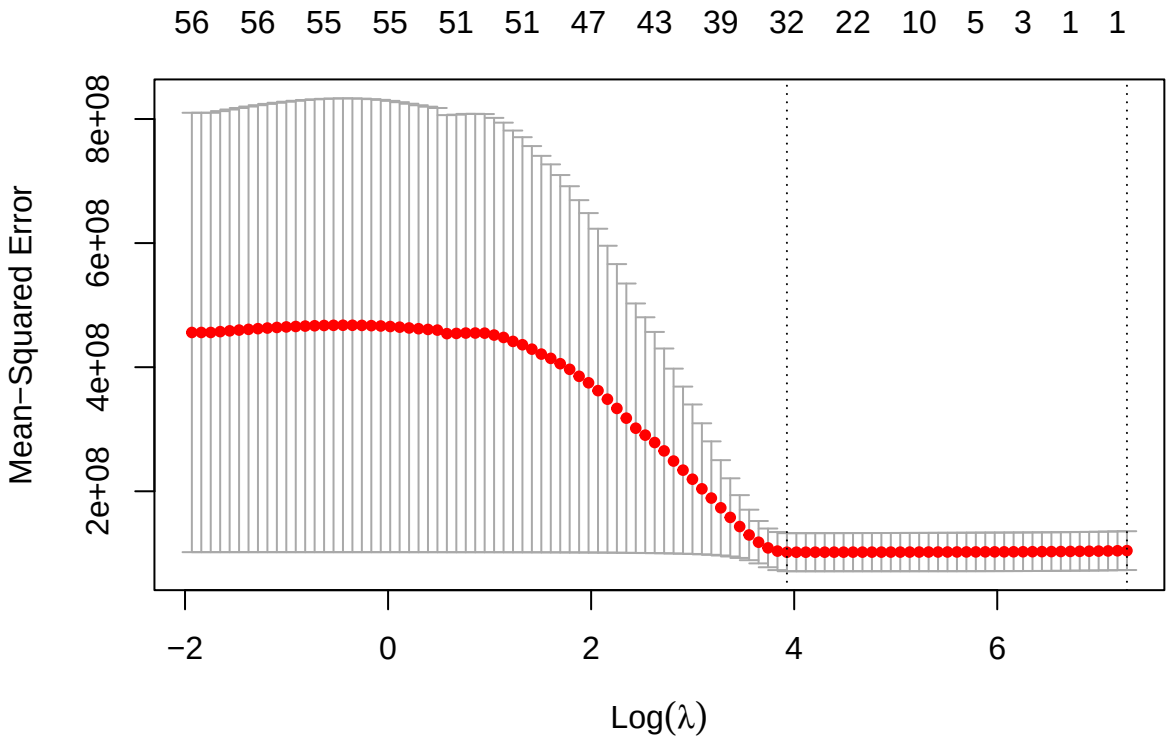
e. Repeat (c), but for a lasso model.

```
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1, lambda = grid)
plot(lasso.mod, xvar="lambda")
```



f. Repeat (d), but for a lasso model. Which features are included in this lasso model?

```
set.seed(1)
cv.out.las <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out.las)
```




```
bestlam.las <- cv.out.las$lambda.min
bestlam.las
```

```
## [1] 50.82092
```

```
lasso.pred <- predict(lasso.mod, s = bestlam.las, newx = x[test, ])
mean((lasso.pred - y.test)^2)
```

```
## [1] 163929192
```

We can see in this graph that as the value of lambda increases, the number of coefficient estimates grow sparse. This was however not observed in Ridge regression.

Finally, we refit our lasso regression model on the full data set, using the value of λ chosen by cross-validation, and examine the coefficient estimates. Here we see that 25 of the 58 coefficient estimates are exactly 0. So the lasso model with λ chosen by cross-validation contains 33 variables.

```
out.las <- glmnet(x, y, alpha = 1, lambda = grid)
rownames(coef(out.las, s = bestlam.las))[coef(out.las, s = bestlam.las)[,1]!= 0]
```

```
##  [1] "(Intercept)"                   "n_tokens_title"
##  [3] "n_tokens_content"              "num_hrefs"
##  [5] "num_self_hrefs"                "num_imgs"
##  [7] "average_token_length"          "num_keywords"
```

```
##  [9] "data_channel_is_lifestyle"     "data_channel_is_entertainment"
## [11] "kw_min_min"                    "kw_min_max"
## [13] "kw_max_max"                    "kw_min_avg"
## [15] "kw_max_avg"                    "kw_avg_avg"
## [17] "self_reference_min_shares"     "self_reference_max_shares"
## [19] "self_reference_avg_sharess"    "weekday_is_monday"
## [21] "weekday_is_tuesday"            "weekday_is_thursday"
## [23] "weekday_is_saturday"           "is_weekend"
## [25] "LDA_02"                        "LDA_03"
## [27] "global_subjectivity"           "global_rate_positive_words"
## [29] "min_positive_polarity"         "avg_negative_polarity"
## [31] "max_negative_polarity"         "title_sentiment_polarity"
## [33] "abs_title_subjectivity"        "abs_title_sentiment_polarity"
```

4. Consider using the Auto data set to predict mpg using polynomial functions of horsepower in a least squares linear regression.

a. Perform the validation set approach, and produce a plot like the one in the right-hand panel of Figure 5.2 of the textbook. Your answer won't look exactly the same as the results in Figure 5.2, since you'll be starting with a different random seed. Discuss your findings. What degree polynomial is best, and why?

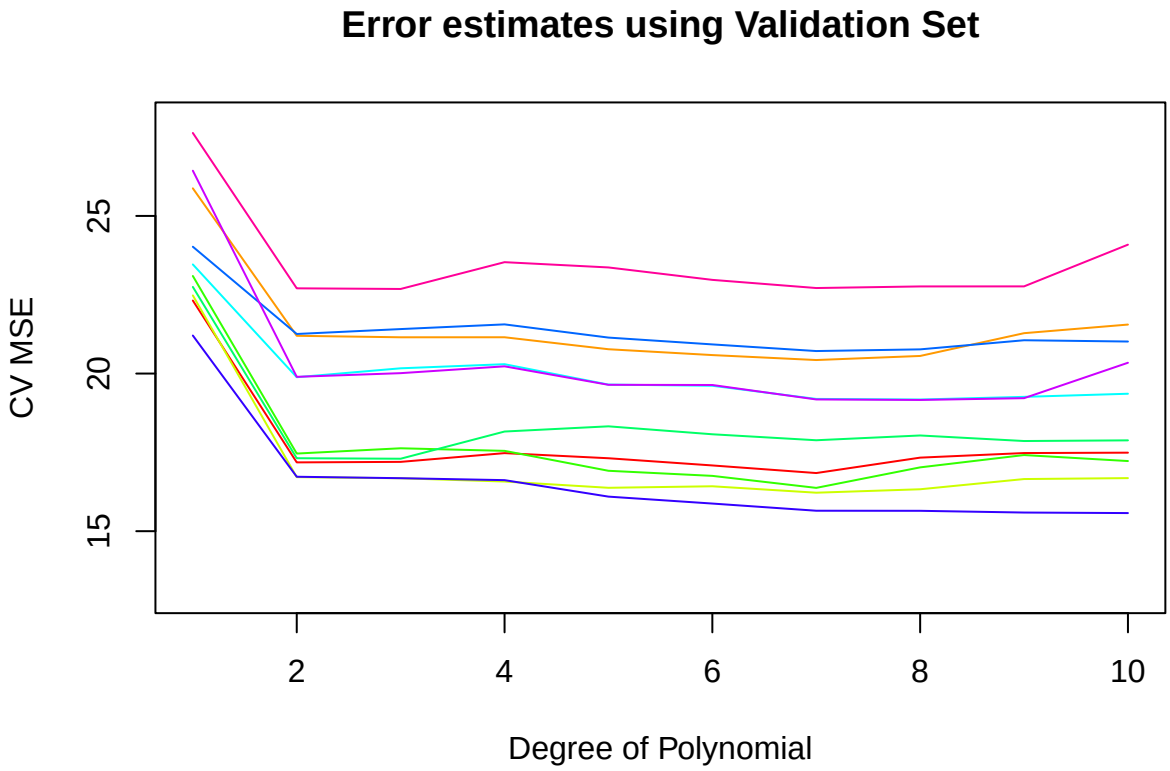
```
library(ISLR2)
data(Auto)

set.seed(14)

error <- matrix(0,10,10)

for(j in 1:10){
  train <- sample(1:nrow(Auto), nrow(Auto) / 2)
  for(i in 1:10){
    lm.fit4 <- lm(mpg ~ poly(horsepower, i), data=Auto, subset=train)
    error[j,i] <- mean((Auto$mpg - predict(lm.fit4, Auto))[-train]^2)
  }
}
plot(0,0, xlim=c(1,10), ylim = c(13,28), xlab='Degree of Polynomial',
     ylab='CV MSE', type='b', main='Error estimates using Validation Set')

color<- rainbow(10)
for(i in 1:10){
  lines(error[i, ], col=color[i])
}
```




```
mean.error <- apply(error, 2, FUN='mean')
mean.error
```

```
##  [1] 23.92699 19.03378 19.09106 19.31493 18.96007 18.79462 18.52011 18.68001
##  [9] 18.85812 19.12143
```

```
which.min(mean.error)
```

```
## [1] 7
```

The validation set approach was repeated 10 times for 10 different splits of train and validation set. We see that the errors in the validation set are highly variable and is dependent on how the data was split. I took average of errors for all validation sets for every degree polynomial. We see that 7 degree polynomial has the lowest error. Hence it looks like degree 7 polynomial is the best fit for our data.

b. Perform leave-one-out cross-validation, and produce a plot like the one in the left-hand panel of Figure 5.4 of the textbook. Discuss your findings. What degree polynomial is best, and why?

```
library(boot)
```

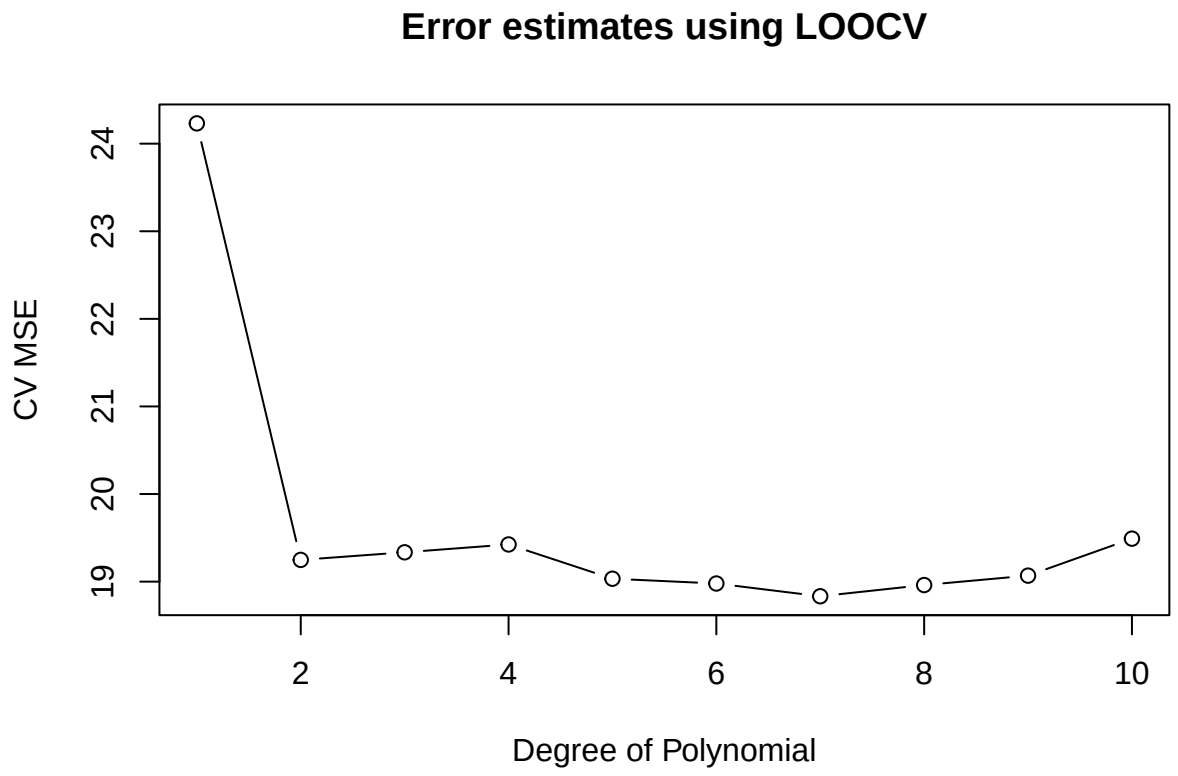
```
##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:lattice':
##
##     melanoma
```

```
loocv.error <- rep(0, 10)

for (i in 1:10) {
    glm.fit <- glm(mpg ~ poly(horsepower, i), data=Auto)
    loocv.error[i] <- cv.glm(Auto, glm.fit)$delta[1]
}

plot(loocv.error, type='b', xlab='Degree of Polynomial',ylab='CV MSE',
     main='Error estimates using LOOCV')
```




```
which.min(loocv.error)
```

```
## [1] 7
```

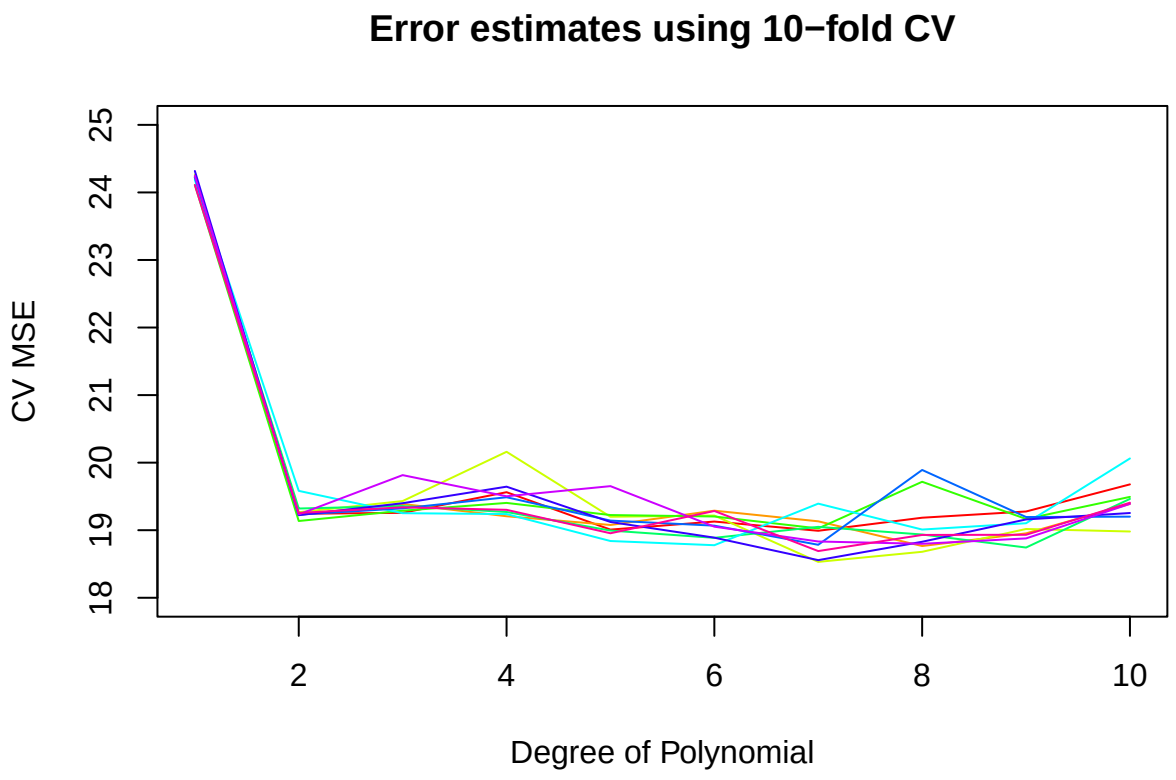
It took a lot more time to compute the LOOCV for this than any other procedure because it leaves out exactly one data point for testing and repeats it for every data point for all the degree polynomials. The degree 7 polynomial has the lowest error. Hence it looks like degree 7 polynomial is the best fit for our data.

c. Perform 10-fold cross-validation, and produce a plot like the one in the right-hand panel of Figure 5.4 of the textbook. Discuss your findings. What degree polynomial is best, and why?

```
error.k <- matrix(0,10,10)

for(j in 1:10){
  for (i in 1:10) {
    glm.fit.2 <- glm(mpg ~ poly(horsepower, i), data=Auto)
    error.k[j,i] <- cv.glm(Auto, glm.fit.2, K=10)$delta[1]
}
}
plot(0,0, xlim=c(1,10), ylim = c(18,25), xlab='Degree of Polynomial',
     ylab='CV MSE', type='b', main='Error estimates using 10-fold CV')

color<- rainbow(10)
for(i in 1:10){
  lines(error.k[i, ], col=color[i])
}
```




```
mean.error.k <- apply(error.k, 2, FUN='mean')
mean.error.k
```

```
##  [1] 24.21435 19.27207 19.38531 19.47793 19.12157 19.07954 18.89880 19.07431
##  [9] 19.04258 19.43188
```

```
which.min(mean.error.k)
```

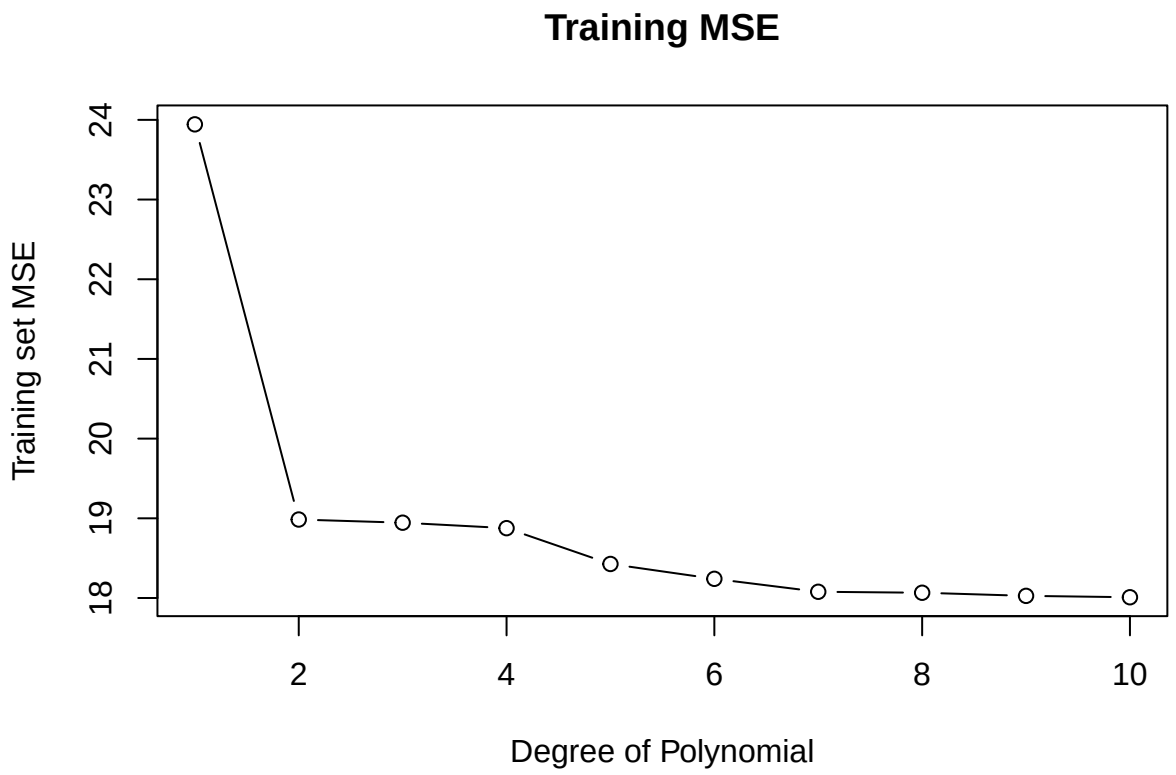
```
## [1] 7
```

The k-foLD Cross Validation approach was repeated 10 times for 10 different splits of train and validation sets. We see that the errors are highly variable and is dependent on how the data was split. I took average of errors for all splits for every degree polynomial. We see that 7 degree polynomial has the lowest error. Hence it looks like degree 7 polynomial is the best fit for our data.

d. Fit a least squares linear model to predict mpg using polynomials of degrees from 1 to 10, using all available observations. Make a plot showing "Degree of Polynomial" on the x-axis, and "Training Set Mean Squared Error" on the y-axis. Discuss your findings.

```
error.tr <- rep(0,10)
for(i in 1:10){
    lm.fit4d <- lm(mpg ~ poly(horsepower, i), data=Auto)
    error.tr[i] <- mean((Auto$mpg - predict(lm.fit4d, Auto))^2)
}

plot(error.tr, xlab='Degree of Polynomial', ylab='Training set MSE',
     type='b', main='Training MSE')
```



```
which.min(error.tr)
```

```
## [1] 10
```

```
error.tr[which.min(error.tr)]
```

```
## [1] 18.00953
```

In the previous questions we saw using Cross Validation (Validation Set, LOOCV, K-fold CV) that degree 7 polynomial best fit the data. We did this by train on a sample of data and then testing for our results on the validation data that was not used during the training process. However, here we don't do that. We train on the entire data and calculate error on the training data itself. This leads to overfitting in higher degree polynomial due to increased flexibility. We hence verify the fact that training MSE is not a true representation of expected prediction error. And so, we should not try to minimize training error but try to estimate test error and take modelling decisions based on estimated test error.

e. Fit a least squares linear model to predict mpg using a degree-10 polynomial, using all available observations. Using the summary command in R, examine the output. Comment on the output, and discuss how this relates to your findings in (a)–(d).

```
lm.fit4e <- lm(mpg ~ poly(horsepower, 10), data=Auto)
summary(lm.fit4e)
```

```
##
## Call:
## lm(formula = mpg ~ poly(horsepower, 10), data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.7081  -2.5904  -0.1922   2.2859  14.8338
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               23.4459     0.2174 107.840   <2e-16 ***
## poly(horsepower, 10)1   -120.1377     4.3046 -27.909   <2e-16 ***
## poly(horsepower, 10)2     44.0895     4.3046  10.242   <2e-16 ***
## poly(horsepower, 10)3     -3.9488     4.3046  -0.917   0.3595
## poly(horsepower, 10)4     -5.1878     4.3046  -1.205   0.2289
## poly(horsepower, 10)5     13.2722     4.3046   3.083   0.0022 **
## poly(horsepower, 10)6     -8.5462     4.3046  -1.985   0.0478 *
## poly(horsepower, 10)7      7.9806     4.3046   1.854   0.0645 .
## poly(horsepower, 10)8      2.1727     4.3046   0.505   0.6140
## poly(horsepower, 10)9     -3.9182     4.3046  -0.910   0.3633
## poly(horsepower, 10)10    -2.6146     4.3046  -0.607   0.5440
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.305 on 381 degrees of freedom
## Multiple R-squared:  0.7036, Adjusted R-squared:  0.6958
## F-statistic: 90.45 on 10 and 381 DF,  p-value: < 2.2e-16
```

For (a) - (c), we preformed Cross Validation using different approaches on the dataset and arrived at a similar result that degree 7 polynomial best fits the data. In (d), we see that if we train on all the datapoints, we see that the degree 10 polynomial performs the best. If we continue this, we can see that as flexibility increases, the training error will keep on reducing. Say, if we fit a 11 degree polynomial, it will have lower train MSE than 10-degree polynomial. As we increase flexibility, there would also be a connect the dots model that will fit all the data points and the train MSE would be zero. But in case like this, the test error would not be equal to the training error. It won't even be close to the training error.

When all data is taken as the training data, we are essentially trying to fit a model that best represents the data at the risk of overfitting. When we use CV, we are estimating the test error on untrained of unseen data points. When we do this, we arrive at a model which is better representative of our true model.

So, here we come to a conclusion that training MSE is not a good estimation of prediction error. Instead, estimate of test error should be done using untrained data.

5. Extra Credit! Let's consider doing least squares and ridge regression under a very simple setting, in which $p = 1$, and $\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} x_i = 0$. We consider regression without an intercept. (It's usually a bad idea to do regression without an intercept, but if our feature and response each have mean zero, then it is okay to do this!)

a. The least squares solution is the value of $\beta \in \mathbb{R}$ that minimizes

$$\sum_{i=1}^{n}(y_i - \beta x_i)^2$$

Write out an analytical (closed-form) expression for this least squares solution. Your answer should be a function of $x_1, ..., x_n$ and $y_1, ..., y_n$.Hint: Calculus!!

We want to find $\hat{\beta}_{LS}$ such that it minimizes the sum of squared residuals $\sum_{i=1}^{n}(y_i - \beta x_i)^2$

$$\hat{\beta}_{LS} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n}(y_i - \beta x_i)^2$$

Writing it in Matrix notation,

$$\hat{\beta}_{LS} = \underset{\beta}{\operatorname{argmin}} (Y - \beta X)^T (Y - \beta X)$$

Let,

$$S = Y^T Y - Y^T \beta X - \beta^T X^T Y + \beta^T X^T X \beta$$

We partially differentiate the above with respect to β and equate it to zero.

$$\frac{\partial S}{\partial \beta} = -2X^T Y + 2X^T X \beta$$

$$X^T Y = X^T X \beta$$

Solving for β,

$$\hat{\beta}_{LS} = (X^T X)^{-1} X^T Y$$

provided that the inverse of $X^T X$ exists, which means that the matrix X should have rank p. As X is an $n \times p$ matrix, this requires in particular that $n \geq p$ — that is, the number of parameters is smaller than or equal to the number of observations. In practice we will almost always require that p is considerably smaller than n.

Proof of minima:

If the matrix X has rank p, it follows that the Hessian matrix,

$\frac{\partial^2 S}{\partial \beta \partial \beta^T} = 2X^T X$

is a positive definite matrix.

This implies that $(X^T X)^{-1} X^T Y$ is indeed the minimum of $(Y - \beta X)^T (Y - \beta X)$

For our problem, $p = 1$,

$$X^T X = \sum_{i=1}^{n} x_i^2$$

and

$$X^T Y = \sum_{i=1}^{n} x_i y_i$$

So, $\hat{\beta}_{LS} = (X^T X)^{-1} X^T Y$

$$\implies \hat{\beta}_{LS} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}$$

$$\implies \hat{\beta}_{LS} = \frac{x_1 y_1 + ... + x_n y_n}{x_1^2 + ... + x_n^2}$$

b. For a given value of λ, the ridge regression solution minimizes

$$\sum_{i=1}^{n} (y_i - \beta x_i)^2 + \lambda \beta^2$$

Write out an analytical (closed-form) expression for the ridge regression solution, in terms of $x_1, ..., x_n$ and $y_1, ..., y_n$ and λ.

$$\hat{\beta}_{ridge} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \|\beta\|^2$$

Writing it in Matrix notation,

$$\hat{\beta}_{ridge} = \underset{\beta}{\operatorname{argmin}} \|X\beta - Y\|^2 + \lambda \|\beta\|^2$$

$$\frac{\partial \hat{\beta}_{ridge}}{\partial \beta} = 0$$

$$\implies 2X^T (X\beta - Y) + 2\lambda\beta = 0$$

Rearranging,

$$(X^T X + \lambda I)\beta - X^T Y = 0$$

$$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T Y$$

Proof of minima:

This solution is a global minimum if we know that $\|X\beta - Y\|^2 + \lambda \|\beta\|^2$ is strictly convex. This is true. Because, $f_{ridge}(\beta, \lambda)$ is equivalent to f_{LS} constrained to $\|\beta\|^2 \leq t$. From this perspective, $f_{ridge}(\beta, \lambda)$ is just

the Lagrangian function used to find the global minima of convex objective function f_{LS} constrained with convex function $\|\beta\|^2$

For our problem, $p = 1$,

$$X^T X = \sum_{i=1}^{n} x_i^2$$

and

$$X^T Y = \sum_{i=1}^{n} x_i y_i$$

and

$\lambda I = \lambda$

So,

$$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T Y$$

$$\implies \hat{\beta}_{ridge} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} (x_i^2) + \lambda}$$

$$\implies \hat{\beta}_{ridge} = \frac{x_1 y_1 + ... + x_n y_n}{x_1^2 + ... + x_n^2 + \lambda}$$

c. Suppose that the true data-generating model is

$$Y = 3X + \epsilon$$

where ϵ has mean zero, and X is fixed (non-random). What is the expectation of the least squares estimator from (a)? Is it biased or unbiased?

From (a),

$\hat{\beta} = (X^T X)^{-1} X^T Y$

$\hat{\beta} = (X^T X)^{-1} X^T (X\beta + \epsilon)$

$\hat{\beta} = \beta + (X^T X)^{-1} \epsilon$

$\mathbb{E}[\hat{\beta}|X] = \beta + \mathbb{E}[(X^T X)^{-1} \epsilon | X]$

By exogenity of independent variables,

$\mathbb{E}[\epsilon | x_1, ..., x_n] = 0$

So,

$\mathbb{E}[(X^T X)^{-1} \epsilon | X] = 0$

$\mathbb{E}[\hat{\beta}|X] = \beta$

So,

$\mathbb{E}[\hat{\beta}] = \mathbb{E}_X[\mathbb{E}[\hat{\beta}|X] = \mathbb{E}_X[\beta] = \beta$

Therefore, Least Squares estimator in unbiased.

For our problem,

$\mathbb{E}[\hat{\beta}] = 3$

d. Suppose again that the true data-generating model is $Y = 3X + \epsilon$, where ϵ has mean zero, and X is fixed (non-random). What is the expectation of the ridge regression estimator from (b)? Is it biased or unbiased? Explain how the bias changes as a function of λ.

$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T Y$

$\mathbb{E}[\hat{\beta}_{ridge}] = \mathbb{E}[(X^T X + \lambda I)^{-1} X^T Y]$

$\mathbb{E}[\hat{\beta}_{ridge}] = (X^T X + \lambda I)^{-1} X^T \mathbb{E}[Y]$

$\mathbb{E}[\hat{\beta}_{ridge}] = (X^T X + \lambda I)^{-1} X^T (X\beta)$

$\mathbb{E}[\hat{\beta}_{ridge}] = (X^T X + \lambda I)^{-1} X^T X\beta$

$\mathbb{E}[\hat{\beta}_{ridge}] = (X^T X + \lambda I)^{-1} (X^T X + \lambda I - \lambda I)\beta$

$\mathbb{E}[\hat{\beta}_{ridge}] = (X^T X + \lambda I)^{-1} (X^T X + \lambda I)\beta - (X^T X + \lambda I)^{-1} (\lambda I)\beta$

$\mathbb{E}[\hat{\beta}_{ridge}] = \beta - (X^T X + \lambda I)^{-1} (\lambda I)\beta$

$\implies \mathbb{E}[\hat{\beta}_{ridge}] \neq \beta;\ \forall \lambda > 0$

Hence, Ridge estimator is biased.

For our problem,

$$\mathbb{E}[\hat{\beta}_{ridge}] = 3 - \frac{3\lambda}{\sum_{i=1}^{n}(x_i^2) + \lambda}$$

$\implies \mathbb{E}[\hat{\beta}_{ridge}] = 3 - \frac{3\lambda}{x_1^2 + \ldots + x_n^2 + \lambda}$

The bias of the estimator is $\mathbb{E}[\hat{\beta}_{ridge}] - \beta$

$Bias[\hat{\beta}_{ridge}] = (\beta - (X^T X + \lambda I)^{-1} (\lambda I)\beta - \beta)$

$Bias[\hat{\beta}_{ridge}] = -(X^T X + \lambda I)^{-1} (\lambda I)\beta$

For our problem,

$$Bias[\hat{\beta}_{ridge}] = \frac{-3\lambda}{\sum_{i=1}^{n}(x_i^2) + \lambda}$$

$\implies Bias[\hat{\beta}_{ridge}] = \frac{-3\lambda}{x_1^2 + \ldots + x_n^2 + \lambda}$

$\implies Bias^2[\hat{\beta}_{ridge}] = \left(\frac{-3\lambda}{x_1^2 + \ldots + x_n^2 + \lambda}\right)^2$

$\implies Bias^2[\hat{\beta}_{ridge}] = \frac{9\lambda^2}{(x_1^2 + \ldots + x_n^2 + \lambda)^2}$

This is how the bias changes as a function of λ

e. Suppose that the true data-generating model is $Y = 3X + \epsilon$, where ϵ has mean zero and variance σ^2, and X is fixed (non-random), and also $Cov(\epsilon_i, \epsilon_j) = 0$ for all $i \neq j$. What is the variance of the least squares estimator from (a)?

$\hat{\beta}_{LS} = (X^T X)^{-1} X^T Y$

Let our true data generating model be $Y = \beta X + \epsilon$

$\hat{\beta}_{LS} = (X^T X)^{-1} X^T (\beta X + \epsilon)$

$\hat{\beta}_{LS} = (X^T X)^{-1}\beta(X^T X) + (X^T X)^{-1}X^T\epsilon$

$\hat{\beta}_{LS} = \beta + (X^T X)^{-1}X^T\epsilon$

$var(\hat{\beta}_{LS}) = (X^T X)^{-1}X^T var(\epsilon)X(X^T X)^{-1}$

$var(\hat{\beta}_{LS}) = (X^T X)^{-1}X^T(\sigma^2 I)X(X^T X)^{-1}$

$var(\hat{\beta}_{LS}) = (X^T X)^{-1}\sigma^2$

For our problem,

$$X^T X = \sum_{i=1}^{n} x_i^2$$

So,

$$var(\hat{\beta}_{LS}) = \frac{\sigma^2}{\sum_{i=1}^{n} x_i^2}$$

$var(\hat{\beta}_{LS}) = \frac{\sigma^2}{x_1^2+\ldots+x_n^2}$

f. uppose that the true data-generating model is $Y = 3X + \epsilon$, where ϵ has mean zero and variance σ^2, and X is fixed (non-random), and also $Cov(\epsilon_i, \epsilon_j) = 0$ for all $i \neq j$. What is the variance of the ridge estimator from (b)? How does the variance change as a function of λ?

$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1}X^T Y$

This could be written as,

$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1}(X^T X)(X^T X)^{-1}X^T Y$

But we know that, $\hat{\beta}_{LS} = (X^T X)^{-1}X^T Y$. So, rewriting the above equation,

$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1}(X^T X)\hat{\beta}_{LS}$

$var(\hat{\beta}_{ridge}) = (X^T X + \lambda I)^{-1}(X^T X)var(\hat{\beta}_{LS})X^T X(X^T + \lambda I)^{-1}$

From (e.), we know that $var(\hat{\beta}_{LS}) = (X^T X)^{-1}\sigma^2$

So,

$var(\hat{\beta}_{ridge}) = (X^T X + \lambda I)^{-1}(X^T X)(X^T X)^{-1}\sigma^2 X^T X(X^T + \lambda I)^{-1}$

$var(\hat{\beta}_{ridge}) = (X^T X + \lambda I)^{-1}X^T X(X^T X + \lambda I)^{-1}\sigma^2$

For our problem,

$$X^T X = \sum_{i=1}^{n} x_i^2$$

and

$\lambda I = \lambda$

$$var(\hat{\beta}_{ridge}) = \frac{\sigma^2\left(\sum_{i=1}^{n} x_i^2\right)}{\left(\sum_{i=1}^{n}(x_i^2) + \lambda\right)^2}$$

$var(\hat{\beta}_{ridge}) = \frac{\sigma^2(x_1^2+\ldots+x_n^2)}{(x_1^2+\ldots+x_n^2+\lambda)^2}$

g. In light of your answers to parts (d) and (f), argue that λ in ridge regression allows us to control model complexity by trading off bias for variance.

Linear regression finds the coefficient values that minimize RSS. But this may not be the best model, and will give a coefficient for each predictor provided. This includes terms with little predictive power. This results in a high-variance, low bias model. We therefore have the potential to improve our model by trading some of that variance with bias to reduce our overall error. This trade comes in the form of regularization, in which we modify our cost function to restrict the values of our coefficients. This allows us to trade our excessive variance for some bias, potentially reducing our overall error.

We increase our bias a little bit as a function of λ as

$Bias^2[\hat{\beta}_{ridge}] = \frac{9\lambda^2}{(x_1^2+...+x_n^2+\lambda)^2}$

as compared to $Bias^2[\hat{\beta}_{LS}] = 0$

and decrease the variance as

$var(\hat{\beta}_{ridge}) = \frac{\sigma^2(x_1^2+...+x_n^2)}{(x_1^2+...+x_n^2+\lambda)^2}$

as compared to $var(\hat{\beta}_{LS}) = \frac{\sigma^2}{x_1^2+...+x_n^2}$

So, overall, we are trying to reduce the reducible error by trading some of the variance with bias. By intuition, this should happen.

In our favorite graph, as we go from more flexibility to less flexibility, we reduce variance and increase bias in the model.