HW04_zilinw3

Zilin Wang (zilinw3)

2024-09-25

Contents

Question 1: Sparsity and Correlation

During our lecture, we considered a simulation model to analyze the variable selection property of Lasso. Now let's further investigate the prediction error of both Lasso and Ridge, and understand the bias-variance trade-off. Consider the linear model defined as:

$$Y = X^{\mathrm{T}}\beta + \epsilon$$

Where $\beta = (\beta_1, \beta_2, ..., \beta_{100})^T$ with $\beta_1 = \beta_{11} = \beta_{21} = \beta_{31} = 0.4$ and all other β parameters set to zero. The p-dimensional covariate X follows a multivariate Gaussian distribution:

$$\mathbf{X} \sim N(\mathbf{0}, \Sigma_{p\times p}).$$

In Σ, all diagonal elements are 1, and all off-diagonal elements are ρ.

a. [15 points] A single Simulation Run

- Generate 200 training and 500 testing samples independently based on the above model.
- Use $\rho = 0.1$.
- Fit Lasso using `cv.glmnet()` on the training data with 10-fold cross-validation. Use `lambda.1se` to select the optimal λ.
- Report:
  - Prediction error (MSE) on the test data.
  - Report whether the true model was selected (you may refer to HW3 for this property).

Answer:

```
library(MASS)
library(glmnet)
```

```
##      Matrix
```

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

```
set.seed(123)

# Parameters
n_train <- 200
n_test <- 500
p <- 100
rho <- 0.1

# Beta coefficients
beta <- rep(0, 100)
beta[c(1, 11, 21, 31)] <- 0.4

# Covariance matrix
Sigma <- matrix(rho, nrow = p, ncol = p)
diag(Sigma) <- 1

# Generate training data
X_train <- mvrnorm(n = n_train, mu = rep(0, p), Sigma = Sigma)
y_train <- X_train %*% beta + rnorm(n_train)

# Generate testing data
X_test <- mvrnorm(n = n_test, mu = rep(0, p), Sigma = Sigma)
y_test <- X_test %*% beta + rnorm(n_test)

# Fit Lasso model using glmnet
lasso_model <- cv.glmnet(X_train, y_train, alpha = 1, nfolds = 10)
best_lambda <- lasso_model$lambda.1se

# Predict and calculate MSE
predictions <- predict(lasso_model, s = best_lambda, newx = X_test)
mse <- mean((predictions - y_test) ^ 2)

# Check selected variables
coef_lasso <- predict(lasso_model, type = "coefficients", s = best_lambda)[1:p, , drop = FALSE]
selected_vars <- which(coef_lasso != 0)[-1]

# Results
cat("Prediction Error (MSE):", mse, "\n")
```

```
## Prediction Error (MSE): 1.071154
```

```
cat("Selected variables:", selected_vars, "\n")
```

```
## Selected variables: 2 5 12 22 32 45 62 83
```

```
cat("The true model was selected:", all(c(1, 11, 21, 31) %in% selected_vars), "\n")
```

```
## The true model was selected: FALSE
```

b. [15 points] Higher Correlation and Multiple Simulation Runs

- Write a code to compare the previous simulation with $\rho = 0.1, 0.3, 0.5, 0.7, 0.9$.
- Perform 100 simulation runs as in part a) and record the prediction error and the status of the variable selection for Lasso.
- Report the average prediction error and the proportion of runs where the correct model was selected for each value of ρ.
- Discuss the reasons behind any observed changes.

Answer:

```r
set.seed(123)

# Parameters
n_train <- 200
n_test <- 500
p <- 100
n_sim <- 100
rho_values <- c(0.1, 0.3, 0.5, 0.7, 0.9)

# Beta coefficients
beta <- rep(0, p)
beta[c(1, 11, 21, 31)] <- 0.4
true_model <- c(1, 11, 21, 31)

# Initialize storage for results
results <- data.frame(rho = numeric(), avg_mse = numeric(), correct_model_prop = numeric())

# Simulation function
is_correct_model <- function(selected_vars, true_model) {
  return(all(true_model %in% selected_vars))
}

for (rho in rho_values) {
  mse_vals <- numeric(n_sim)
  correct_model_count <- 0

  for (i in 1:n_sim) {
    # Generate covariance matrix
    Sigma <- matrix(rho, nrow = p, ncol = p)
    diag(Sigma) <- 1

    # Generate training data
    X_train <- mvrnorm(n = n_train, mu = rep(0, p), Sigma = Sigma)
    y_train <- X_train %*% beta + rnorm(n_train)

    # Generate testing data
    X_test <- mvrnorm(n = n_test, mu = rep(0, p), Sigma = Sigma)
    y_test <- X_test %*% beta + rnorm(n_test)

    # Fit Lasso model
    lasso_model <- cv.glmnet(X_train, y_train, alpha = 1, nfolds = 10)
```

```
    best_lambda <- lasso_model$lambda.1se

    # Predict and calculate MSE
    predictions <- predict(lasso_model, s = best_lambda, newx = X_test)
    mse_vals[i] <- mean((predictions - y_test) ^ 2)

    # Check selected variables
    coef_lasso <- predict(lasso_model, type = "coefficients", s = best_lambda)[1:p, , drop = FALSE]
    selected_vars <- which(coef_lasso != 0)[-1]

    # Check if the true model was selected
    if (is_correct_model(selected_vars, true_model)) {
      correct_model_count <- correct_model_count + 1
    }
  }

  # Store average MSE and proportion of correct model selections
  results <- rbind(results, data.frame(rho = rho, avg_mse = mean(mse_vals),
                                       correct_model_prop = correct_model_count / n_sim))
}

# Print results
print(results)
```

```
##   rho  avg_mse correct_model_prop
## 1 0.1 1.195208                  0
## 2 0.3 1.171225                  0
## 3 0.5 1.206304                  0
## 4 0.7 1.176233                  0
## 5 0.9 1.148667                  0
```

The results show how the average MSE and the proportion of correct model selection changes as ρ increases.

The proportion of correctly selected models is consistently zero across all ρ values. This indicates that Lasso failed to accurately select the correct model in every simulation across all specified correlation levels.

The average MSE varies slightly across different values of ρ. There isn't a consistent increase or decrease, but there is an inconsistently but generally trends first go upward then downward.

When ρ increases, the multicollinearity increases, MSE increases due to the difficulty in selecting variables.

However, the decrease in MSE at higher ρ values is opposite from what is stated above. This can be partially explained by how Lasso deals with correlated variables. As correlation increases, Lasso tends to shrink coefficients more aggressively. When predictors are highly correlated, Lasso may effectively "average" these predictors by distributing the importance across them or by picking one as a representative, thus leading to more stable predictions, despite higher bias. Also, increased correlation can stabilize the model predictions if Lasso consistently selects similar sets of predictors across different samples, even if those are not the correct ones initially specified. This could reduce variance in predictions, slightly lowering MSE.

c. [15 points] Ridge Regression

- Repeat task b) with the ridge regression. You do not need to record the variable selection status since ridge always select all variables.
- Report the average prediction error, do you see any difference between ridge and Lasso? Any performance differences within ridge regression as ρ changes?
- Discuss the reasons behind any observed changes.

Answer:

```r
set.seed(123)
# Ridge regression simulation for multiple rho values
results_ridge <- data.frame(rho = numeric(), avg_mse = numeric())

for (rho in rho_values) {
  mse_vals <- numeric(n_sim)

  for (i in 1:n_sim) {
    # Generate covariance matrix
    Sigma <- matrix(rho, nrow = p, ncol = p)
    diag(Sigma) <- 1

    # Generate training and testing data
    X_train <- mvrnorm(n = n_train, mu = rep(0, p), Sigma = Sigma)
    y_train <- X_train %*% beta + rnorm(n_train)
    X_test <- mvrnorm(n = n_test, mu = rep(0, p), Sigma = Sigma)
    y_test <- X_test %*% beta + rnorm(n_test)

    # Fit Ridge model
    ridge_model <- cv.glmnet(X_train, y_train, alpha = 0, nfolds = 10)
    best_lambda <- ridge_model$lambda.1se

    # Predict and calculate MSE
    predictions <- predict(ridge_model, s = best_lambda, newx = X_test)
    mse_vals[i] <- mean((predictions - y_test) ^ 2)
  }

  # Store average MSE
  results_ridge <- rbind(results_ridge, data.frame(rho = rho, avg_mse = mean(mse_vals)))
}

# Print Ridge results
print(results_ridge)
```

```
##   rho  avg_mse
## 1 0.1 1.474492
## 2 0.3 1.397598
## 3 0.5 1.360568
## 4 0.7 1.250803
## 5 0.9 1.151606
```

As ρ increases from 0.1 to 0.9, the average MSE decreases consistently.

Difference between ridge and Lasso:

Lasso: MSE inconsistent but generally first increases then decreases as ρ increases.

Ridge: There's a clear, consistent decrease in MSE as ρ increases.

The reasons why MSE changes:

Ridge regression can reduce variance at the cost of increasing bias. As ρ increases, predictors become more correlated, Ridge regression can effectively manage this by distributing the effect among the correlated

predictors, thus reducing the variance without a substantial increase in bias. Therefore, as ρ increases, the model's predictions become more stable due to the reduced variance, leading to the decrease in MSE. Also, Ridge regression shrinks coefficients towards zero but never exactly to zero. This shrinkage effect is beneficial in highly correlated scenarios because it prevents any one variable from having too much influence on the model due to multicollinearity. This balanced shrinkage across all variables could lead to more accurate predictions as the correlation increases.

Question 2: Shrinkage Methods and Testing Error

In this question, we will predict the number of applications received using the variables in the College dataset that can be found in `ISLR2` package. The output variable will be the number of applications (Apps) and the other variables are predictors. If you use Python, consider migrating the data to an excel file and read it in Python.

a. [10 pts] Use the code below to divide the data set into a training set (600 observations) and a test set (177 observations). Fit a linear model (with all the input variables) using least squares on the training set using `lm()`, and report the test error (i.e., testing MSE).

```
  library(ISLR2)
```

```
##
##     'ISLR2'
```

```
## The following object is masked from 'package:MASS':
##
##     Boston
```

```
  data(College)

  # generate the indices for the testing data
  set.seed(7)
  test_idx = sample(nrow(College), 177)
  train = College[-test_idx,]
  test = College[test_idx,]
```

Answer:

```
library(ISLR2)
data(College)

# generate the indices for the testing data
set.seed(7)
test_idx <- sample(nrow(College), 177)
train <- College[-test_idx,]
test <- College[test_idx,]

# Fit a linear model on the training set
lm_model <- lm(Apps ~ ., data = train)
```

```r
# Make predictions on the test set
predictions <- predict(lm_model, newdata = test)

# Calculate Mean Squared Error (MSE) on the test data
mse_test <- mean((predictions - test$Apps)^2)

# Results
cat("Test Error (MSE):", mse_test, "\n")
```

```
## Test Error (MSE): 961142.3
```

b. [10 pts] Compare Lasso and Ridge regression on this problem. Train the model using cross-validation on the training set. Report the test error for both Lasso and Ridge regression. Use `lambda.min` and `lambda.1se` to select the optimal λ for both methods.

Answer:

```r
library(glmnet)

set.seed(7)

# Prepare the data
x_train <- model.matrix(Apps ~ ., data = train)[,-1]
y_train <- train$Apps
x_test <- model.matrix(Apps ~ ., data = test)[,-1]
y_test <- test$Apps

# Fit Lasso and Ridge regression models using cross-validation
cv_lasso <- cv.glmnet(x_train, y_train, alpha = 1)
cv_ridge <- cv.glmnet(x_train, y_train, alpha = 0)

# Identify the best lambda(lambda.min) for each model
best_lambda_min_lasso <- cv_lasso$lambda.min
best_lambda_min_ridge <- cv_ridge$lambda.min

# Identify the best lambda(lambda.1se) for each model
best_lambda_1se_lasso <- cv_lasso$lambda.1se
best_lambda_1se_ridge <- cv_ridge$lambda.1se

# Make predictions and calculate MSE for Lasso
predictions_lasso_min <- predict(cv_lasso, s = best_lambda_min_lasso, newx = x_test)
mse_lasso_min <- mean((y_test - predictions_lasso_min)^2)
predictions_lasso_1se <- predict(cv_lasso, s = best_lambda_1se_lasso, newx = x_test)
mse_lasso_1se <- mean((y_test - predictions_lasso_1se)^2)

# Make predictions and calculate MSE for Ridge
predictions_ridge_min <- predict(cv_ridge, s = best_lambda_min_ridge, newx = x_test)
mse_ridge_min <- mean((y_test - predictions_ridge_min)^2)
predictions_ridge_1se <- predict(cv_ridge, s = best_lambda_1se_ridge, newx = x_test)
mse_ridge_1se <- mean((y_test - predictions_ridge_1se)^2)
```

```r
# Output the MSEs
cat("Lasso MSE with lambda.min:", mse_lasso_min, "\n")
```

```
## Lasso MSE with lambda.min: 946414.7
```

```r
cat("Lasso MSE with lambda.1se:", mse_lasso_1se, "\n")
```

```
## Lasso MSE with lambda.1se: 1076206
```

```r
cat("Ridge MSE with lambda.min:", mse_ridge_min, "\n")
```

```
## Ridge MSE with lambda.min: 875548.9
```

```r
cat("Ridge MSE with lambda.1se:", mse_ridge_1se, "\n")
```

```
## Ridge MSE with lambda.1se: 1466561
```

c. [20 pts] The `glmnet` package implemented a new feature called `relaxed` fits and the associated tuning parameter `gamma`. You can find some brief explaination of this feature at the documentation of this package. See

- CRAN Documentation
- glmnet Vignette

Read these documentations regarding the `gamma` parameter, and summarize the idea of this feature in terms of the loss function being used. You need to write it specifically in terms of the data vectors $\mathbf{y}$ and matrix $\mathbf{X}$ and define any notations you need. Only consider the Lasso penalty for this question.

After this, implement this feature and utilize the cross-validation to find the optimal λ and γ for the College dataset. Report the test error for the optimal model.

Answer:

The relaxed fit in Lasso regression adds flexibility to the model by introducing a new tuning parameter γ. This parameter determines how much the Lasso solution is "shrunk" towards a simpler model. Essentially, γ controls how much we relax the Lasso solution, allowing us to find a balance between a fully constrained model (Lasso) and an unconstrained model (OLS).

To explain the relaxed fit mathematically, let's first define the core components involved:

Let $y \in \mathbb{R}^n$ be a response vector, which represents the number of applications (Apps) in this case.

Let $X \in \mathbb{R}^{n \times p}$ be a matrix of predictors, which contains the other variables in the dataset.

Lasso regression minimizes the following loss function:

$$\hat{\beta}_\lambda = \arg\min_\beta \left\{ \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\}$$

where λ is a tuning parameter that controls the strength of the Lasso penalty (the ℓ_1-norm of β).

The Lasso penalty encourages sparsity by shrinking some coefficients toward zero, potentially leading to simpler models. However, it may over regularize the model by letting too many coefficients to zero, even when their inclusion could improve model performance.

The relaxed fit introduces a new tuning parameter γ, which allows a mixture between the Lasso solution and a refitting of the coefficients without penalization. Specifically, after determining which coefficients are non-zero in the Lasso model (selected variables), the relaxed fit finds the optimal value of those coefficients with reduced penalization. Mathematically, it does the following:

First, fit the Lasso model with a regularization parameter λ, which provides an initial estimate $\hat{\beta}_\lambda$.

Then, relax the Lasso solution by blending it with an ordinary least squares (OLS) fit on the selected variables. The relaxed fit solves:

$$\hat{\beta}_{\lambda,\gamma} = \gamma \cdot \hat{\beta}_\lambda + (1-\gamma) \cdot \hat{\beta}_{OLS}$$

where: $\hat{\beta}_\lambda$ is the Lasso solution at a given λ.

$\hat{\beta}_{OLS}$ is the OLS estimate using the variables selected by Lasso.

$\gamma \in [0,1]$ is a tuning parameter that interpolates between the Lasso solution ($\gamma = 1$) and the OLS solution on the variables selected by Lasso ($\gamma = 0$).

When $\gamma = 1$, we recover the standard Lasso solution. When $\gamma = 0$, the solution corresponds to OLS on the subset of variables selected by Lasso.

The goal of the relaxed fit is to improve the predictive performance of the Lasso model by avoiding over-regularization. It allows the non-zero coefficients from the Lasso model to "relax" and be refitted with less or no penalization, which can lead to better performance, particularly when some of the predictor variables should not be fully shrunk to zero.

Summary

The loss function for the relaxed model is:

$$\text{minimize}_\beta \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \gamma\lambda \sum_{j \in S} |\beta_j| \right\}$$

where: y is the vector of response variables.

X is the matrix of predictors.

β are the coefficients.

λ is the regularization parameter that controls the amount of shrinkage.

n is the number of observations.

p is the number of predictors.

S is the set of predictors that were selected in the initial fit (i.e., those for which $\beta_j \neq 0$).

Here, γ scales the penalty term, with values ranging from 0 (no penalty on selected coefficients) to 1 (full initial penalty).

Next, implement this feature.

```
library(glmnet)

set.seed(7)

# Prepare the data
x_train <- model.matrix(Apps ~ ., data = train)[,-1]
y_train <- train$Apps
x_test <- model.matrix(Apps ~ ., data = test)[,-1]
```

```r
y_test <- test$Apps

# Fit Lasso with cross-validation including relaxed fits
cv_fit_relaxed <- cv.glmnet(x_train, y_train, alpha = 1, relax = TRUE)

# Identify the best lambda and gamma
best_lambda <- cv_fit_relaxed$lambda.min
best_gamma <- cv_fit_relaxed$glmnet.fit$gamma[which.min(cv_fit_relaxed$cvm)]

# Fit the optimal model
relaxed_model <- glmnet(x_train, y_train, alpha = 1, lambda = best_lambda, gamma = best_gamma)

# Make predictions
predictions_relaxed <- predict(relaxed_model, s = best_lambda, type = "response", newx = x_test)

# Calculate MSE for the relaxed model
mse_relaxed <- mean((y_test - predictions_relaxed)^2)

# Output the results
cat("Test Error (MSE) for Optimal Model:", mse_relaxed, "\n")
```

```
## Test Error (MSE) for Optimal Model: 946675.5
```

Question 3: Penalized Logistic Regression

In HW3, we used `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). The outcome `golub.cl` is an indicator for two leukemia types: Acute Lymphoblastic Leukemia (ALL) or Acute Myeloid Leukemia (AML). In genetic analysis, many gene expressions are highly correlated. Hence we could consider the Elastic net model for both sparsity and correlation.

```r
  if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
  BiocManager::install("multtest")
```

[15 pts] Fit logistic regression to this dataset. Use a grid of α values in $[0, 1]$ and report the best α and λ values using cross-validation.

Answer:

```r
library(glmnet)
library(multtest)
data(golub)

# Check the structure of the data
str(golub)
```

```
##  num [1:3051, 1:38] -1.458 -0.752 0.457 3.135 2.766 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : NULL
```

```r
str(golub.cl)
```

```
##  num [1:38] 0 0 0 0 0 0 0 0 0 0 ...
```

```r
x <- as.matrix(golub)
y <- factor(golub.cl, levels = c(0, 1))

# Define alpha
alpha_values <- seq(0, 1, by = 0.1)

# Initialize a list to store results
cv_results <- list()

# Perform cross-validation for each alpha value
for (alpha in alpha_values) {
  set.seed(42)
  cv_fit <- cv.glmnet(t(x), y, family = "binomial", alpha = alpha)
  cv_results[[paste("alpha", alpha, sep = "_")]] <- cv_fit
}

# Find the best alpha and corresponding lambda
best_cvm <- Inf
best_alpha <- NULL
best_lambda <- NULL

for (i in names(cv_results)) {
  cvm <- cv_results[[i]]$cvm
  min_cvm_index <- which.min(cvm)
  min_cvm <- cvm[min_cvm_index]
  if (min_cvm < best_cvm) {
    best_cvm <- min_cvm
    best_alpha <- as.numeric(sub("alpha_", "", i))
    best_lambda <- cv_results[[i]]$lambda[min_cvm_index]
  }
}

# Results
cat("Best Alpha:", best_alpha, "\n")
```

```
## Best Alpha: 0.3
```

```r
cat("Best Lambda:", best_lambda, "\n")
```

```
## Best Lambda: 0.01304836
```