# Stat 432 Homework 4

Sharvi Tomar (stomar2)

30/08/21

## Contents

Question 1: The Bias-Variance Trade-Off Simulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
Question 2: The Cross-Validation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
**What is the best penalty level you obtained from this** **7**

## Question 1: The Bias-Variance Trade-Off Simulation

Let us further extend the bias-variance trade-off simulation study in the lecture note to complete our analysis. For the original simulation study, please read the two examples in the "Bias and Variance of Ridge Regression" section to grasp the idea. And we are going to extend the ridge regression bias-variance example. For this question, you can use the same code already presented in the lecture notes, and you can remove the OLS part (anything associated with `allolsbeta`). You can keep most settings the same as the original code, but some modifications need to be done to complete this question.

- Change the covariance between $X_1$ and $X_2$ to 0.9.
- Instead of recording all $\widehat{\beta}$ values, we will only focus on the first parameter $\widehat{\beta}_1$, with true value 1. Note that we do not have the intercept term here. Out of the 1000 simulations, you will obtain 1000 of such estimated values. Compare the average of these estimations with the truth would allow us to calculate the Bias. And you can also obtain the variance of them. Make sure that you use your UIN as the random seed.

```r
library(MASS)
set.seed(667346304)
nsim = 1000
n = 100
lambda = 0.3

# save all estimated variance in a vector
allridgebeta = matrix(NA, nsim, 1)

for (i in 1:nsim)
{
  # Setting the covariance between X_1 and X_2 to 0.9
  X = mvrnorm(n, c(0, 0), matrix(c(1, 0.9, 0.9, 1), 2, 2))
  y = rnorm(n, mean = X[, 1] + X[, 2])
  # Saving the first parameter beta1
  betas=solve(t(X) %*% X + lambda * n * diag(2)) %*% t(X) %*% y
  allridgebeta[i] = betas[1]
}

# For 1000 simulations, we obtain 1000 beta1 estimated values
dim(allridgebeta)
```

```
## [1] 1000    1
```

```r
# Mean of beta1 estimates
colMeans(allridgebeta)
```

```
## [1] 0.8605662
```

```r
# Comparing mean of beta1 estimates with truth value=1 - calculating the Bias
colMeans(allridgebeta)-1
```

```
## [1] -0.1394338
```

```r
# Variance of beta1 estimates
apply(allridgebeta, 2, var)
```

```
## [1] 0.005636914
```

- You also need to perform the above task for many lambda values. Hence, you would need to write a "double-loop" with the outside loop going through all $\lambda$ values and the inside-loop being what you have done in the previous step. For the choice of $\lambda$ values, consider a grid of 100 values from 0 to 0.5. Hence, at the end of this simulation, you should have a vector of 100 bias values and 100 variance values.

```r
# Setting seed to UIN
set.seed(667346304)
nsim = 1000
n = 100
lambda_grid = seq(0, 0.5, length.out = 100)

# save estimated variance for first parameter in a vector
allridgebeta = matrix(NA, nsim, 100)

for (index in 1:length(lambda_grid)) {
  for (i in 1:nsim)
  {
    # create highly correlated variables and a linear model
    # Setting the covariance between X1 and X2 to 0.9.
    X = mvrnorm(n, c(0, 0), matrix(c(1, 0.9, 0.9, 1), 2, 2))
    y = rnorm(n, mean = X[, 1] + X[, 2])

    # Saving the first parameter beta1 at each nsim
    betas = solve(t(X) %*% X + lambda_grid[index] *
                    n * diag(2)) %*% t(X) %*% y
    allridgebeta[i, index] = betas[1]
  }
}
# Mean of beta1 estimates
colMeans(allridgebeta)
```

```
##   [1] 1.0001073 1.0054469 0.9970089 0.9850143 0.9833523 0.9841861 0.9928146
##   [8] 0.9822238 0.9690792 0.9654929 0.9719941 0.9731450 0.9750746 0.9590550
##  [15] 0.9615694 0.9577935 0.9512892 0.9627446 0.9533765 0.9510182 0.9521045
##  [22] 0.9483537 0.9465023 0.9408461 0.9368286 0.9425164 0.9325144 0.9346068
##  [29] 0.9300373 0.9232596 0.9271140 0.9239530 0.9229838 0.9132573 0.9149104
##  [36] 0.9136700 0.9095483 0.9151423 0.9087026 0.9083456 0.9016716 0.8983918
##  [43] 0.8951099 0.8941419 0.8921623 0.8890555 0.8874975 0.8866376 0.8886262
##  [50] 0.8800730 0.8799428 0.8746363 0.8746434 0.8753950 0.8675171 0.8674900
##  [57] 0.8705527 0.8643672 0.8619951 0.8607788 0.8564731 0.8598549 0.8561637
##  [64] 0.8541605 0.8533794 0.8437262 0.8463046 0.8440409 0.8437522 0.8426142
##  [71] 0.8361956 0.8365136 0.8368812 0.8332662 0.8341422 0.8311506 0.8294035
##  [78] 0.8216640 0.8277142 0.8201193 0.8235461 0.8189030 0.8147511 0.8171600
##  [85] 0.8153169 0.8099827 0.8126170 0.8079265 0.8090840 0.8070509 0.8035020
##  [92] 0.8028648 0.7962329 0.7977744 0.7952889 0.7954240 0.7929002 0.7923247
##  [99] 0.7930310 0.7888826
```

```
# Bias of beta1 estimates
bias_values=colMeans(allridgebeta)-1

# Variance of beta1 estimates
variance_values = apply(allridgebeta, 2, var)
variance_values
```

```
##   [1] 0.053644541 0.051800387 0.042713415 0.039696046 0.038707675 0.034887038
##   [7] 0.030954778 0.031056575 0.028960507 0.027027219 0.026387646 0.022554901
##  [13] 0.024110764 0.020278832 0.020011465 0.017798288 0.017607544 0.016904238
##  [19] 0.015868357 0.015431083 0.014361352 0.013726436 0.012463793 0.013991754
##  [25] 0.012766180 0.011742378 0.011776554 0.011606837 0.010502310 0.010904811
##  [31] 0.010347443 0.010118195 0.010144002 0.009135767 0.009518160 0.008772039
##  [37] 0.008417715 0.009086623 0.007979236 0.007995229 0.007551273 0.007870178
##  [43] 0.008075137 0.007314317 0.007703474 0.007416633 0.006842172 0.007389842
##  [49] 0.007305757 0.006939238 0.006355224 0.006298767 0.006480053 0.005664828
##  [55] 0.005667658 0.005580417 0.005853779 0.005605468 0.005254574 0.005832736
##  [61] 0.005641656 0.005225520 0.005386162 0.005545460 0.005320555 0.005086806
##  [67] 0.004737079 0.005471678 0.005107054 0.004505692 0.004856518 0.004566281
##  [73] 0.004481399 0.004824319 0.004575875 0.004561380 0.004602742 0.004589617
##  [79] 0.004574788 0.004811606 0.004370537 0.004289735 0.004100251 0.004155211
##  [85] 0.003850570 0.004237285 0.004199785 0.004260861 0.004180988 0.004155693
##  [91] 0.003924152 0.004118907 0.004139284 0.003847638 0.003999404 0.003952695
##  [97] 0.003937342 0.004121274 0.004173435 0.004014504
```

```
# Length of bias_values vector
length(bias_values)
```

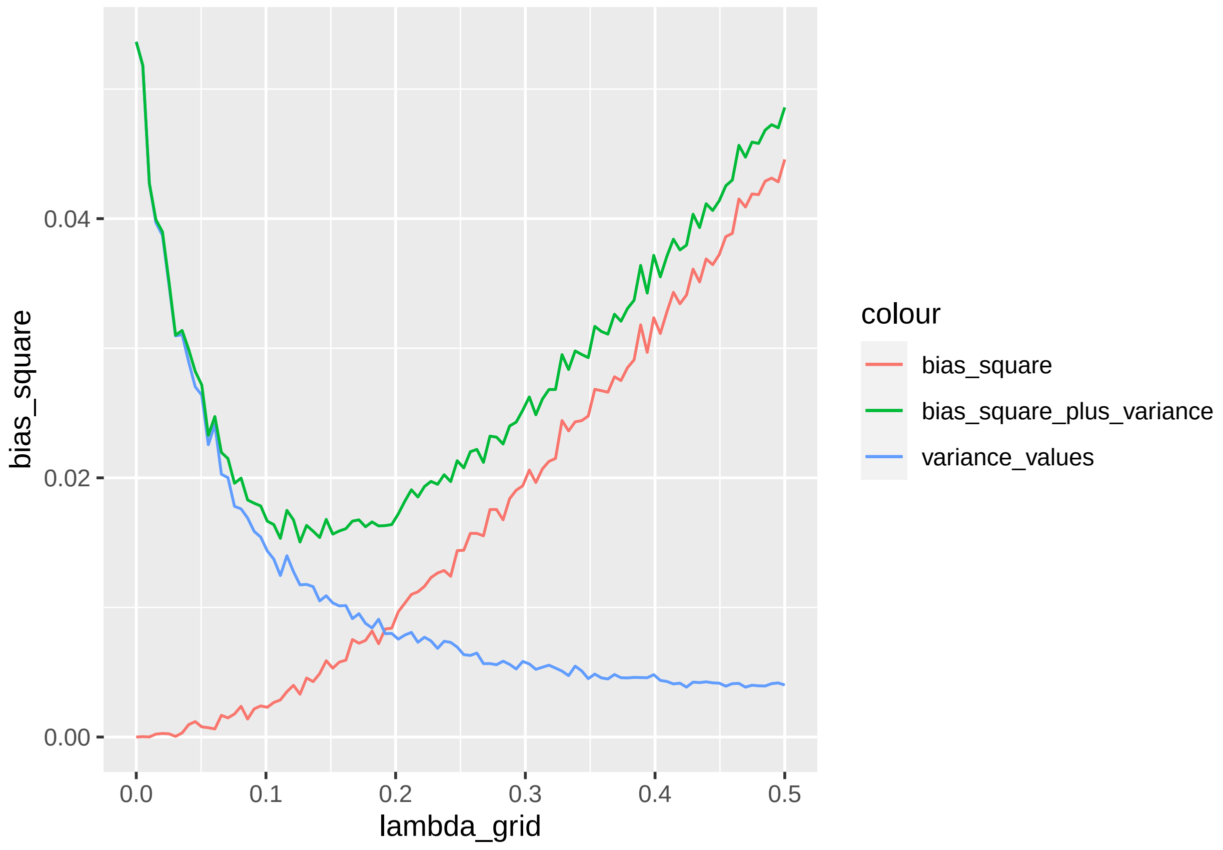
```
## [1] 100
```

```r
# Length of variance_values vector
length(variance_values)
```

```
## [1] 100
```

- Make a plot of three quantities over each of your $\lambda$ values: Bias$^2$, Variance, and Bias$^2$ + Variance. Hence there should be three curves over the range of lambda from 0 to 0.5. My curve looks like the following, but yours may differ based on your specific setup and the random seed.

```r
bias_square=bias_values^2
bias_square_plus_variance=(bias_values^2)+variance_values
df=data.frame(lambda_grid,bias_square,variance_values,bias_square_plus_variance)
library(ggplot2)
ggplot(df, aes(x=lambda_grid)) +
  geom_line(aes(y = bias_square, color= "bias_square")) +
  geom_line(aes(y = variance_values, color= "variance_values")) +
  geom_line(aes(y = bias_square_plus_variance,color= "bias_square_plus_variance"))
```



- Lastly, what have you observed in terms of the trend for Bias$^2$, Variance, and their sum, respectively? What is causing these? And if you are asked to choose a $\lambda$ value that works the best, which value would you choose?

With increase in value of lambda, the variance(in blue) decreases however, the square of the bias/absolute value of bias(in red) increases. Since we aim to ultimately reduce prediction error which is the sum of

(Bias2+Variance) and a term called irreducible error, we focus to reduce the (Bias2+Variance). Hence, the lambda value I will choose would be the one that minimizes (Bias2+Variance).

```
# Value of lambda where bias_square_plus_variance is minimum
lambda_grid[which.min(bias_square_plus_variance)]
```

```
## [1] 0.1262626
```

Lambda=0.126262 works the best.

## Question 2: The Cross-Validation

We used the `mtcars` data in the lecture notes, and also introduced the *k*-fold cross-validation. For this question you need to complete the following:

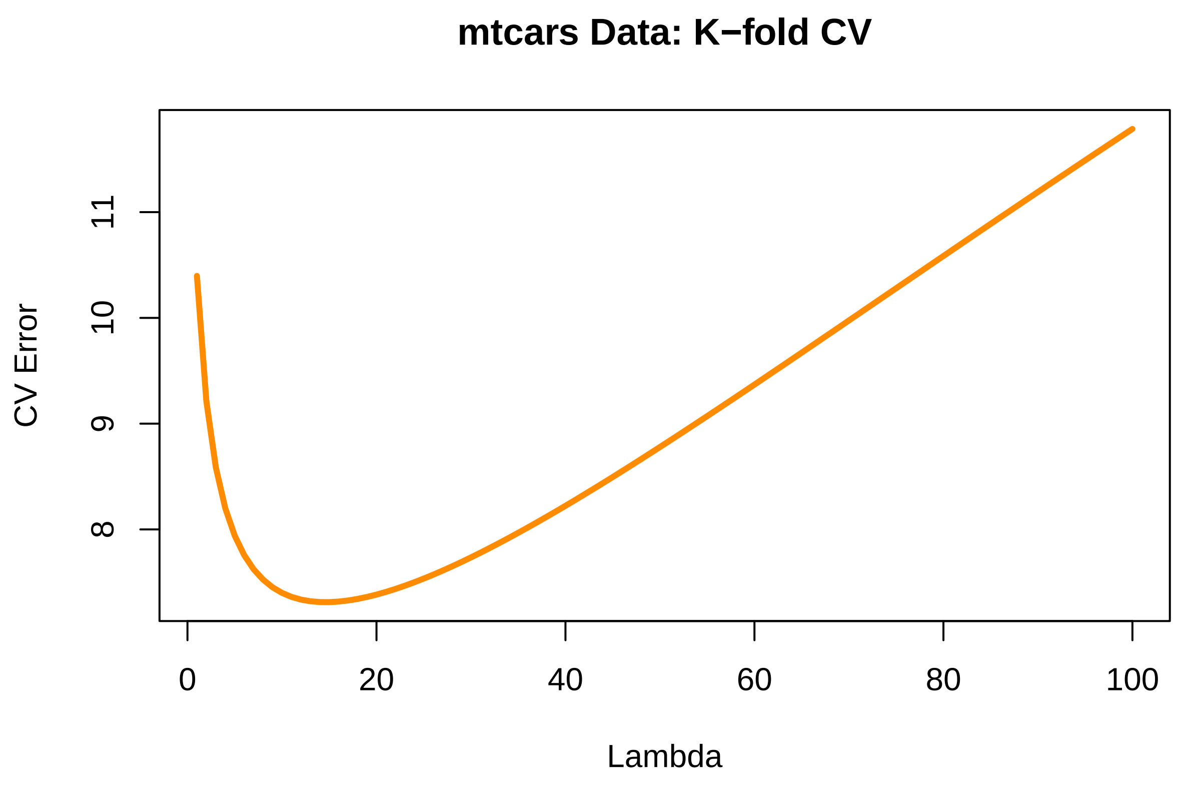
- Write a 5-fold cross-validation code by yourself, using the `lm.ridge()` function to fit the model and predict on the testing data. Choose an appropriate range of lambda values based on how this function specifies the penalty. Obtain the cross-validation error corresponding to each $\lambda$ and produce an intuitive plot to how it changes over different $\lambda$. What is the best penalty level you obtained from this procedure? Compare that with the GCV result. Please note that you should clearly state the intention of each step of your code and state your result. For details regrading writing a report, please watch the `Comment Video on HW` from week 1 webpage, or the discussion broad.

```
# 5-fold cross-validation code using lm.ridge()

library(MASS)
testing_error = matrix(NA, 100, 5)
avg_testing_error = rep(NA, 100)
lambda_seq = seq(1, 100, length.out = 100)

# Making folds of 7,6,6,6,7
folds <- cut(seq(1,nrow(mtcars)),breaks=5,labels=FALSE)

for (k in 1:5) {
  testIndexes <- which(folds==k,arr.ind=TRUE)
  #print(testIndexes)
  test <- mtcars[testIndexes, ]
  train <- mtcars[-testIndexes, ]

  train_data=train[,-c(1)]
  fit1 = lm.ridge(train$mpg ~ .,
                  data = train_data,
                  lambda = seq(1, 100, length.out = 100))

  # fit1 = lm.ridge(mpg ~ .,
  #                 data = train,
  #                 lambda = seq(1, 100, length.out = 100))

  ## add intercept column, remove mpg column
  test_data = cbind(1, test)
  test_data = test_data[-c(2)]
```

```r
  for (i in 1:100) {
    # predict on the testing data
    y_pred = data.matrix(test_data) %*% coef(fit1)[i,]
    # For each k, obtaining the testing error
    testing_error[i, k] = mean((y_pred - test$mpg) ^ 2)
  }
}
```

```r
# Average all K testing errors
avg_testing_error = apply(testing_error, 1, mean)
```

```r
# Plot to show how cross-validation error changes over different lambda
plot(
  lambda_seq,
  avg_testing_error,
  type = "l",
  col = "darkorange",
  ylab = "CV Error",
  xlab = "Lambda",
  lwd = 3
)
title("mtcars Data: K-fold CV")
```



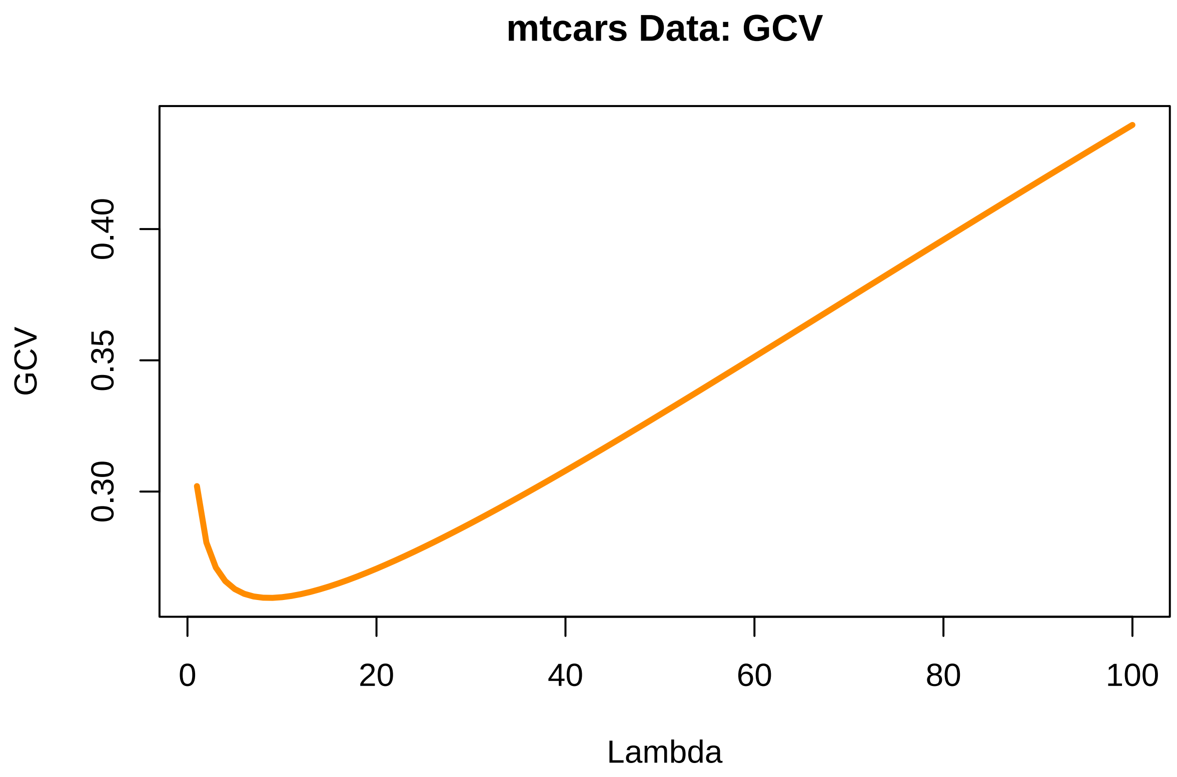
## What is the best penalty level you obtained from this

```
lambda_seq[which.min(avg_testing_error)]
```

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

The best penalty level obtained using the 5-fold cross-validation code above is 15.

```
# Comparing that with the GCV result
 plot(fit1$lambda[1:100], fit1$GCV[1:100], type = "l", col = "darkorange",
         ylab = "GCV", xlab = "Lambda", lwd = 3)
    title("mtcars Data: GCV")
```



```
fit1$lambda[which.min(fit1$GCV)]
```

```
## [1] 9
```

The lambda value obtained from GCV approach is 9.

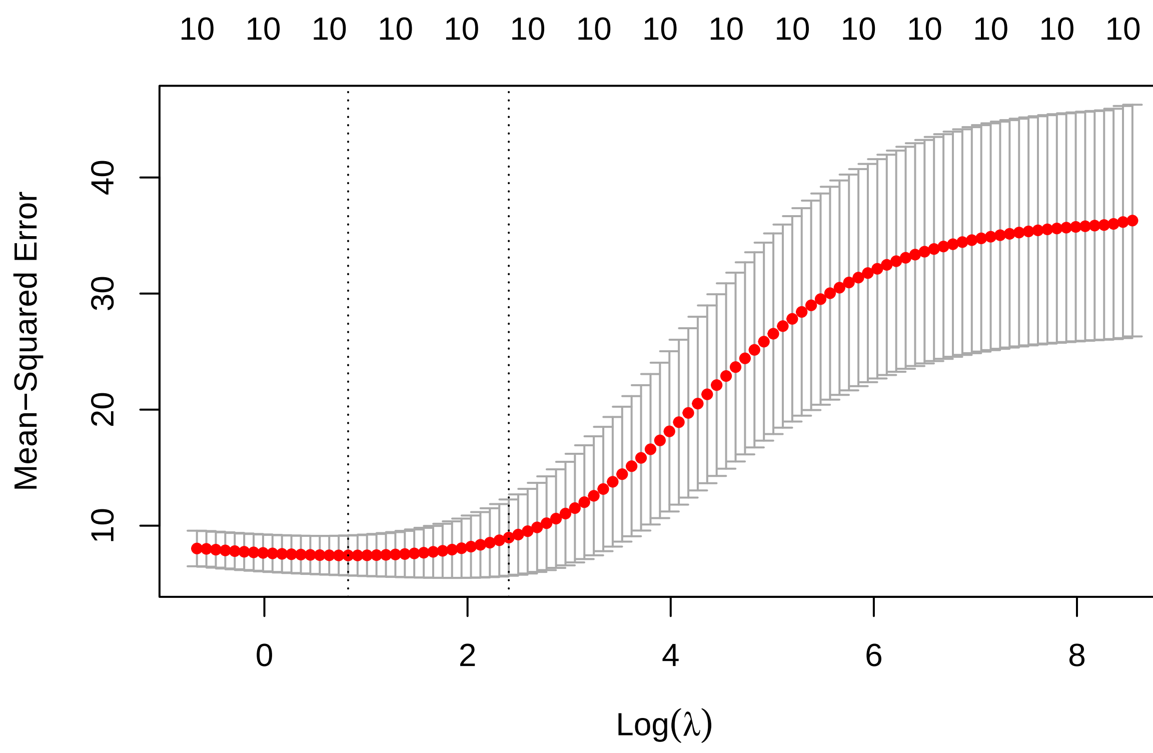
- Use the `cv.glmnet()` function from the `glmnet` package to perform a 5-fold cross-validation using their built-in feature. Produce the cross-validation error plot against $\lambda$ values. Report the `lambda.min` and `lambda.1se` selected $\lambda$ value.

```
# Use cv.glmnet() from the glmnet package to perform a 5-fold cross-validation
library(glmnet)
```

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

```
## Loaded glmnet 4.1-2
```

```
set.seed(667346304)
fit2 = cv.glmnet(
  x = data.matrix(mtcars[,-1]),
  y = mtcars$mpg,
  nfolds = 5,
  alpha = 0
)

# Plotting cross-validation error against   values
plot(fit2)
```



```
# lambda.min value
fit2$lambda.min
```

```
## [1] 2.280432
```

```r
# lambda.1se value
fit2$lambda.1se
```

```
## [1] 11.08883
```