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:

```
# 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)</pre>
```

```
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)

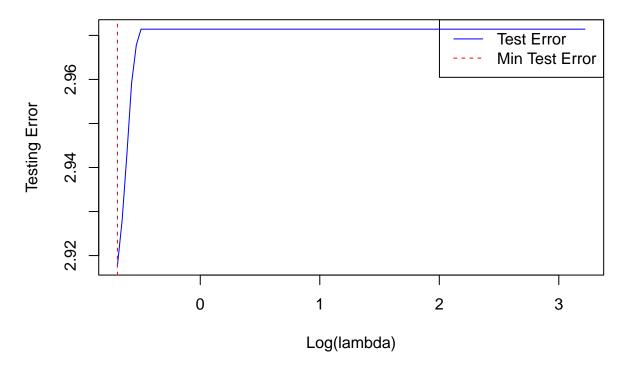
cat("MSE of Ordinary Linear Regression:", lm_mse, "\n")
cat("MSE of Ridge Regression:", ridge_mse, "\n")</pre>
```

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 The first term is correct, ridge does have a variance-bias trade off. The code shows MSE, it is var+bias, so it does not completely show the first term. We can calculate the variance and bias separately, also we need many different values of lambda to show the trade offs. The second term is also correct, but the code does not show this. We can perform many simulations to see how linear model would produce different coefficient on predictors with multicollinearity and how ridge regression will produce a relatively stable result. The third thing is not true, both regression will include all predictors. We can show it by directly printing out what predictor is included in the outcome model.
- 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 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?

```
# Load the glmnet package
library(glmnet)
## Loading required package: Matrix
## Loaded glmnet 4.1-8
# Set seed for reproducibility
set.seed(123)
# Parameters
trainn <- 50
testn <- 200
p <- 200
b \leftarrow rep(0.1, p)
# Generate Training Data
X_train <- matrix(rnorm(trainn * p), trainn, p)</pre>
y_train <- X_train %*% b + rnorm(trainn)</pre>
# Generate Testing Data
X_test <- matrix(rnorm(testn * p), testn, p)</pre>
y_test <- X_test %*% b + rnorm(testn)</pre>
# Define lambda values
lambda_values <- exp(seq(log(0.5), log(0.01), length.out = 100)) * trainn
# Fit the model using qlmnet
fit <- glmnet(X_train, y_train, alpha = 1, lambda = lambda_values)</pre>
# Predict on testing data for each lambda and calculate test error
predictions <- predict(fit, newx = X_test)</pre>
test_errors <- apply(predictions, 2, function(pred) mean((pred - y_test)^2))</pre>
# Plot the testing error against log(lambda)
plot(log(lambda_values), test_errors, type = "1", col = "blue",
     xlab = "Log(lambda)", ylab = "Testing Error",
     main = "Testing Error vs Regularization Parameter (lambda)")
# Highlight points of high bias and high variance
abline(v = log(lambda_values[which.min(test_errors)]), col = "red", lty = 2)
legend("topright", legend = c("Test Error", "Min Test Error"),
       col = c("blue", "red"), lty = c(1, 2))
```

Testing Error vs Regularization Parameter (lambda)



The plot demonstrates the bias-variance tradeoff in regularized regression using different values of λ . On the left side, where λ is small, the model has low bias but high variance. As λ increases, the model becomes more regularized, reducing variance but increasing bias, causes the testing error to rise. The red dashed line indicates the optimal λ , where the model achieves the best balance between bias and variance, minimizing the testing error. This reflects how increasing λ shifts the model from low-bias/high-variance to high-bias/low-variance. ## Question 2: Modeling High-Dimensional Data

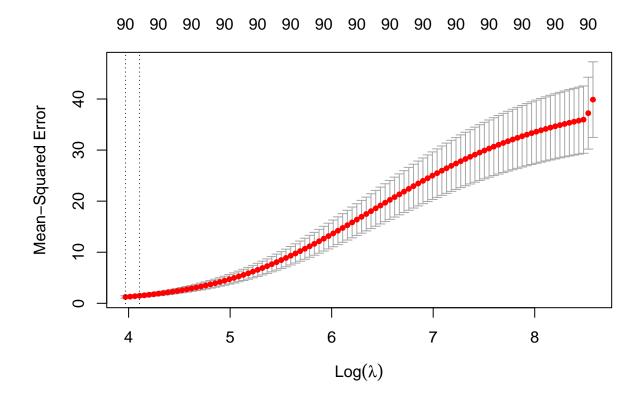
We will use the golub dataset from the multtest package. This dataset contains 3051 genes from 38 tumor mRNA samples from the leukemia microarray study Golub et al. (1999). This package is not included in R, but on bioconductor. Install the latest version of this package from bioconductor, and read the 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?
- b. [5 points] Based on your results, do you observe any bias-variance trade-off? If not, can you explain why?

```
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, sort,
##
       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")'.
##
library(BiocManager)
## Warning: package 'BiocManager' was built under R version 4.3.3
## Bioconductor version '3.18' is out-of-date; the current release version '3.19'
     is available with R version '4.4'; see https://bioconductor.org/install
library(glmnet)
data(golub)
test_stat <- mt.teststat(golub, golub.cl, test = "t.equalvar")</pre>
p_values <- 2 * pt(-abs(test_stat), df = length(golub.cl) - 2)</pre>
# Sort the p-values and select the top 100 genes
top_100_genes <- order(p_values)[1:100]</pre>
golub_top_100 <- golub[top_100_genes, ]</pre>
# Top 10 genes dataset
golub_top_10 <- golub[top_100_genes[1:10], ]</pre>
# Remaining 90 genes (X)
X <- golub[top_100_genes[11:100], ]</pre>
# Perform PCA
pca_result <- prcomp(t(golub_top_100))</pre>
# Extract the first principal component
y <- pca_result$x[, 1]
# Perform ridge regression with 19-fold cross-validation
cv_ridge <- cv.glmnet(t(X), y, alpha = 0, nfolds = 19)</pre>
```

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

```
# Plot the cross-validation curve
plot(cv_ridge)
```



```
print(cv_ridge$lambda.min)
```

[1] 52.9102

```
# Best lambda value
lambda_best <- cv_ridge$lambda.min
# Fit the ridge regression model using the best lambda
ridge_model <- glmnet(t(X), y, alpha = 0, lambda = lambda_best)
predicted_values <- predict(ridge_model, newx = t(X)) #modify the order
# Model fitting results
coef(ridge_model)</pre>
```

```
## V4
                 0.12830938
## V5
                 0.06200483
## V6
                 0.11015831
## V7
                -0.10199285
## V8
                 0.11146828
## V9
                -0.10066013
## V10
                -0.10907034
## V11
                 0.12134289
## V12
                -0.09120350
## V13
                -0.16660167
## V14
                0.09363349
## V15
                -0.18889498
## V16
                 0.10619176
## V17
                 0.05524015
## V18
                 0.15869056
## V19
                 0.12485712
## V20
                -0.18886232
## V21
                 0.09180342
## V22
                -0.11010613
## V23
                -0.06895677
## V24
                -0.13229057
## V25
                 0.08877202
## V26
                 0.11788656
## V27
                 0.07210223
## V28
                 0.09885771
## V29
                 0.08951376
## V30
                -0.07806448
## V31
                 0.14709023
## V32
                 0.11445514
## V33
                -0.15537027
## V34
                 0.12577899
## V35
                -0.10806787
## V36
                -0.12116131
                 0.11972262
## V37
## V38
                 0.11622623
## V39
                 0.08274805
## V40
                 0.09181162
## V41
                 0.15766888
## V42
                 0.12851343
## V43
                 0.06985951
## V44
                 0.13073708
## V45
                -0.12052193
## V46
                 0.10085328
## V47
                 0.09006716
## V48
                 0.16864868
## V49
                -0.14286575
## V50
                -0.13838831
## V51
                 0.12751175
## V52
                -0.07307115
## V53
                -0.11285129
## V54
                -0.16241507
## V55
                0.12575561
## V56
                -0.12320830
## V57
                -0.12537639
```

```
## V58
               -0.12581756
## V59
               -0.12031018
## V60
               0.09496847
## V61
               -0.11214930
## V62
                0.10473258
## V63
                0.05537495
## V64
                0.11800545
## V65
                0.12706479
## V66
                0.09097733
## V67
                0.10573702
## V68
               -0.09866149
## V69
               -0.05224602
## V70
               -0.13100117
## V71
               -0.06887120
## V72
               -0.10042435
## V73
               -0.07769141
## V74
               0.09213623
## V75
               -0.11222883
## V76
               0.14205207
## V77
               -0.14307375
## V78
               -0.08968158
## V79
               -0.09136936
## V80
               -0.11947478
## V81
               -0.11073302
## V82
               0.08893704
## V83
               0.12708744
## V84
               -0.07397766
## V85
               -0.18546572
## V86
               0.10097108
## V87
               -0.12942234
## V88
               -0.15817376
## V89
               -0.22404892
## V90
               -0.11578620
cv_mse <- min(cv_ridge$cvm)</pre>
cv_mse
## [1] 1.238109
var(y)
## [1] 39.53252
# Ridge regression using GCV
ridge_gcv <- glmnet(t(X), y, alpha = 0, lambda = 0)</pre>
# Check model coefficients for GCV-based fit
gcv_mse <- min(ridge_gcv$cvm)</pre>
```

Warning in min(ridge_gcv\$cvm): no non-missing arguments to min; returning Inf

```
print(gcv_mse)
## [1] Inf
ridge_gcv <- glmnet(t(X), y, alpha = 0)</pre>
gcv_score <- cv.glmnet(t(X), y, alpha = 0, type.measure = "mse", nfolds = 10)$cvm</pre>
print(gcv_score)
##
     [1] 40.465688 38.966659 37.306892 37.104164 36.893546 36.674859 36.447882
##
     [8] 36.212399 35.968195 35.715062 35.452793 35.181191 34.900064 34.609229
    [15] 34.308516 33.997762 33.676820 33.345557 33.003854 32.651613 32.288753
##
    [22] 31.915215 31.530963 31.135984 30.730294 30.313934 29.886977 29.449527
##
    [29] 29.001719 28.543724 28.075747 27.598031 27.110856 26.614540 26.109442
##
    [36] 25.595957 25.074522 24.545613 24.009743 23.467464 22.919365 22.366070
##
    [43] 21.808236 21.246551 20.681734 20.114526 19.545695 18.976026 18.406320
##
    [50] 17.837300 17.269369 16.703786 16.141310 15.581833 15.029462 14.481378
    [57] 13.940519 13.407785 12.883080 12.367098 11.860497 11.363891 10.877852
##
##
    [64] 10.402904
                    9.939523
                               9.488133
                                         9.049108
                                                   8.622611
                                                              8.208298
                                                                        7.806480
##
          7.416909
                    7.043638
                               6.683987
                                         6.337959
                                                    6.005520
                                                              5.686591
                                                                         5.381048
    [71]
##
    [78]
          5.088726
                    4.809421
                               4.542896
                                         4.288880
                                                    4.047079
                                                              3.817172
                                                                         3.598822
                                         2.833774
                                                    2.667740
##
    [85]
          3.391674
                    3.195365
                               3.009524
                                                              2.511044
                                                                         2.363313
```

the MSE is much smaller than the variance of y, indicating that the model is explaining a good portion of the variability. The lambda has a clear minimum, which suggests that the model is stable and not overly sensitive to lambda. This can indicate a good fit. According to the plot, the error bars are relatively small and do not overlap with the values on the right side, indicating stability. All these indicates a good fit.

1.854632

1.746204

1.644557

1.549343

##

##

[92]

[99]

2.224173

1.461918

2.093256

1.396638

1.970197

For GCV as the criterion, it does not seems to be a good fit. Though the score is decreasing, but we failed to take a minimum value, which indicates no convergence.

Nothing directly shows the bias-variance trade-off. The current code only returns the optimal lambda and corresponding cross-validation error without explicitly breaking down how the trade-off is happening. It more shows how good my model fit is. ## 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)</pre>
```

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:

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

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

$$a = \frac{1}{n} \sum_{i=1}^{n} x_{i1}^2, \quad b = \frac{1}{n} \sum_{i=1}^{n} x_{i2}^2, \quad c = \frac{2}{n} \sum_{i=1}^{n} x_{i1} x_{i2}$$

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

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.

```
set.seed(432)
n <- 100
x <- matrix(rnorm(n*2), n, 2)
y <- 0.7 * x[, 1] + 0.5 * x[, 2] + rnorm(n)
a <- sum(x[, 1]^2) / n
b <- sum(x[, 2]^2) / n
c <- 2 * sum(x[, 1] * x[, 2]) / n
d <- -2 * sum(y * x[, 1]) / n
e <- -2 * sum(y * x[, 2]) / n
f <- sum(y^2) / n</pre>
# Output the results
list(a = a, b = b, c = c, d = d, e = e, f = f)
```

```
## $a
## [1] 0.9241812
##
## $b
## [1] 0.8625308
##
## $c
## [1] -0.2694035
##
## $d
## [1] -1.122192
##
## $e
## [1] -0.5161552
##
## $f
## $f
## [1] 1.25586
```

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

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

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

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.

```
set.seed(432)
n <- 100
x \leftarrow matrix(rnorm(n * 2), n, 2)
y \leftarrow 0.7 * x[, 1] + 0.5 * x[, 2] + rnorm(n)
# Calculate coefficients
a \leftarrow sum(x[, 1]^2) / n
b \leftarrow sum(x[, 2]^2) / n
c \leftarrow 2 * sum(x[, 1] * x[, 2]) / n
d \leftarrow -2 * sum(y * x[, 1]) / n
e \leftarrow -2 * sum(y * x[, 2]) / n
# Initial values
beta1 <- 0
beta2 <- 0
# Stopping criterion parameters
tolerance <- 1e-6
max_iter <- 100
# Coordinate descent algorithm
for (i in 1:max_iter) {
  beta1_old <- beta1
  beta2_old <- beta2
  # Update beta1 keeping beta2 fixed
  beta1 <- (-c * beta2 - d) / (2 * a)
  # Update beta2 keeping beta1 fixed
  beta2 <- (-c * beta1 - e) / (2 * b)
}
# Output the estimated values of beta1 and beta2
cat("Estimated beta1:", beta1, "\n")
## Estimated beta1: 0.6658955
cat("Estimated beta2:", beta2, "\n")
## Estimated beta2: 0.4032028
# Validate with lm()
model <- lm(y ~ x - 1) # Without intercept
summary(model)
```

```
## Call:
## lm(formula = y \sim x - 1)
##
## Residuals:
                1Q Median
       Min
                                 3Q
## -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
##
\mbox{\tt \#\#} 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
```