

# Stat 432 Homework 3

Jason Li (bl34)

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

## Contents

Question 1: Another Simulation Study . . . . .	1
Question 2: Modeling High-Dimensional Data . . . . .	4
Question 3: Linear Regression with Coordinate Descent . . . . .	6

## 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:

```

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

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

### Answer:

Based on what we have learned, the answer was not fully correct. While Ridge regression does have the property of bias-variance trade-off, the code above does not confirm this property, because the code compares the MSE instead of analyzing the variance and bias. We can demonstrate this property by performing multiple simulations with different penalty values and plotting the variance and bias across simulations to visualize the trade-off.

Additionally, the second property is correct as well, but the code fails to display it because it only fitted the models on one data sample. To display this property, we can resample the data several times and fit both models accordingly. Then we can compare how the coefficients varies.

Lastly, Ridge regression does include all predictors, but OLS also includes all predictors; therefore, in order to demonstrate this advantage, we need to compare the Ridge regression model with Lasso regression instead. We also need to print out the coefficients of the Ridge regression model to display that they are not set to zero.

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:
  - training 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  $\lambda$  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)

## Warning:  'glmnet' R 4.2.3

##      Matrix

## Loaded glmnet 4.1-7

set.seed(42)

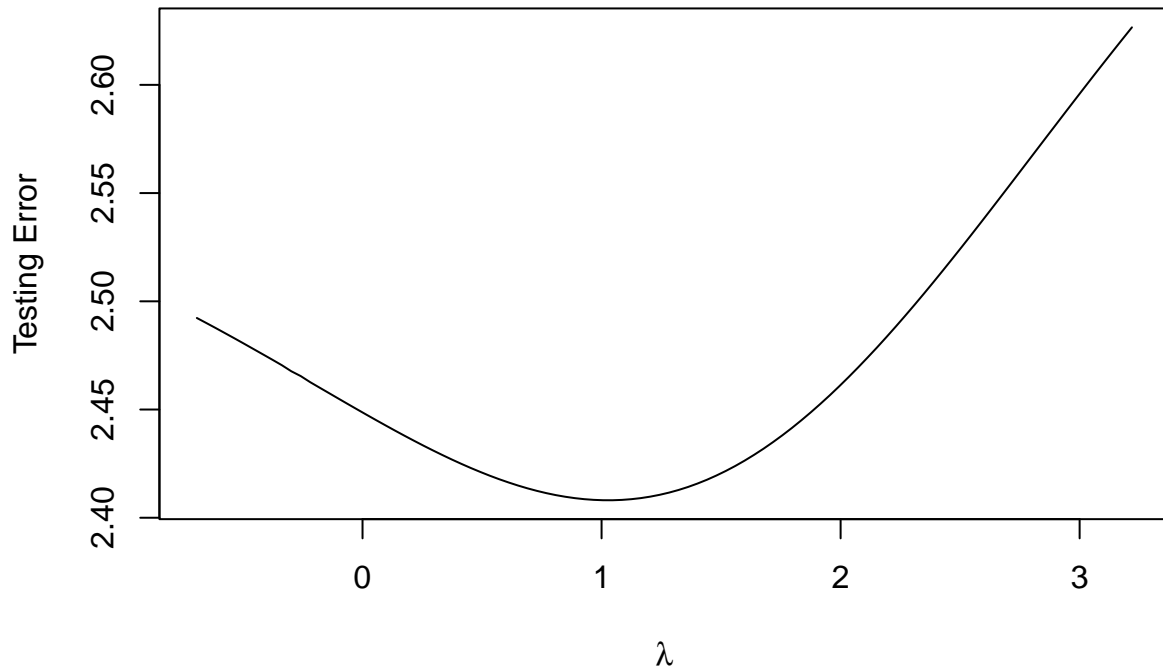
trainn <- 50
testn <- 200
p <- 200
b <- rep(0.1, p)
lambda_vals <- exp(seq(log(0.5), log(0.01), length.out = 100)) * trainn

X_train <- matrix(rnorm(trainn * p), trainn, p)
y_train <- X_train %*% b + rnorm(trainn)
X_test <- matrix(rnorm(testn * p), testn, p)
y_test <- X_test %*% b + rnorm(testn)

ridge_model <- glmnet(X_train, y_train, alpha = 0, lambda = lambda_vals)
testing_error <- numeric(length(lambda_vals))

for (i in seq_along(lambda_vals)) {
  y_pred <- predict(ridge_model, s = lambda_vals[i], newx = X_test)
  testing_error[i] <- mean((y_test - y_pred)^2)
}

plot(log(lambda_vals), testing_error, type="l", xlab=expression(lambda), ylab="Testing Error")
```



According to the plot, the left side with a small  $\lambda$  displays a small bias and a large variance, while the right side with a large  $\lambda$  displays a large bias and a small variance. Therefore, it demonstrates the bias-variance trade-off because the variance decreases and the bias increases as the penalty term,  $\lambda$ , increases.

## 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**  $y$ . Be careful about the orientation 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?

```

# Load the dataset
library(multtest)

##      BiocGenerics

##
##      '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, sort,
##      table, tapply, union, unique, unsplit, which.max, which.min

##      Biobase

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

data(golub)

# Perform marginal association test for each gene
test <- mt.teststat(golub, golub.cl, test="t.equalvar")

# Sort the genes by their p-values and select the top 100 genes
p_values <- 2 * pt(-abs(test), df = length(golub.cl) - 2)
df_pvalues <- data.frame(Gene = 1:nrow(golub), PValue = p_values)
sorted <- df_pvalues[order(df_pvalues$PValue), ]
top100 <- sorted$Gene[1:100]

# Construct a dataset with the top 10 genes and another one (call it X) with the remaining genes
top10 <- t(golub[top100[1:10], ])
X <- t(golub[top100[11:100], ])

# Perform principal component analysis (PCA)
top100_t <- t(golub[top100, ])
PCA <- prcomp(top100_t, scale. = TRUE)
y <- PCA$x[, 1]

```

```
# Perform ridge regression with 19-fold cross-validation
set.seed(432)
ridge <- cv.glmnet(X, y, alpha = 0, nfolds = 19)
best_lambda <- ridge$lambda.min
ridge_model <- glmnet(X, y, alpha = 0, lambda = best_lambda)
predictions <- predict(ridge_model, X)
SSR <- sum((y - predictions)^2)
SST <- sum((y - mean(y))^2)
R_squared <- 1 - SSR/SST
cat("R-squared under 19-fold cross-validation:", R_squared, "\n")
```

```
## R-squared under 19-fold cross-validation: 0.9772938
```

```
# Fit ridge regression but use GCV as the criterion
set.seed(432)
ridge_gcv <- cv.glmnet(X, y, alpha = 0, nfolds = nrow(X))
best_lambda_gcv <- ridge_gcv$lambda.min
ridge_model_gcv <- glmnet(X, y, alpha = 0, lambda = best_lambda_gcv)
predictions_gcv <- predict(ridge_model_gcv, X)
SSR_gcv <- sum((y - predictions_gcv)^2)
SST_gcv <- sum((y - mean(y))^2)
R_squared_gcv <- 1 - SSR_gcv/SST_gcv
cat("R-squared under GCV:", R_squared_gcv)
```

```
## R-squared under GCV: 0.9772938
```

Both the model with 19-fold cross-validation and the model with GCV fit well, as they have a R-Squared value of 0.977, which is very close to 1, meaning that my models are fitted pretty well.

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

**Answer:** I think there is not a clear bias-variance trade-off based on my results. Given the high R-squared values, it seems that cross-validation is able to help my model to balance the variance and bias.

### 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  $\beta_1$  and  $\beta_2$  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  $j$ th variable of the  $i$ th 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.**

**Answer:**

Expanding the objective function, we can get:

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

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

Based on the formula above,  $a = \frac{1}{n} \sum_{i=1}^n x_{i1}^2$ ,  $b = \frac{1}{n} \sum_{i=1}^n x_{i2}^2$ ,  $c = \frac{2}{n} \sum_{i=1}^n x_{i1}x_{i2}$ ,  $d = -\frac{2}{n} \sum_{i=1}^n y_i x_{i1}$ ,  $e = -\frac{2}{n} \sum_{i=1}^n y_i x_{i2}$ , and  $f = \frac{1}{n} \sum_{i=1}^n y_i^2$ .

```
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)
```

```
cat("a = ", a, "\n")
```

```
## a = 0.9241812
```

```
cat("b = ", b, "\n")
```

```
## b = 0.8625308
```

```
cat("c = ", c, "\n")
```

```
## c = -0.2694035
```

```
cat("d = ", d, "\n")
```

```
## d = -1.122192
```

```
cat("e = ", e, "\n")
```

```
## e = -0.5161552
```

```
cat("f = ", f, "\n")
```

```
## f = 1.25586
```

- b. [10 points] A coordinate descent algorithm essentially does two steps: i. Update  $\beta_1$  to its optimal value while keeping  $\beta_2$  fixed ii. Update  $\beta_2$  to its optimal value while keeping  $\beta_1$  fixed

Write down the updating rules for  $\beta_1$  and  $\beta_2$  using the coordinate descent algorithm. Use those previously defined coefficients in your formula 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  $\beta_1$  and  $\beta_2$ . Validate your solution using the `lm()` function.

**Answer:**

Updating rules for  $\beta_1$  (Set the derivative of  $f(x, y)$  with respect to  $\beta_1$  to 0 and then solve for  $\beta_1$ ):

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

Updating rules for  $\beta_2$  (Set the derivative of  $f(x, y)$  with respect to  $\beta_2$  to 0 and then solve for  $\beta_2$ ):

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

```
set.seed(432)
beta1 <- 0
beta2 <- 0

for (iter in 1:100) {
  beta1_prev <- beta1
  beta2_prev <- beta2
  beta1 <- sum(x[, 1] * (y - beta2 * x[, 2])) / sum(x[, 1]^2)
  beta2 <- sum(x[, 2] * (y - beta1 * x[, 1])) / sum(x[, 2]^2)

  if (abs(beta1 - beta1_prev) < 1e-6 && abs(beta2 - beta2_prev) < 1e-6) {
    break
  }
}

cat("beta1 =", beta1, ", beta2 =", beta2, "\n")
```

```
## beta1 = 0.6658955 , beta2 = 0.4032028
```

```
lm_model <- lm(y ~ x[, 1] + x[, 2] - 1)
summary(lm_model)
```

```
##
```

```
## Call:
```



```

## lm(formula = y ~ x[, 1] + x[, 2] - 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|)
## x[, 1]  0.66590    0.09377   7.102 1.98e-10 ***
## x[, 2]  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

```