DATA 558: Statistical Machine Learning
Homework 3

Abhishek Saini

May 2022

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:

   ```
   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

   $$Y = 2X_1 + 3X_2 + \epsilon, \qquad \epsilon \sim N(0,1).$$

   Since there is no dependence on any column of X and Y is drawn from $N(0,1)$, the expression for the model corresponding to this data generation procedure is given by:

   $$Y = \epsilon, \qquad \epsilon \sim N(0,1).$$

   (b) What is the value of the irreducible error?
   The irreducible error term has mean, $\mathbb{E}[\epsilon] = 0$, and variance, $\text{Var}(\epsilon) = 1$

   (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?
   Since $\mathbb{E}[\hat{f}(x)] = \mathbb{E}[Y] = 0$, the bias of this procedure is 0.

   ii. What is the variance of this procedure?
   Since for different instances of training data, the function $\hat{f}(x)$, that we learn, remains the same, the variance of this procedure is 0.

   iii. What is the expected prediction error of this procedure?
   Since the squared bias and variance of this procedure are 0, the expectation of the squared prediction error will be equal to the variance of the irreducible error. Mathematically, $\mathbb{E}[(Y - \hat{f}(x))^2] = \text{Var}(\epsilon) = 1$.

iv. Use the validation set approach to estimate the test error of this procedure. What answer do you get?
With validation set approach, we estimate the test error by the MSE of the fitted model on the validation set, which turns out to be 1.128815.

```
set.seed(42)
n = 100
p = 10000
y = rnorm(n)
y = as.data.frame(y)
x = matrix(rnorm(p*n), ncol = p)
x = as.data.frame(x)
data = cbind(y, x)
type_row = rep("train", n)

val_size = 0.5*n
val = sample(x = 1:n, size = val_size, replace = FALSE)
type_row[val] = "val"
data["type"] = type_row

res = data$y[val] - 0
mse = sum(res^2)/val_size; mse
[1] 1.128815
```

v. Comment on your answers to (iii) and (iv). Do your answers agree with each other? Explain. The answers are close to each other. They will very likely not be exactly equal to each other because of the finite sample size.

(d) Now use the validation set approach to estimate the test error of a least squares linear model using $X_1, \ldots, X_{10,000}$ to predict Y. What is the estimated test error?

```
train_data = data[data$type=="train",]
train_data = subset(train_data, select = -c(type))
val_data = data[data$type=="val",]
val_data = subset(val_data, select = -c(type))

lm.fit = lm(y ~ ., data = train_data)

y_pred = predict(lm.fit, val_data)
res = val_data$y - y_pred
mse = sum(res^2)/val_size; mse
```

```
[1] 66.83542
```

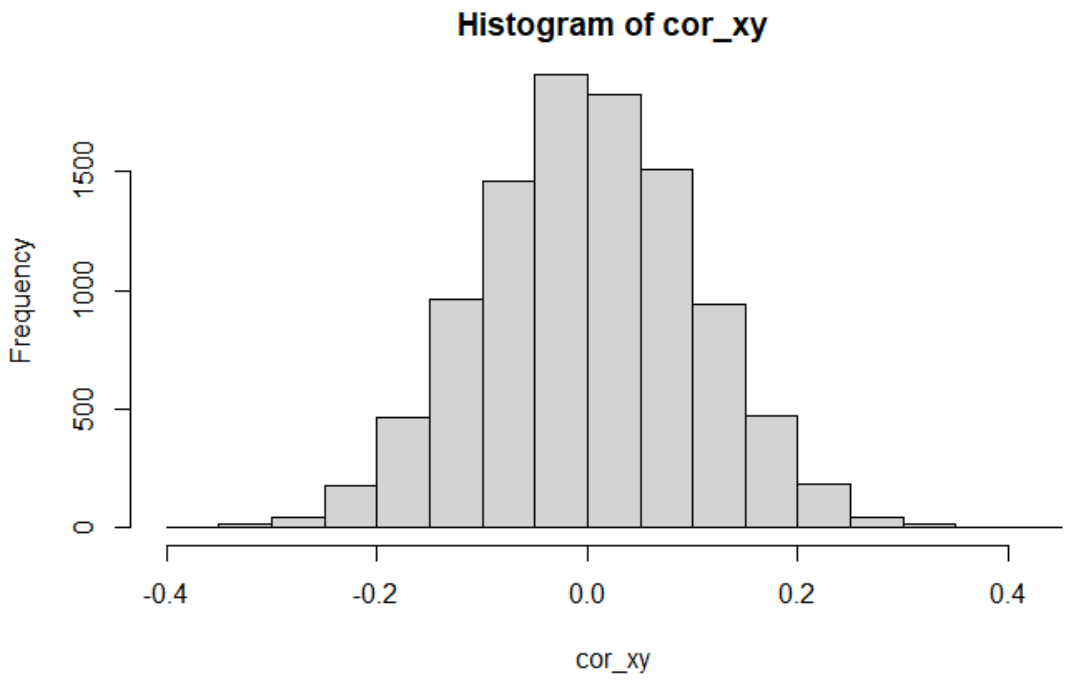
The estimated test error based on the least squares model fit on the simulated data above using the validation set approach is 66.83542.

(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.
The model which always predicts 0 has a smaller estimated test error.
Both models will have zero bias because since, on average, every parameter will be 0.
Since the number of features is much larger than the number of training data, any model that we fit using least squares will fit the noise instead of the data. The model fit via least squares has a higher variance because a different model is fit for different realization of the training data.

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.

(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?
The 10 largest correlations are -0.3397968, 0.3774119, -0.3325739, -0.3316668, 0.3460830, -0.3329436, -0.3308413, -0.3691199, 0.4123726, -0.3457064

```
# make histogram of correlations
cor_xy = cor(x, y)
q = 10

hist(cor_xy)

# get q most correlated features
cor_xy_abs = abs(cor_xy)
corr_filter = sort(cor_xy_abs)[10000-q+1]
filter_indices = which(cor_xy_abs >= corr_filter)
cor_xy[filter_indices]
[1] -0.3397968  0.3774119 -0.3325739 -0.3316668  0.3460830 -0.3329436 -0.3308413 -0.36
[10] -0.3457064
```



Histogram of Correlations

(b) Now try out "Option 1" with $q = 10$. What is the estimated test error?
The estimated error with "Option 1" is 0.6012443

```
# option 1) train and test on these features
data2 = cbind(y, x_o1)
data2["type"] = type_row
train_data2 = data2[data2$type=="train",]
train_data2 = subset(train_data2, select = -c(type))
val_data2 = data2[data2$type=="val",]
val_data2 = subset(val_data2, select = -c(type))

lm2.fit = lm(y ~ ., data = train_data2)

y_pred = predict(lm2.fit, val_data2)
res = val_data2$y - y_pred
mse = sum(res^2)/val_size; mse
[1] 0.6012443
```

(c) Now try out "Option 2" with $q = 10$. What is the estimated test error?
The estimated error with "Option 1" is 1.875918

```
# option 2) use train val set from q1 first, then get q most correlated features in t
```

```
cor_xy = cor(train_data[, -1], train_data$y)
cor_xy_abs = abs(cor_xy)
corr_filter = sort(cor_xy_abs)[10000-q+1]
filter_indices = which(cor_xy_abs >= corr_filter)
cor_xy[filter_indices]

x_o2 = x[, filter_indices]

data3 = cbind(y, x_o2)
data3["type"] = type_row
train_data3 = data3[data3$type=="train",]
train_data3 = subset(train_data3, select = -c(type))
val_data3 = data3[data3$type=="val",]
val_data3 = subset(val_data3, select = -c(type))

lm3.fit = lm(y ~ ., data = train_data3)

y_pred = predict(lm3.fit, val_data3)
res = val_data3$y - y_pred
mse = sum(res^2)/val_size; mse
[1] 1.875918
```

(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.
"Option 1" will give a spurious estimate of the test error. This happens because the act of limiting ourselves to the top 10 features with highest correlation for the overall data (test and train) causes our test data to not be independent of the train data. This leads to the observations in test set being correlated with the response variable, and thus the model trained on the training data will perform well on the test set. This may mislead us into believing the model performs better than it would actually perform on previously unseen data. "Option 2" avoids this pitfall by first splitting the training and test set and then choosing the top 10 features with highest correlation **only** on the training set.

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.

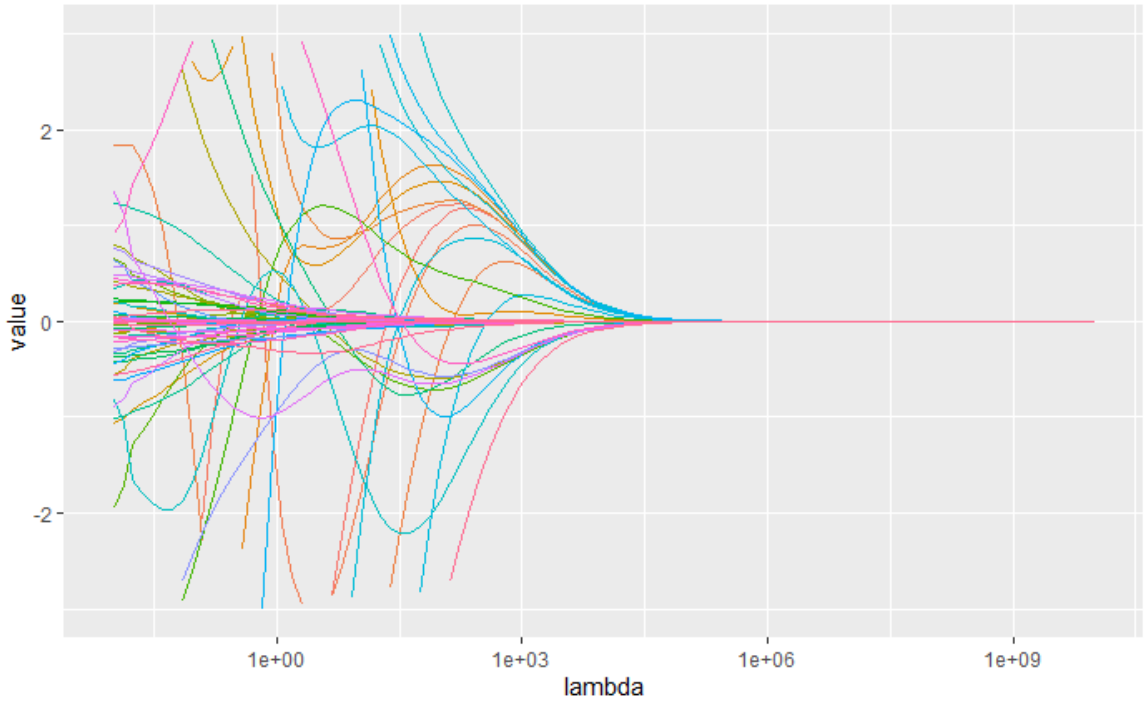
(a) Describe the data. Where did you get it from? What is the meaning of the response, and what are the meanings of the predictors?
The dataset that I have used is the Superconductivity Data Data Set that I have taken from

the UCI Machine Learning Repository. Here is a link to the dataset. The first 81 columns contain 81 features extracted from 21263 superconductors. The 82nd column contains the critical temperature for that superconductor. The 81 features can be used to predict the critical temperature. All the 81 features are quantitative predictors that denote the chemical and physical properties of the superconductor like atomic mass, entropy, electron affinity, fusion heat, thermal conductivity, atomic radius, etc.

(b) Fit a least squares linear model to the data, and provide an estimate of the test error. (Explain how you got this estimate.)
The code for this part is given below. To estimate the test error, I have used the K-fold cross validation with number of folds $K = 10$.

```
set.seed(42)
library(glmnet)
library(boot)
library(ggplot2)

data1 <- read.csv("train.csv", header=TRUE, stringsAsFactors=FALSE)

# 3 b)

# There are no missing values in the data frame
sum(is.na(data1))

glm.fit <- glm(critical_temp ~ ., data = data1)
cv.glm(data1, glm.fit, K = 10)$delta[1]
[1] 310.4991
```

(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.



Ridge regression coefficients as a function of λ

```
# 3 c)
x = model.matrix(critical_temp ~ ., data1)[, -1]
y = data1$critical_temp

grid <- 10^seq(10, -2, length = 100)
ridge.mod <- glmnet(x, y, alpha = 0, lambda = grid)

coeffs = coef(ridge.mod)
rows = rownames(coeffs)

cols = c("lambda", "coeff", "value")
plot_df = data.frame(matrix(ncol = length(cols), nrow = 0))
colnames(plot_df) = cols

for(row in rows[2:length(rows)]) {
  temp_df = data.frame(lambda = grid, coeff = row, value = coeffs[row,])
  plot_df = rbind(plot_df, temp_df)
}

ggplot(data = plot_df, aes(x=lambda, y=value)) + geom_line(aes(color = coeff), show.le
```

(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.)

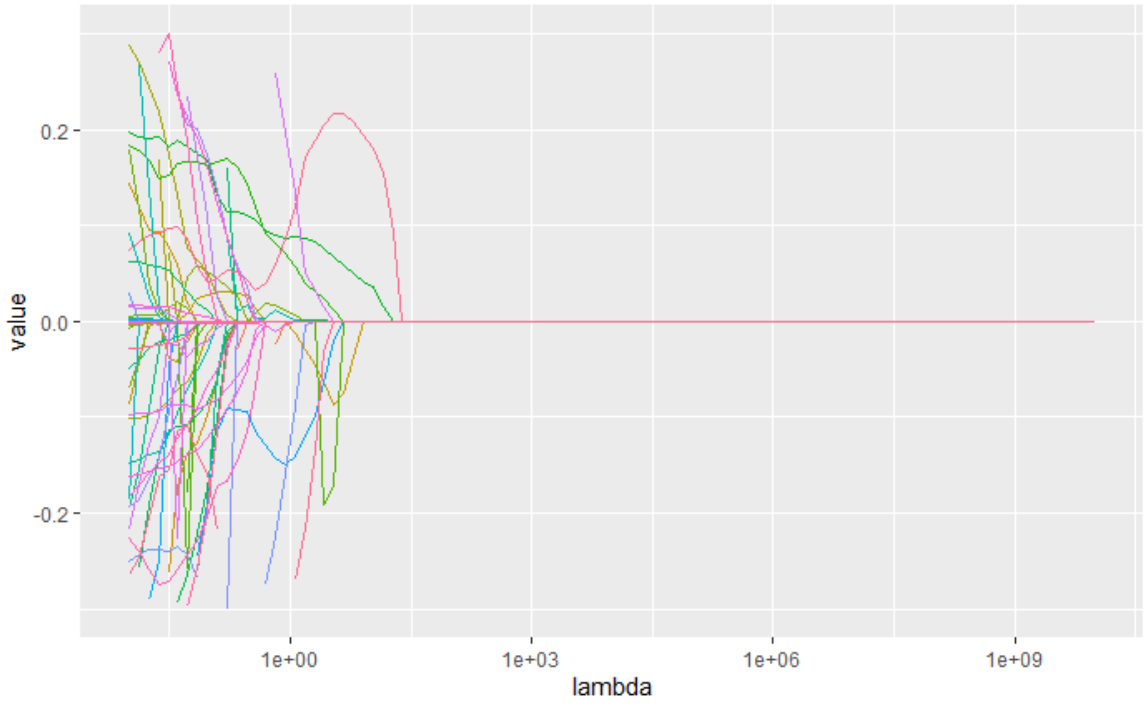
The value of λ with the smallest estimated test error is 2.457059. This is calculated by first splitting the data into train and test set and then choosing the best λ by performing K-fold cross validation using 10 folds created from the training data. Test set is not used in this step.

The estimate of the test error is 348.2352. This is calculated by first creating a model that is trained on the entire training set and evaluated on the test set for the best λ we got previously.

Code for this is given below.

```
# 3 d)
train <- sample (1: nrow(x), nrow(x) / 2)
test <- (-train)
y.test <- y[test]

ridge.mod <- glmnet(x[train , ], y[train], alpha = 0, lambda = grid , thresh = 1e-12)

cv.out <- cv.glmnet(x[train , ], y[train], alpha = 0)
bestlam = cv.out$lambda.min; bestlam
[1] 2.457059

ridge.pred <- predict(ridge.mod, s = bestlam , newx = x[test , ])
mean (( ridge.pred - y.test)^2)
[1] 348.2352
```

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



Lasso regression coefficients as a function of λ

```
lasso.mod <- glmnet(x, y, alpha = 1, lambda = grid)

coeffs = coef(lasso.mod)
rows = rownames(coeffs)

cols = c("lambda", "coeff", "value")
plot_df = data.frame(matrix(ncol = length(cols), nrow = 0))
colnames(plot_df) = cols

for(row in rows[2:length(rows)]) {
  temp_df = data.frame(lambda = grid, coeff = row, value = coeffs[row,])
  plot_df = rbind(plot_df, temp_df)
}

ggplot(data = plot_df, aes(x=lambda, y=value)) + geom_line(aes(color = coeff), show.le
```

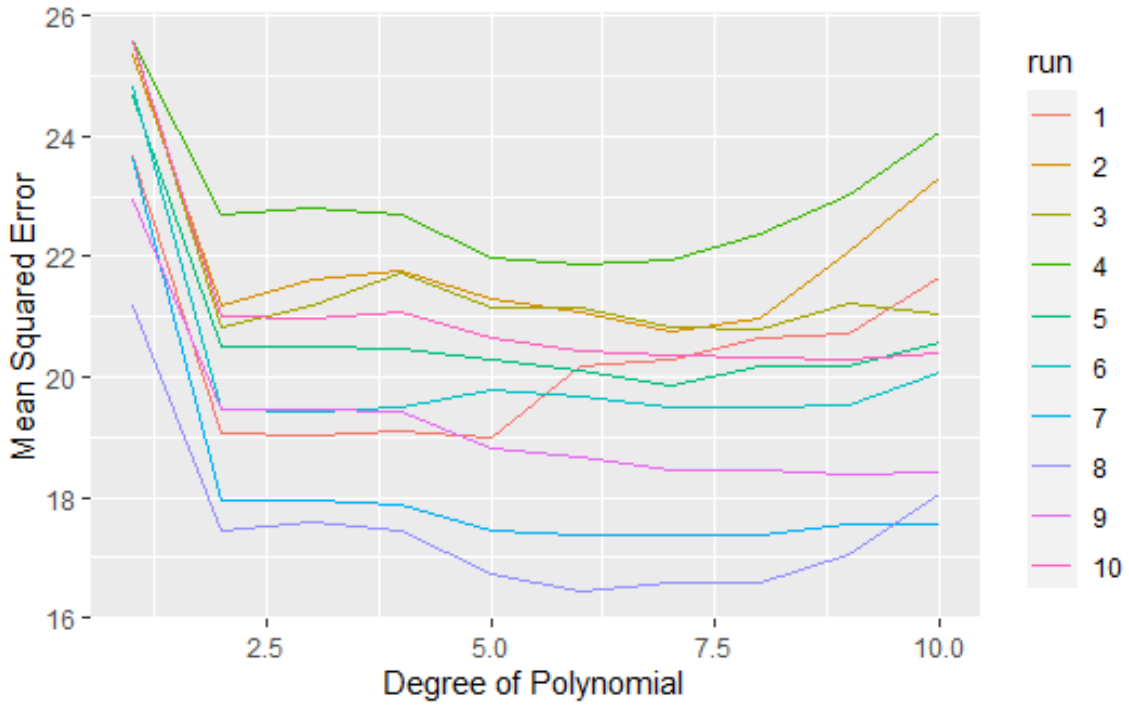
(f) Repeat (d), but for a lasso model. Which features are included in this lasso model?
The value of λ with the smallest estimated test error is 0.002457059. This is calculated by first splitting the data into train and test set and then choosing the best λ by performing K-fold cross validation using 10 folds created from the training data. Test set is not used in this step.

The estimate of the test error is 307.7581. This is calculated by first creating a model that is trained on the entire training set and evaluated on the test set for the best λ we got previously.
All features except gmean_fie, wtd_gmean_fie, entropy_fie, entropy_atomic_radius, wtd_gmean_FusionHeat, mean_ThermalConductivity, mean_Valence, wtd_mean_Valence, std_Valence are included in the Lasso model.
Code for this is given below.

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

cv.out <- cv.glmnet(x[train , ], y[train], alpha = 1)
bestlam = cv.out$lambda.min; bestlam
[1] 0.002457059

lasso.pred <- predict(lasso.mod
, s = bestlam , newx = x[test , ])
mean (( lasso.pred - y.test)^2)
[1] 307.7581

lasso.coef <- predict(lasso.mod, type = "coefficients", s = bestlam)
lasso.coef
```

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?



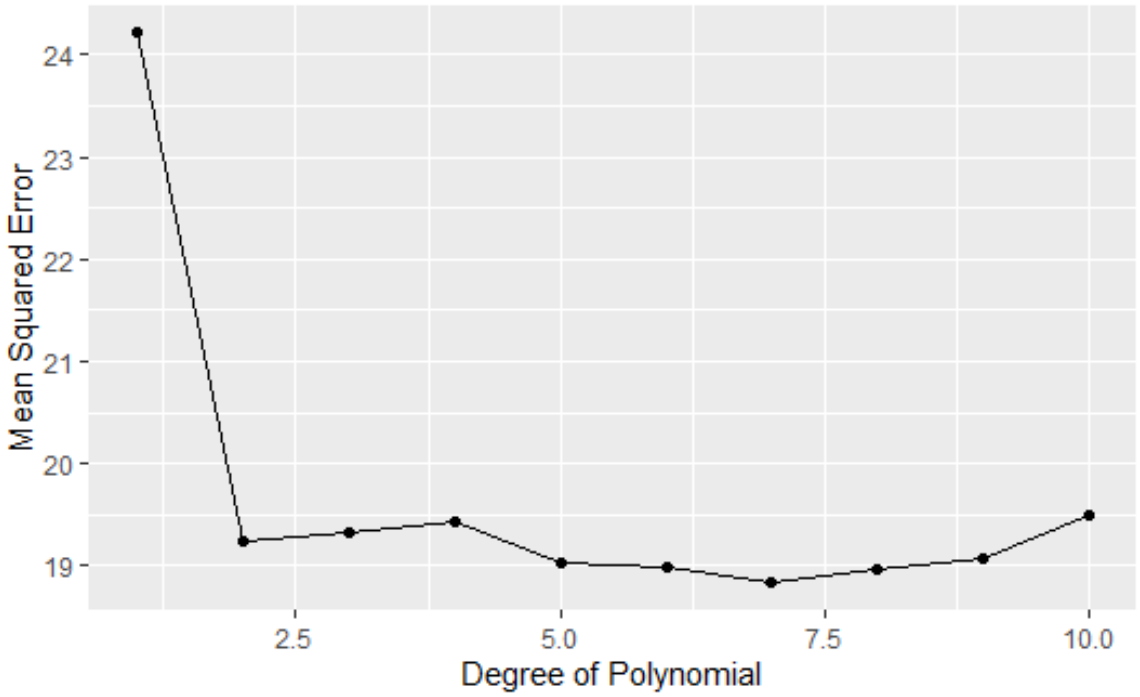
Estimate of test error (validation set approach) vs degree of polynomial

For different splits between training and test data, we get different test MSE estimates. However, there is a sharp drop in MSE on increasing the degree of the polynomial from 1 to 2. Increasing the degree of polynomial beyond 2 doesn't lead to any observable decrease in test MSE. So degree 2 polynomial is the best choice since it is the simplest model that performs equally well as any other model with higher degree of polynomial.

```
library(ISLR2)
set.seed(99)

attach(Auto)

# 4 a)
max_run = 10
max_degree = 10

test_mse_array = matrix(0, max_run, max_degree)

for (run in 1:max_run){
  # get train indices
  train <- sample (392 , 196)
```

```
  # run the validation method for this train test split
  for (degree in 1:max_degree){
    lm.fit = lm(mpg ~ poly(horsepower, degree), data = Auto , subset = train)
    test_mse_array[run, degree] = mean(( mpg - predict(lm.fit , Auto))[-train ]^2)
  }
}

get_test_mse <- function(run, degree){
  return(test_mse_array[run, degree])
}

run_arr = rep(1:max_degree, each=max_run)
degree_arr = rep(1:max_run, max_degree)
plot_df = data.frame(run = run_arr, degree = degree_arr, test_mse=mapply(get_test_mse
plot_df$run = factor(plot_df$run)

ggplot(data = plot_df, aes(x=degree, y=test_mse)) + geom_line(aes(colour=run)) +
  labs(x="Degree of Polynomial", y="Mean Squared Error")
```

(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?

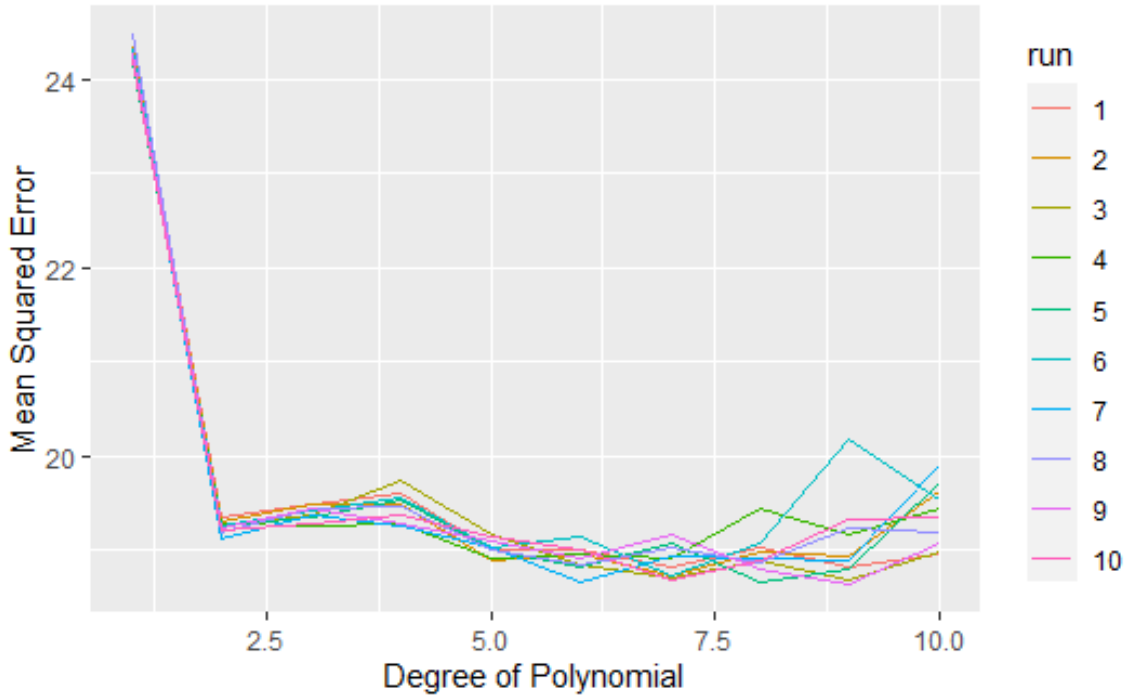


Estimate of test error (LOOCV) vs degree of polynomial

There is a sharp drop in MSE on increasing the degree of the polynomial from 1 to 2. Increasing the degree of polynomial beyond 2 doesn't lead to any observable decrease in test MSE. So degree 2 polynomial is the best choice since it is the simplest model that performs equally well as any other model with higher degree of polynomial.

```
# 4 b)
library(boot)
cv.error <- rep(0, max_degree)
for (degree in 1:max_degree) {
  glm.fit <- glm(mpg ~ poly(horsepower , degree), data = Auto)
  cv.error[degree] <- cv.glm(Auto , glm.fit)$delta[1]
}
cv.error

plot_df2 = data.frame(degree = 1:max_degree, test_mse = cv.error)

ggplot(data = plot_df2, aes(x=degree, y=test_mse)) + geom_line() + geom_point() +
  labs(x="Degree of Polynomial", y="Mean Squared Error")
```

(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?



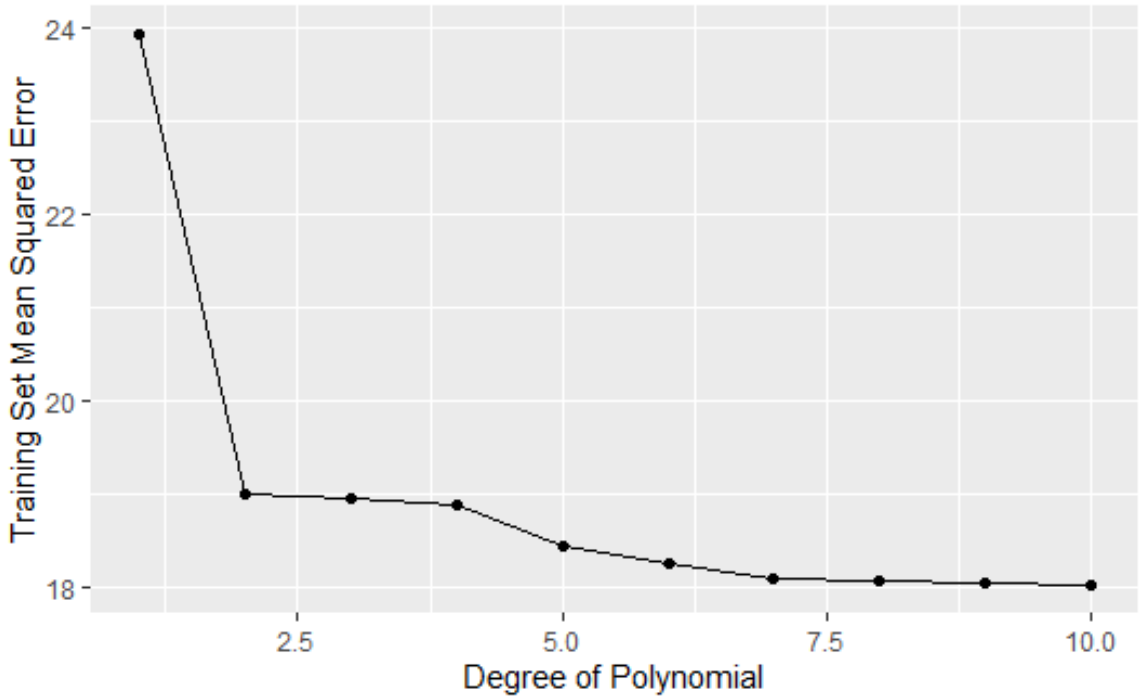
Estimate of test error (K-fold approach) vs degree of polynomial

For different splits between training and test data, we get different test MSE estimates. However, there is a sharp drop in MSE on increasing the degree of the polynomial from 1 to 2. Increasing the degree of polynomial beyond 2 doesn't lead to any observable decrease in test MSE. So degree 2 polynomial is the best choice since it is the simplest model that performs equally well as any other model with higher degree of polynomial.

```
# 4 c)
test_mse_array = matrix(0, max_run, max_degree)
for (run in 1:max_run){
  for (degree in 1:max_degree) {
    glm.fit <- glm(mpg ~ poly(horsepower , degree), data = Auto)
    test_mse_array[run, degree] <- cv.glm(Auto , glm.fit , K = 10)$delta [1]
  }
}
test_mse_array

plot_df3 = data.frame(run = run_arr, degree = degree_arr, test_mse=mapply(get_test_mse
plot_df3$run = factor(plot_df3$run)
```

```
ggplot(data = plot_df3, aes(x=degree, y=test_mse)) + geom_line(aes(colour=run)) +
  labs(x="Degree of Polynomial", y="Mean Squared Error")
```

(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.



Training error vs degree of polynomial

There is a sharp drop in training set MSE on increasing the degree of the polynomial from 1 to 2. The training set MSE monotonically decreases as the degree of polynomial increases. This is consistent with what we would expect as the training set performance would get better with each additional feature.

```
# 4 d)
train.error <- rep(0, max_degree)
for (degree in 1:max_degree) {
  glm.fit <- glm(mpg ~ poly(horsepower , degree), data = Auto)
  train.error[degree] <- mean((glm.fit$residuals)^2)
}
plot_df4 = data.frame(degree = 1:max_degree, train_mse = train.error)

ggplot(data = plot_df4, aes(x=degree, y=train_mse)) + geom_line() + geom_point() +
```

```
  labs(x="Degree of Polynomial", y="Training Set Mean Squared 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).

We can see that the p-value associated with the coefficient of the intercept, first degree and second degree of horsepower are extremely small. This is consistent with our observation in (a)-(d) where adding the quadratic term significantly reduces the training and test MSE. We also notice that the coefficient associated with the fifth degree of horsepower is small which could explain the drop in train, test MSE when degree increases from 4 to 5.

```
# 4 e)
glm.fit <- glm(mpg ~ poly(horsepower , degree), data = Auto)
summary(glm.fit)
Call:
glm(formula = mpg ~ poly(horsepower, degree), data = Auto)

Deviance 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, degree)1    -120.1377     4.3046 -27.909   <2e-16 ***
poly(horsepower, degree)2      44.0895     4.3046  10.242   <2e-16 ***
poly(horsepower, degree)3      -3.9488     4.3046  -0.917   0.3595
poly(horsepower, degree)4      -5.1878     4.3046  -1.205   0.2289
poly(horsepower, degree)5      13.2722     4.3046   3.083   0.0022 **
poly(horsepower, degree)6      -8.5462     4.3046  -1.985   0.0478 *
poly(horsepower, degree)7       7.9806     4.3046   1.854   0.0645 .
poly(horsepower, degree)8       2.1727     4.3046   0.505   0.6140
poly(horsepower, degree)9      -3.9182     4.3046  -0.910   0.3633
poly(horsepower, degree)10     -2.6146     4.3046  -0.607   0.5440
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 18.52949)

    Null deviance: 23819.0  on 391  degrees of freedom
```

```
Residual deviance: 7059.7 on 381 degrees of freedom
AIC: 2269.7

Number of Fisher Scoring iterations: 2
```

5. Let's consider doing least P squares and P n ridge regression under n 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 $\beta \in R$ 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 $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$.

   $$\begin{aligned} L(\beta) &= \sum_{i=1}^{n}(y_i - \beta x_i)^2 \\ &= \sum_{i=1}^{n}(y_i^2 + \beta^2 x_i^2 - 2\beta y_i x_i) \end{aligned}$$

   Differentiating the cost function with respect to β and setting it to zero,

   $$\begin{aligned} \frac{\partial L(\beta)}{\partial \beta} &= \frac{\partial \sum_{i=1}^{n}(y_i^2 + \beta^2 x_i^2 - 2\beta y_i x_i)}{\partial \beta} \\ 0 &= \sum_{i=1}^{n}(2\hat{\beta} x_i^2 - 2 y_i x_i) \\ \hat{\beta} &= \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2} \end{aligned}$$

   (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 $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ and λ.

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

Differentiating the cost function with respect to β and setting it to zero,

$$0 = \sum_{i=1}^{n} (2\hat{\beta}_R x_i^2 - 2y_i x_i) + 2\lambda\hat{\beta}_R$$
$$\hat{\beta}_R = \frac{\sum_{i=1}^{n} x_i y_i}{\lambda + \sum_{i=1}^{n} x_i^2}$$

(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?

$$\begin{aligned}
\mathbb{E}[\hat{\beta}] &= \mathbb{E}\left[\frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}\right] \\
&= \mathbb{E}\left[\frac{\sum_{i=1}^{n} x_i(3x_i + \epsilon_i)}{\sum_{i=1}^{n} x_i^2}\right] \\
&= \mathbb{E}\left[\frac{\sum_{i=1}^{n} (3x_i^2 + x_i\epsilon_i)}{\sum_{i=1}^{n} x_i^2}\right] \\
&= \frac{\mathbb{E}\left[\sum_{i=1}^{n} (3x_i^2 + x_i\epsilon_i)\right]}{\sum_{i=1}^{n} x_i^2} \\
&= \frac{\sum_{i=1}^{n} 3x_i^2 + \sum_{i=1}^{n} x_i\,\mathbb{E}\left[\epsilon_i\right]}{\sum_{i=1}^{n} x_i^2} \\
&= \frac{3\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n} x_i^2} = 3
\end{aligned}$$

Hence, the least squares estimator from (a) gives an unbiased estimate.

(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.

$$\begin{aligned}
\mathbb{E}[\hat{\beta}_R] &= \mathbb{E}\left[\frac{\sum_{i=1}^{n} x_i y_i}{\lambda + \sum_{i=1}^{n} x_i^2}\right] \\
&= \mathbb{E}\left[\frac{\sum_{i=1}^{n} (3x_i^2 + x_i\epsilon_i)}{\lambda + \sum_{i=1}^{n} x_i^2}\right] \\
&= \frac{3\sum_{i=1}^{n} x_i^2}{\lambda + \sum_{i=1}^{n} x_i^2} \\
&= \frac{3}{1 + \frac{\lambda}{\sum_{i=1}^{n} x_i^2}}
\end{aligned}$$

Hence, for any $\lambda > 0$, the ridge regression estimator is biased. For $\lambda = 0$, the estimator has no bias whereas as λ increases, the bias increases. As λ becomes larger and larger, $\mathbb{E}[\hat{\beta}_R]$ gets closer and closer to 0, so bias gets closer and closer to 3 to which it eventually converges.

(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 $\text{Cov}(\epsilon_i, \epsilon_{i'}) = 0$ for all $i \neq i'$. What is the variance of the least squares estimator from (a)?

$$\begin{aligned}
\text{Var}(\hat{\beta}) &= \text{Var}\left[\frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2}\right] \\
&= \frac{\text{Var}\left[\sum_{i=1}^n x_i y_i\right]}{(\sum_{i=1}^n x_i^2)^2} \\
&= \frac{\text{Var}\left[\sum_{i=1}^n (3x_i^2 + x_i\epsilon_i)\right]}{(\sum_{i=1}^n x_i^2)^2} \\
&= \frac{\text{Var}\left[\sum_{i=1}^n x_i\epsilon_i\right]}{(\sum_{i=1}^n x_i^2)^2} \\
&= \frac{\sum_{i=1}^n \text{Var}\left[x_i\epsilon_i\right]}{(\sum_{i=1}^n x_i^2)^2} \\
&= \frac{\sum_{i=1}^n x_i^2 \text{Var}\left[\epsilon_i\right]}{(\sum_{i=1}^n x_i^2)^2} \\
&= \frac{\sigma^2}{\sum_{i=1}^n x_i^2}
\end{aligned}$$

(f) 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 $\text{Cov}(\epsilon_i, \epsilon_{i'}) = 0$ for all $i \neq i'$. What is the variance of the ridge estimator from (b)? How does the variance change as a function of λ?

$$\begin{aligned}
\text{Var}(\hat{\beta}_R) &= \text{Var}\left[\frac{\sum_{i=1}^n x_i y_i}{\lambda + \sum_{i=1}^n x_i^2}\right] \\
&= \frac{\text{Var}\left[\sum_{i=1}^n (3x_i^2 + x_i\epsilon_i)\right]}{(\lambda + \sum_{i=1}^n x_i^2)^2} \\
&= \frac{\sum_{i=1}^n \text{Var}\left[x_i\epsilon_i\right]}{(\lambda + \sum_{i=1}^n x_i^2)^2} \\
&= \frac{\sigma^2 \sum_{i=1}^n x_i^2}{(\lambda + \sum_{i=1}^n x_i^2)^2} \\
&= \frac{\sigma^2}{(\sum_{i=1}^n x_i^2)(1 + \frac{\lambda}{\sum_{i=1}^n x_i^2})^2}
\end{aligned}$$

For any $\lambda > 0$, the ridge regression estimator has lower variance compared to the least squares estimator. For $\lambda = 0$, the ridge estimator has the same variance as a least squares estimator, whereas as λ increases, the variance decreases. As λ becomes larger and larger, $\text{Var}(\hat{\beta}_R)$ gets closer and closer to 0, to which it eventually converges.

(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.

From (d) and (f), we saw that increasing λ increases the variance of the ridge estimator compared to the least squares estimator. However, this increase in bias reduces the variance of the ridge estimator when compared to the least squares estimator. Thus, by increasing λ, one can increase the bias and decrease the variance allowing one to control the model complexity.