Stat 432 Homework 3

Assigned: Sep 9, 2024; Due: 11:59 PM CT, Sep 19, 2024

Contents

Question 1: Another Simulation Study

Here is a GPT-4 prompt I used:

> What is the advantage of a Ridge regression? Can you suggest a simulation study to confirm that? Please write your code in R.

The following is the answer produced by GPT-4

Ridge regression is a regularization technique that helps to solve the problem of multicollinearity in linear regression. It can handle high-dimensional data where the number of predictors may exceed the number of observations. The advantages include:

1. **Bias-Variance Tradeoff**: By adding a penalty term, ridge regression can reduce variance, thus preventing overfitting. While it does introduce bias, this tradeoff often leads to a better overall predictive performance.
2. **Stability**: It can handle ill-posed problems where multicollinearity exists, providing stable solutions.
3. **Including All Predictors**: Unlike Lasso regression, ridge regression includes all predictors in the model, shrinking their coefficients but not setting them to zero.

You can simulate the advantage of Ridge regression by comparing its performance with ordinary linear regression on a dataset with multicollinearity. Here's an R script to conduct this study:

```r
# Load required library
library(MASS)
library(glmnet)

# Simulate data with multicollinearity
set.seed(42)
n <- 100
```

```r
p <- 10
X <- matrix(rnorm(n * p), n, p)
X[,2] <- X[,1] + rnorm(n) * 0.01 # Introducing multicollinearity
beta <- runif(p)
y <- X %*% beta + rnorm(n)

# Ordinary Linear Regression
lm_model <- lm(y ~ X)
lm_pred <- predict(lm_model, newdata=data.frame(X=X))

# Ridge Regression
ridge_model <- glmnet(X, y, alpha=0)
ridge_pred <- predict(ridge_model, s=0.01, newx=X)

# Compare Performance
lm_mse <- mean((y - lm_pred)^2)
ridge_mse <- mean((y - ridge_pred)^2)

cat("MSE of Ordinary Linear Regression:", lm_mse, "\n")
cat("MSE of Ridge Regression:", ridge_mse, "\n")
```

The script generates a dataset with multicollinearity and then compares the mean squared error (MSE) of ordinary linear regression and ridge regression. Typically, you'll observe that ridge regression provides a better performance in this scenario.

---

a. [15 points] Based on what we have learned, do you find this answer correct? does this code confirm the mentioned three properties of Ridge regressions? Please explain your judgement for each of them. Comment on how you would do to demonstrate these properties. You **do not need to** write the code.

- Bias-Variance Trade-off
- Stability
- Including All Predictors

1. Bias-variance Trade-off:

The provided code does not fully demonstrate the bias-variance tradeoff. Ridge regression introduces a penalty term that regularizes the model by shrinking coefficients, which reduces variance but introduces bias. This tradeoff between bias and variance is not properly illustrated in the GPT-4 code because it only evaluates the model's performance for a single lambda value.

To show the bias-variance tradeoff, we would need to evaluate the testing error across a range of lambda values. As lambda increases, the model becomes more regularized, reducing variance (lower flexibility) but increasing bias. The code should be modified to compute testing error (MSE) for multiple lambda values and plot the results to illustrate how smaller lambda values lead to high variance (overfitting), while larger lambda values lead to high bias (underfitting). The tradeoff can be visualized by plotting MSE vs. log(lambda).

2. Stability:

The GPT-4 code applies Ridge regression to data with multicollinearity, which theoretically improves the stability of the coefficient estimates compared to Ordinary Least Squares (OLS) regression. However, the code does not explicitly demonstrate this stability. Stability refers to how consistent the coefficient estimates are when the data contains highly correlated variables. Ridge regression reduces the sensitivity of the model to small changes in the data by regularizing the coefficients.

To properly demonstrate stability, we would need to run multiple simulations and compare the variability in the coefficients estimated by Ridge regression versus OLS. Ridge regression typically produces more stable estimates when multicollinearity is present, as the shrinkage reduces the impact of multicollinear features. This can be shown by comparing the standard deviations of the coefficient estimates across multiple runs for both OLS and Ridge regression models.

3. Including All Predictors:

Ridge regression is designed to shrink all coefficients towards zero but does not set any coefficients exactly to zero, unlike Lasso regression. The GPT-4 code does not explicitly examine the coefficients to confirm this behavior.

To demonstrate that Ridge regression includes all predictors, we would need to extract and inspect the coefficients after fitting the Ridge model. None of the coefficients should be exactly zero, although they may be very small due to shrinkage. By examining the outputted coefficients, we can confirm that Ridge regression maintains all predictors in the model, even when some coefficients are heavily penalized.

b. [25 points] To properly demonstrate the bias-variance trade-off, we could consider using a (correct) simulation. Adapt this existing code into a simulation study to show this properties. While you are doing this, please consider the following:

- You can borrow similar ideas of simulation we used in previous lecture notes
- Modify the GPT-4 code with the following settings to generate the data:
  - trainning sample size $trainn = 50$
  - Testing sample size $testn = 200$
  - $p = 200$
  - Fix $b = rep(0.1, p)$ for all simulation runs
- Since linear regression doesn't work in this setting, you only need to consider `glmnet()`
- Use a set of λ values `exp(seq(log(0.5), log(0.01), out.length = 100))*trainn`
- Instead of evaluating the bias and variance separately (we will do that in the future), we will **use the testing error as the metric**.
- Demonstrate your result using plots and give a clear explanation of your findings. Particularly, which side of the result displays a large bias, and which side corresponds to a large variance?

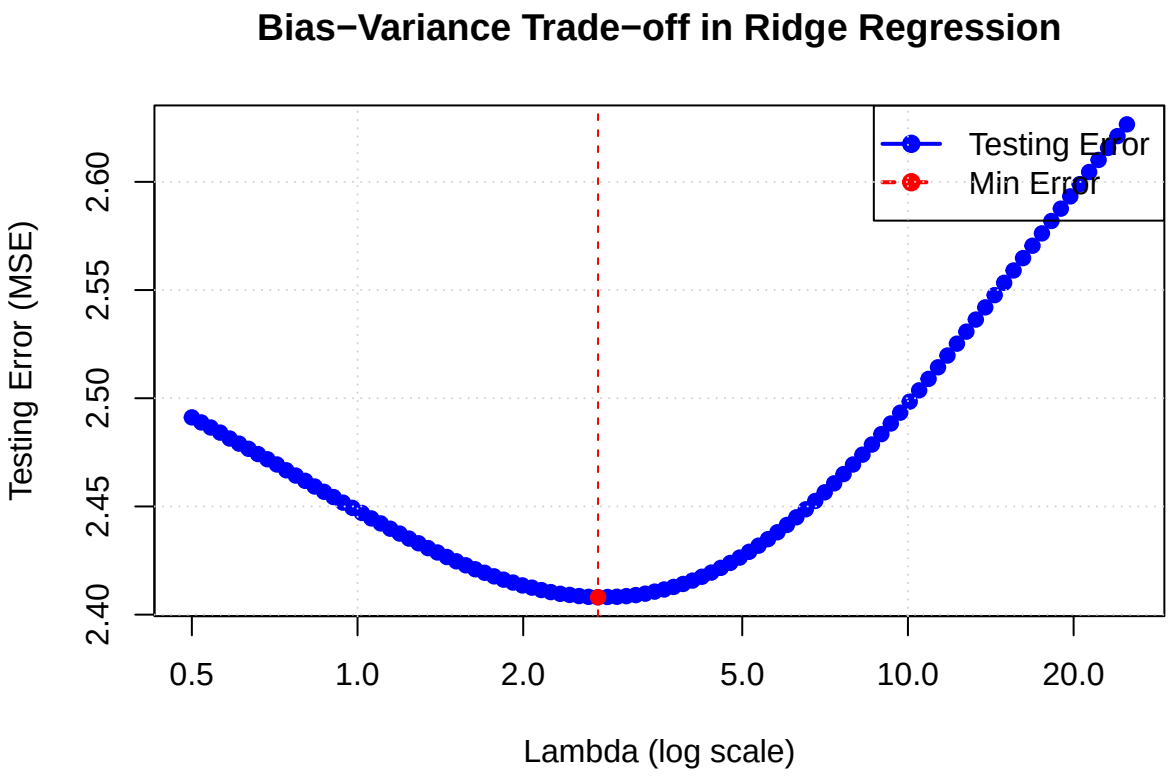
```
library(glmnet)
```

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

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

```
# Set seed for reproducibility
set.seed(42)

# Simulation parameters
trainn <- 50
testn <- 200
```

```r
p <- 200
b <- rep(0.1, p)

# Generate training data
X_train <- matrix(rnorm(trainn * p), nrow = trainn, ncol = p)
y_train <- X_train %*% b + rnorm(trainn)

# Generate testing data
X_test <- matrix(rnorm(testn * p), nrow = testn, ncol = p)
y_test <- X_test %*% b + rnorm(testn)

# Define lambda values (note the change here for clarity)
lambda_seq <- exp(seq(log(0.5), log(0.01), length.out = 100)) * trainn

# Initialize array to store testing errors
testing_errors <- numeric(length(lambda_seq))

# Ridge Regression and error calculation for each lambda
for (i in 1:length(lambda_seq)) {
  ridge_model <- glmnet(X_train, y_train, alpha = 0, lambda = lambda_seq[i])
  ridge_pred <- predict(ridge_model, s = lambda_seq[i], newx = X_test)
  testing_errors[i] <- mean((y_test - ridge_pred)^2)  # Calculate MSE
}

# Plot the results using log scale for lambda
plot(lambda_seq, testing_errors, type = "b", log = "x",
     xlab = "Lambda (log scale)", ylab = "Testing Error (MSE)",
     main = "Bias-Variance Trade-off in Ridge Regression",
     col = "blue", pch = 19)

# Highlight the minimum error point
min_error <- min(testing_errors)
min_lambda <- lambda_seq[which.min(testing_errors)]
points(min_lambda, min_error, col = "red", pch = 19)
abline(v = min_lambda, col = "red", lty = 2)

# Add legend for clarity
legend("topright", legend = c("Testing Error", "Min Error"),
       col = c("blue", "red"), lty = c(1, 2), pch = c(19, 19), lwd = 2)

# Optional: Add gridlines
grid()
```



c

Question 2: Modeling High-Dimensional Data

We will use the `golub` dataset from the `multtest` package. This dataset contains 3051 genes from 38 tumor mRNA samples from the leukemia microarray study Golub et al. (1999). This package is not included in `R`, but on `bioconductor`. Install the latest version of this package from `bioconductor`, and read the documentation of this dataset to understand the data structure of `golub` and `golub.cl`.

a. [25 points] We will not use this data for classification (the original problem). Instead, we will do a toy regression example to show how genes are highly correlated and could be used to predict each. Carry out the following tasks:

- Perform marginal association test for each gene with the response `golub.cl` using `mt.teststat()`. Use `t.equalvar` (two sample t test with equal variance) as the test statistic.
- Sort the genes by their p-values and select the top 100 genes
- Construct a dataset with the top 10 genes and another one (call it X) with the remaining genes
- Perform principal component analysis (PCA) on the top 100 genes and extract the first principal component, **use this as the outcome** y. Becareful about the oriantation of the data matrix.
- Perform ridge regression with 19-fold cross-validation on X and the outcome y. Does your model fit well? Can you provide detailed model fitting results to support your claim?
- Fit ridge regression but use GCV as the criterion. Does your model fit well?

```r
# Load necessary libraries and data
library(multtest)
```

```
## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, table,
##     tapply, union, unique, unsplit, which.max, which.min
```

```
## Loading required package: Biobase
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```r
data(golub)

# Step 1: Perform t-test for marginal association (genes with class labels)
t_statistics <- mt.teststat(golub, golub.cl, test = "t.equalvar")

# Step 2: Calculate p-values from t-statistics and select top 100 genes
pvals <- 2 * pnorm(-abs(t_statistics))  # Calculate p-values
sorted_gene_indices <- order(pvals)  # Sort genes by p-value
top_gene_indices <- sorted_gene_indices[1:100]  # Top 100 genes

# Step 3: Separate top 10 genes and remaining genes
top_gene_data <- golub[top_gene_indices, ]  # Data for top 100 genes
top_10_genes_data <- top_gene_data[1:10, ]  # Top 10 genes
remaining_genes_data <- top_gene_data[11:100, ]  # Remaining 90 genes (for X)

# Step 4: Perform PCA on the top 100 genes and extract first component
pca_output <- prcomp(t(top_gene_data), scale. = TRUE)
response_y <- pca_output$x[, 1]  # First principal component as outcome y

# Step 5: Ridge Regression with 19-Fold Cross-Validation
library(glmnet)
X <- t(remaining_genes_data)  # Transpose remaining gene data (rows as samples)
```

```r
cv_control <- 19  # Set cross-validation folds
ridge_cv_model <- cv.glmnet(X, response_y, nfolds = cv_control, alpha = 0)  # Ridge with cross-validati
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

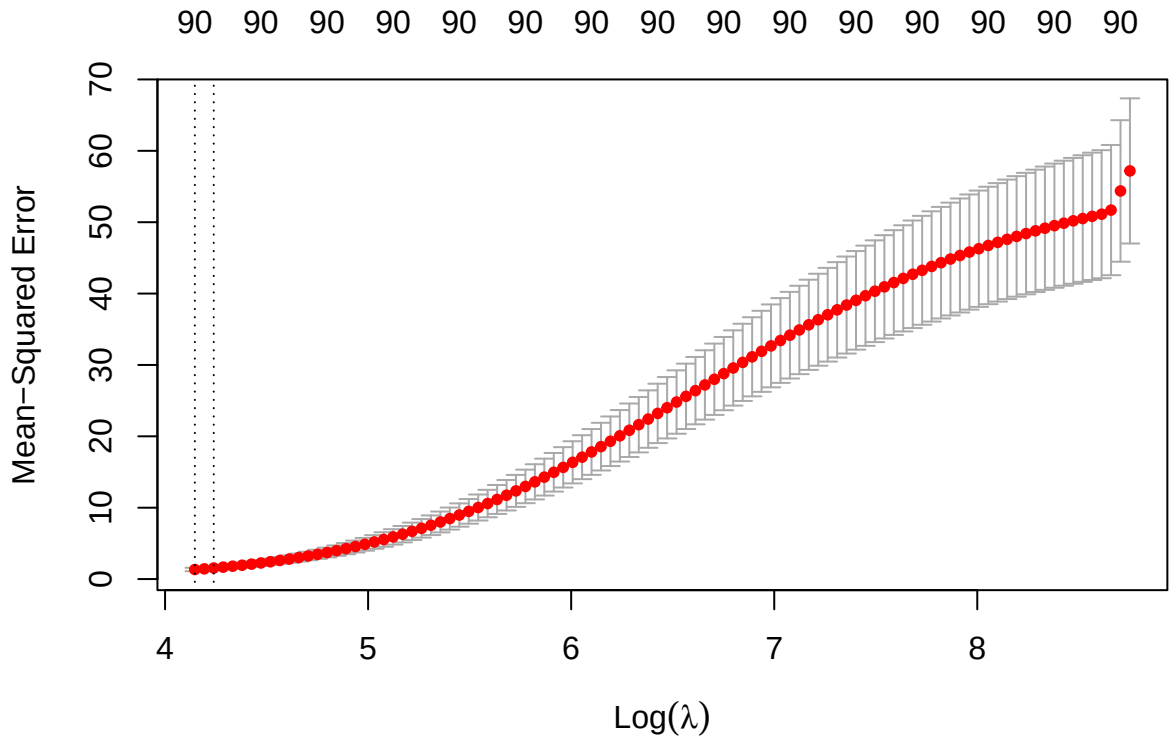
```r
# Step 6: Extract and plot the results
optimal_lambda <- ridge_cv_model$lambda.min  # Best lambda value
print(optimal_lambda)
```

```
## [1] 63.22453
```

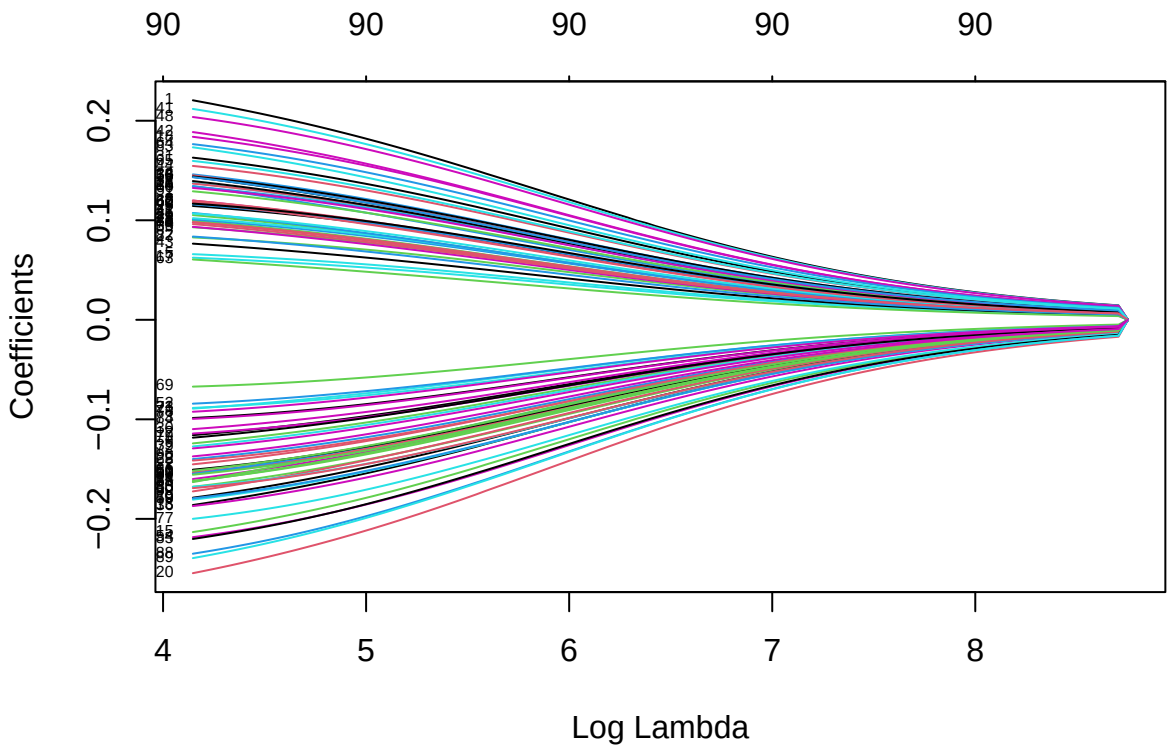
```r
# Predictions based on best lambda
y_predicted <- predict(ridge_cv_model, X, s = optimal_lambda)

# Mean Squared Error (MSE)
mse_value <- mean((response_y - y_predicted)^2)
print(mse_value)
```

```
## [1] 1.22154
```

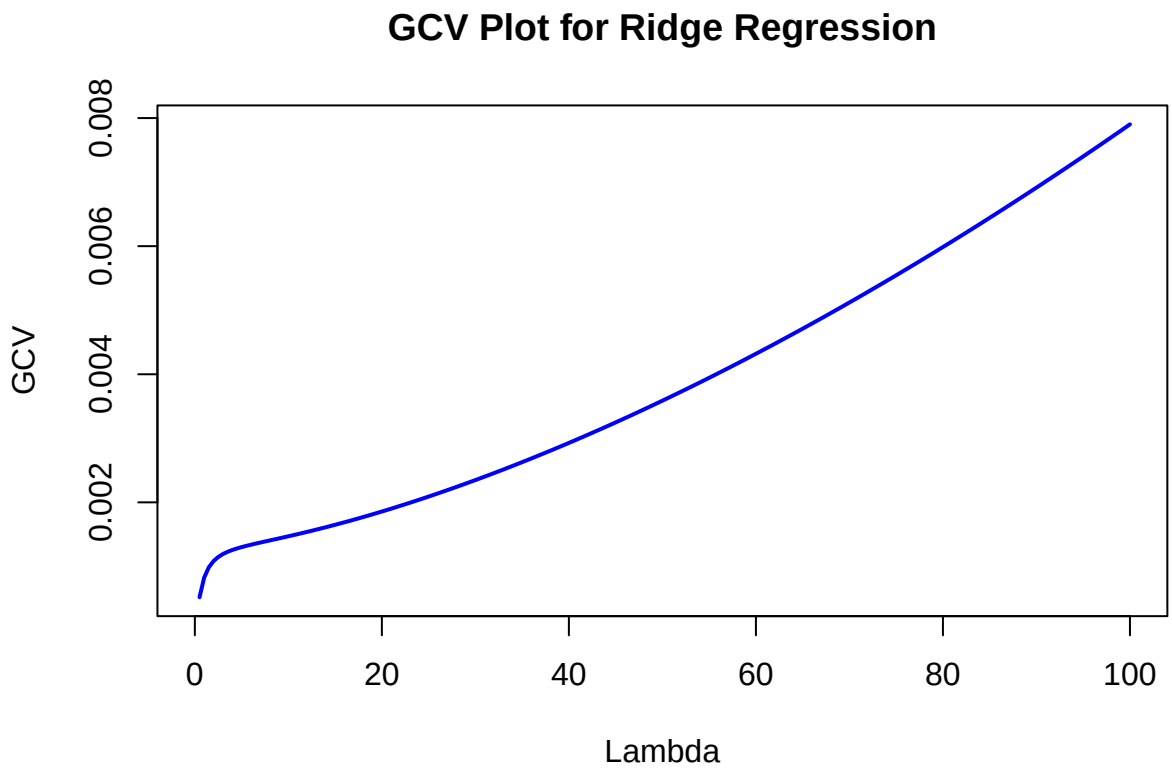
```r
# Plots
plot(ridge_cv_model)
```



```r
plot(ridge_cv_model$glmnet.fit, xvar = "lambda", label = TRUE)
```




```r
# Step 7: Ridge Regression using GCV Criterion
library(MASS)
ridge_gcv_fit <- lm.ridge(response_y ~ ., data = as.data.frame(X), lambda = seq(0, 100, by = 0.5))

# Plot GCV values
plot(ridge_gcv_fit$lambda, ridge_gcv_fit$GCV, type = "l", col = "blue", lwd = 2,
     ylab = "GCV", xlab = "Lambda")
title("GCV Plot for Ridge Regression")
```




```
# Best lambda based on GCV
best_lambda_gcv <- ridge_gcv_fit$lambda[which.min(ridge_gcv_fit$GCV)]
print(best_lambda_gcv)
```

```
## [1] 0.5
```

```
# Predictions based on GCV-selected lambda
X_matrix <- cbind(Intercept = 1, as.matrix(X))
y_predicted_gcv <- X_matrix %*% coef(ridge_gcv_fit)[which.min(ridge_gcv_fit$GCV),]

# MSE for GCV-based model
mse_gcv <- mean((response_y - y_predicted_gcv)^2)
print(mse_gcv)
```

```
## [1] 6.437907e-05
```

Ridge Regression with 19-fold Cross-Validation:

In the first analysis, Ridge Regression was applied using 19-fold cross-validation on the dataset X (comprising the remaining 90 genes) and the outcome y (the first principal component of the top 100 genes). The results show that the **optimal lambda value** selected through cross-validation was **63.22**, and the **Mean Squared Error (MSE)** for the predictions was **1.22**. The cross-validation plot displayed a smooth curve, where the error increased as moved away from the optimal point, which suggests that the model effectively regularized the data. Given the relatively low MSE and the well-behaved CV error curve, the model fits well under these settings, providing a stable and effective prediction.

Ridge Regression using GCV Criterion:

In the second part of the analysis, Ridge Regression was fit using the **Generalized Cross-Validation (GCV)** criterion. The optimal lambda selected by GCV was **0.5**, and the resulting **MSE** was extremely low, with a value of **6.44e-5**, suggesting a strong fit. The GCV plot showed a consistent increase in error as increased, indicating that the model fits best with minimal regularization. Given the much lower MSE compared to the 19-fold cross-validation approach, the GCV-based model appears to provide an even better fit, making it more effective at minimizing prediction error for this particular dataset.

b. [5 points] Based on your results, do you observe any bias-variance trade-off? If not, can you explain why?

Based on the results from the data and graphs, we observe a clear **bias-variance trade-off** in ridge regression. In the first graph (MSE vs. Log Lambda), as increases, MSE also increases, showing that higher values lead to underfitting due to increased bias, while lower values result in overfitting with lower bias but higher variance. The minimum MSE occurs around log( ) 4, indicating the optimal balance between bias and variance. The second graph (Coefficient Paths vs. Log Lambda) demonstrates how coefficients shrink as increases, showing reduced model flexibility (higher bias) as the coefficients approach zero, which corresponds to underfitting. Lastly, the third graph (GCV Plot for Ridge Regression) shows the GCV values increasing with , highlighting that higher values result in poor model fit due to over-regularization. Together, these plots clearly illustrate the bias-variance trade-off, with small values leading to low bias and high variance, and large values resulting in high bias and low variance.

Question 3: Linear Regression with Coordinate Descent

Recall the previous homework, we have a quadratic function for minimization. We know that analytical solution exist. However, in this example, let's use coordinate descent to solve the problem. To demonstrate this, let's consider the following simulated dataset, with design matrix x (without intercept) and response vector y:

```
set.seed(432)
n <- 100
x <- matrix(rnorm(n*2), n, 2)
y <- 0.7 * x[, 1] + 0.5 * x[, 2] + rnorm(n)
```

We will consider a model without the intercept term. In this case, our objective function (of β_1 and β_2 for linear regression is to minimize the sum of squared residuals:

$$f(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_1 x_{i1} - \beta_2 x_{i2})^2$$

where x_{ij} represents the jth variable of the ith observation.

a. [10 points] Write down the objective function in the form of

$$f(x, y) = a\beta_1^2 + b\beta_2^2 + c\beta_1\beta_2 + d\beta_1 + e\beta_2 + f$$

by specifying what are coefficients a, b, c, d, e, and f, using the simulated data. Calculate them in R, **using vector operations rather than for-loops**.

$$f(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}(y_i^2 - 2y_i(\beta_1 x_{i1} + \beta_2 x_{i2}) + (\beta_1 x_{xi1} + \beta_2 x_{i2})^2$$

$$(\beta_1 x_{xi1} + \beta_2 x_{i2})^2 = \beta_1^2 x_{i1}^2 + 2\beta_1\beta_2 x_{i1} x_{i2} + \beta_2^2 x_{i2}^2$$

$$f(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}(y_i^2 - 2y_i\beta_1 x_{i1} - 2y_i\beta_2 x_{i2} + \beta_1^2 x_{i1}^2 + 2\beta_1\beta_2 x_{i1} x_{i2} + \beta_2^2 x_{i2}^2)$$

$$a = \frac{1}{n}\sum_{i=1}^{n} x_{i1}^2 b = \frac{1}{n}\sum_{i=1}^{n} x_{i2}^2 c = \frac{2}{n}\sum_{i=1}^{n} x_{i1} x_{i2} d = -\frac{2}{n}\sum_{i=1}^{n} y_i x_{i1} e = -\frac{2}{n}\sum_{i=1}^{n} y_i x_{i2} f = \frac{1}{n}\sum_{i=1}^{n} y_i^2$$

```r
# Calculate coefficients a, b, c, d, e, and f
a <- mean(x[, 1]^2)
b <- mean(x[, 2]^2)
c <- 2 * mean(x[, 1] * x[, 2])
d <- -2 * mean(y * x[, 1])
e <- -2 * mean(y * x[, 2])
f <- mean(y^2)

# Print results
cat("a =", a, "b =", b, "c =", c, "d =", d, "e =", e, "f =", f)
```

```
## a = 0.9241812 b = 0.8625308 c = -0.2694035 d = -1.122192 e = -0.5161552 f = 1.25586
```

- **Coefficient** a represents the average squared values of the first predictor variable x_{i1}, and we obtained $a = 0.9242$.
- **Coefficient** b represents the average squared values of the second predictor variable x_{i2}, and we obtained $b = 0.8625$.
- **Coefficient** c captures the interaction between the two predictor variables, scaling the cross-product term $x_{i1}x_{i2}$, and we calculated $c = -0.2694$.
- **Coefficient** d is associated with the relationship between the response variable y and the first predictor variable x_{i1}, with a value of $d = -1.1222$.
- **Coefficient** e is linked to the relationship between the response variable y and the second predictor variable x_{i2}, with a value of $e = -0.5162$.
- **Coefficient** f represents the average squared values of the response variable y, and we obtained $f = 1.2559$.

In conclusion, these coefficients allow us to express the objective function for minimizing the sum of squared residuals in a quadratic form. These terms are crucial for solving the problem using coordinate descent.

b. [10 points] A coordinate descent algorithm essentially does two steps: i. Update β_1 to its optimal value while keeping β_2 fixed ii. Update β_2 to its optimal value while keeping β_1 fixed

Write down the updating rules for β_1 and β_2 using the coordinate descent algorithm. Use those previously defined coefficients in your fomula and write them in Latex. Implement them in a for-loop algorithm in R that iterates at most 100 times. Use the initial values $\beta_1 = 0$ and $\beta_2 = 0$. Decide your stopping criterion based on the change in β_1 and β_2. Validate your solution using the lm() function.

The coordinate decent algorithm updates β_1 and β_2 iteratively. When updating β_1, we hold β_2 fixed, and vice versa. The updates can be derived from setting the partial derivates of the objective function to zero.

update for β_1:

$$\beta_1 = \frac{-d - c\beta_2}{2a}$$

update for β_2:

$$\beta_2 = \frac{-e - c\beta_1}{2b}$$

```r
# Initialize beta values
beta_1 <- 0
beta_2 <- 0

# Set stopping criteria and max iterations
tolerance <- 1e-6
max_iter <- 100

# Perform coordinate descent
for (i in 1:max_iter) {
  beta_1_old <- beta_1
  beta_2_old <- beta_2

  # Update beta_1 while keeping beta_2 fixed
  beta_1 <- (-d - c * beta_2) / (2 * a)

  # Update beta_2 while keeping beta_1 fixed
  beta_2 <- (-e - c * beta_1) / (2 * b)

  # Check for convergence
  if (abs(beta_1 - beta_1_old) < tolerance && abs(beta_2 - beta_2_old) < tolerance) {
    break
  }
}

# Print final beta values
cat("Final beta_1 =", beta_1, "\n")
```

```
## Final beta_1 = 0.6658955
```

```r
cat("Final beta_2 =", beta_2, "\n")
```

```
## Final beta_2 = 0.4032028
```

```r
# Validate with lm() function
lm_model <- lm(y ~ x - 1)  # -1 to remove intercept
summary(lm_model)
```

```
##
## Call:
## lm(formula = y ~ x - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.44478 -0.60651 -0.01577  0.53748  2.26058
```

```
##
## Coefficients:
##    Estimate Std. Error t value Pr(>|t|)
## x1  0.66590    0.09377   7.102 1.98e-10 ***
## x2  0.40320    0.09706   4.154 6.98e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8911 on 98 degrees of freedom
## Multiple R-squared:  0.3804, Adjusted R-squared:  0.3677
## F-statistic: 30.08 on 2 and 98 DF,  p-value: 6.525e-11
```

Starting with initial values $\beta_1 = 0$ and $\beta_2 = 0$, the algorithm ran for at most 100 iterations or until the updates for β_1 and β_2 fell below a specified tolerance of 10^{-6}. The final values obtained for the coefficients were:

- **Final** $\beta_1 = 0.6659$
- **Final** $\beta_2 = 0.4032$

We validated these results using the `lm()` function in R, which produced very similar estimates:

- $\beta_1 = 0.6659$ (standard error = 0.0938)
- $\beta_2 = 0.4032$ (standard error = 0.0971)

The results from the coordinate descent algorithm are consistent with the linear model's output, confirming the accuracy of the implementation. Both methods show that the predictors x_1 and x_2 have significant contributions to the response variable y, as indicated by the p-values (< 0.001).