Stat 432 Homework 2

Assigned: Sep 2, 2024; Due: 11:59 PM CT, Sep 12, 2024

Contents

Question 1 (Continuing the Simulation Study) 1
Question 2 (Training and Testing of Linear Regression) 5
Question 3 (Optimization) 8

Question 1 (Continuing the Simulation Study)

During our lecture, we considered a simulation study using the following data generator:

$$Y = \sum_{j=1}^{p} X_j 0.4^{\sqrt{j}} + \epsilon$$

And we added covariates one by one (in their numerical order, which is also the size of their effect) to observe the change of training error and testing error. However, in practice, we would not know the order of the variables. Hence several model selection tools were introduced. In this question, we will use similar data generators, with several nonzero effects, but use different model selection tools to find the best model. The goal is to understand the performance of model selection tools under various scenarios. Let's first consider the following data generator:

$$Y = \frac{1}{2} \cdot X_1 + \frac{1}{4} \cdot X_2 + \frac{1}{8} \cdot X_3 + \frac{1}{16} \cdot X_4 + \epsilon$$

where $\epsilon \sim N(0,1)$ and $X_j \sim N(0,1)$ for $j = 1, \ldots, p$. Write your code the complete the following tasks:

a. [10 points] Generate one dataset, with sample size $n = 100$ and dimension $p = 20$ as our lecture note. Perform best subset selection (with the `leaps` package) and use the AIC criterion to select the best model. Report the best model and its prediction error. Does the approach **selects the correct model**, meaning that all the nonzero coefficient variables are selected and all the zero coefficient variables are removed? Which variable(s) was falsely selected and which variable(s) was falsely removed? **Do not consider the intercept term**, since they are always included in the model. Why do you think this happens?

```r
library(leaps)
set.seed(1)
n <- 100
p <- 20
X <- matrix(rnorm(n * p), nrow = n, ncol = p)
epsilon <- rnorm(n)
Y <- 1/2 * X[,1] + 1/4 * X[,2] + 1/8 * X[,3] + 1/16 * X[,4] + epsilon
```

```r
subset_selection <- regsubsets(Y ~ ., data = as.data.frame(X), nvmax = p)
model_summary <- summary(subset_selection)
AIC_values <- n * log(model_summary$rss / n) + 2 * (1:p + 1)
best_model_index <- which.min(AIC_values)
best_model <- model_summary$which[best_model_index,]



selected_variables <- names(best_model[best_model == TRUE])
selected_variables <- selected_variables[selected_variables != "(Intercept)"]
selected_variables
```

```
## [1] "V1"  "V2"  "V3"  "V8"  "V13"
```

```r
best_model_coef <- coef(subset_selection, id = best_model_index)

best_model <- as.formula(paste("Y ~", paste(selected_variables, collapse = " + ")))

best_model_fit <- lm(best_model, data = as.data.frame(X))

best_fitted_values <- predict(best_model_fit)

error <- (Y - best_fitted_values)^2

mean(error)
```

```
## [1] 0.9877632
```

The selection process does not give me the true model, it includes several 0 coefficient variables. V3 is falsely removed and v10 18 19 are falsely included. This is because of the noise or the correlation between variables.

b. [10 points] Repeat the previous step with 100 runs of simulation, similar to our lecture note. Report
   i. the proportion of times that this approach selects the correct model
   ii. the proportion of times that each variable was selected

```r
set.seed(1)
n <- 100
p <- 20
num_simulations <- 100

# Initialize counters
correct_model_count <- 0
variable_selection_count <- rep(0, p)

# True model: X1, X2, X3, X4 are the true nonzero variables
true_model <- rep(FALSE, p)
true_model[1:4] <- TRUE  # X1 to X4 are the true variables with nonzero coefficients

# Run the simulation 100 times
for (i in 1:num_simulations) {
  # Generate the data
  X <- matrix(rnorm(n * p), nrow = n, ncol = p)
  epsilon <- rnorm(n)
  Y <- 1/2 * X[,1] + 1/4 * X[,2] + 1/8 * X[,3] + 1/16 * X[,4] + epsilon

  # Perform best subset selection
  subset_selection <- regsubsets(Y ~ ., data = as.data.frame(X), nvmax = p)
  model_summary <- summary(subset_selection)

  # Calculate AIC for each model size
  AIC_values <- n * log(model_summary$rss / n) + 2 * (1:p + 1)
  best_model_index <- which.min(AIC_values)
  best_model <- model_summary$which[best_model_index,]
  # Remove the intercept term from consideration
  best_model <- best_model[-1]  # remove intercept entry

  # Check if the selected model is the correct model
  if (all(best_model == true_model)) {
    correct_model_count <- correct_model_count + 1
  }

  # Update the count of how often each variable is selected
  variable_selection_count <- variable_selection_count + as.numeric(best_model)
}

# Calculate proportions
proportion_correct_model <- correct_model_count / num_simulations
proportion_variable_selected <- variable_selection_count / num_simulations
variable_selection_count
```

```
##  [1] 100  84  49  32  22  26  17  15  15  21  20  17  23  20  16  16  19  22  18
## [20]  21
```

```
proportion_correct_model
```

```
## [1] 0.02
```

```
proportion_variable_selected
```

```
##  [1] 1.00 0.84 0.49 0.32 0.22 0.26 0.17 0.15 0.15 0.21 0.20 0.17 0.23 0.20 0.16
## [16] 0.16 0.19 0.22 0.18 0.21
```

c. [10 points] In the previous question, you should be able to observe that the proportion of times that this approach selects the correct model is relatively low. This could be due to many reasons. Can you suggest some situations (setting of the model) or approaches (your model fitting procedure) for which the chance will be much improved (consider using AI tools if needed)? Implement that idea and verify the new selection rate and compare with the previous result. Furthermore,

   i. Discuss each of the settings or appraoches you have altered and explain why it can improve the selection rate.
   ii. If you use AI tools, discuss your experience with it. Such as how to write the prompt and whether you had to further modeify the code.

```r
set.seed(1)
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```r
n <- 1000
p <- 20
true_vars <- c(1, 2, 3, 4)
var_counts <- matrix(0, ncol = p, nrow = 100)  # To count variable selection
correct_model <- 0  # To count how many times the correct model is selected

# Loop over 100 simulations
for (i in 1:100) {
  X <- matrix(rnorm(n * p), nrow = n, ncol = p)  # Design matrix (n x p)
  beta <- c(1/2, 1/4, 1/8, 1/16, rep(0, p - 4))  # True coefficients
  epsilon <- rnorm(n, mean = 0, sd = 0.5)  # Noise with lower standard deviation
  Y <- X %*% beta + epsilon  # Generate response variable

  # Fit Lasso model using cross-validation to select the best lambda
  lasso_fit <- glmnet(X, Y, alpha = 1)
  cv_lasso <- cv.glmnet(X, Y, alpha = 1)
  best_coefs <- coef(cv_lasso, s = "lambda.1se")

  # Identify which variables were selected (non-zero coefficients)
  selected_vars <- which(best_coefs[-1] != 0)

  # Update the counts for selected variables
  var_counts[i, selected_vars] <- 1

  # Check if the selected model matches the true model
  if (length(selected_vars) == length(true_vars) && all(sort(selected_vars) == true_vars)) {
    correct_model <- correct_model + 1
  }
}

# Proportion of times the correct model was selected
prop_correct_model <- correct_model / 100
prop_correct_model
```

```
## [1] 0.54
```

To improve the low selection rate of the correct model observed in best subset selection, we can consider regularization techniques like Lasso regression and increasing sample size. Such method address issues such as noise, multicollinearity, and model complexity, which can obscure the true relationships between predictors and the response variable. For example, Lasso adds a penalty to regression coefficients, encouraging sparsity and reducing the chances of selecting irrelevant variables. Implementing Lasso and comparing its performance with best subset selection can improve the selection rate by focusing on the most relevant predictors.

I copy and paste the question but chatgpt gives wrong answer, so i check the code line by line and modify. It has some problem in checking condition(whether the best model is the true model) or some detailed logic(calculating propotion).

Question 2 (Training and Testing of Linear Regression)

We have introduced the formula of a linear regression

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

Let's use the `realestate` data as an example. The data can be obtained from our course website. Here, $\mathbf{X}$ is the design matrix with 414 observations and 4 columns: a column of 1 as the intercept, and `age`, `distance` and `stores`. $\mathbf{y}$ is the outcome vector of `price`.

a. [10 points] Write an `R` code to properly define both $\mathbf{X}$ and $\mathbf{y}$, and then perform the linear regression using the above formula. You cannot use `lm()` for this step. Report your $\hat{\beta}$. After getting your answer, compare that with the fitted coefficients from the `lm()` function.

```r
realestate = read.csv("realestate.csv", row.names = 1)
y <- realestate$price


X <- cbind(1, realestate$age, realestate$distance, realestate$stores)


beta_hat_manual <- solve(t(X) %*% X) %*% t(X) %*% y
print(beta_hat_manual)
```

```
##             [,1]
## [1,] 42.97728621
## [2,] -0.25285583
## [3,] -0.00537913
## [4,]  1.29744248
```

```r
lm_model <- lm(price ~ age + distance + stores, data = realestate)
print(coef(lm_model))
```

```
## (Intercept)         age    distance      stores
## 42.97728621 -0.25285583 -0.00537913  1.29744248
```

b. [10 points] Split your data into two parts: a testing data that contains 100 observations, and the rest as training data. Use the following code to generate the ids for the testing data. Use your previous code to fit a linear regression model (predict `price` with `age`, `distance` and `stores`), and then calculate the prediction error on the testing data. Report your (mean) training error and testing (prediction) error:

$$\text{Training Error} = \frac{1}{n_{\text{train}}} \sum_{i \in \text{Train}} (y_i - \hat{y}_i)^2 \tag{1}$$

$$\text{Testing Error} = \frac{1}{n_{\text{test}}} \sum_{i \in \text{Test}} (y_i - \hat{y}_i)^2 \tag{2}$$

Here y_i is the original y value and $\hat{y}_i$ is the fitted (for training data) or predicted (for testing data) value. Which one do you expect to be larger, and why? After carrying out your analysis, does the result matches your expectation? If not, what could be the causes?

```r
  set.seed(432)
  test_idx = sample(nrow(realestate), 100)
  train_idx <- setdiff(1:nrow(realestate), test_idx)

X_train <- X[train_idx, ]
X_test <- X[test_idx, ]
y_train <- y[train_idx]
y_test <- y[test_idx]
beta_hat_manual <- solve(t(X_train) %*% X_train) %*% t(X_train) %*% y_train
print(beta_hat_manual)
```

```
##              [,1]
## [1,] 44.411881051
## [2,] -0.293472985
## [3,] -0.005840325
## [4,]  1.142227737
```

```r
# Alternatively, using the lm() function
lm_model <- lm(price ~ age + distance + stores, data = realestate[train_idx, ])
print(coef(lm_model))
```

```
##  (Intercept)          age     distance       stores
## 44.411881051 -0.293472985 -0.005840325  1.142227737
```

```r
y_train_hat_manual <- X_train %*% beta_hat_manual
y_test_hat_manual <- X_test %*% beta_hat_manual

# Predictions using the lm() model
y_train_hat_lm <- predict(lm_model, newdata = realestate[train_idx, ])
y_test_hat_lm <- predict(lm_model, newdata = realestate[test_idx, ])
training_error_manual <- mean((y_train - y_train_hat_manual)^2)
training_error_manual
```

```
## [1] 74.57346
```

```r
# Testing error (manual)
testing_error_manual <- mean((y_test - y_test_hat_manual)^2)
testing_error_manual
```

```
## [1] 119.4458
```

```r
# Training error using lm() model
training_error_lm <- mean((y_train - y_train_hat_lm)^2)
training_error_lm
```

```
## [1] 74.57346
```

```r
# Testing error using lm() model
testing_error_lm <- mean((y_test - y_test_hat_lm)^2)
testing_error_lm
```

```
## [1] 119.4458
```

c. [10 points] Alternatively, you can always use built-in functions to fit linear regression. Setup your code to perform a step-wise linear regression using the `step()` function (using all covariates). Choose one among the AIC/BIC/Cp criterion to select the best model. For the `step()` function, you can use any configuration you like, such as `direction` etc. You should still use the same training and testing ids defined previously. Report your best model, training error and testing error.

```r
test_idx <- sample(nrow(realestate), 100)
train_idx <- setdiff(1:nrow(realestate), test_idx)

# Create training and testing datasets
realestate_train <- realestate[train_idx, ]
realestate_test <- realestate[test_idx, ]

# Fit a full linear model with all covariates
full_model <- lm(price ~ age + distance + stores, data = realestate_train)

# Stepwise regression using AIC as the criterion (you can change this to BIC if needed)
stepwise_model <- step(full_model, direction = "both", trace = 1)  # You can change "both" to "forward"
```

```
## Start:  AIC=1375.86
## price ~ age + distance + stores
##
##            Df Sum of Sq   RSS    AIC
## <none>                  24481 1375.9
## - age       1    2082.8 26564 1399.5
## - stores    1    3228.4 27709 1412.8
## - distance  1    8266.2 32747 1465.2
```

```r
# View the summary of the best model selected
summary(stepwise_model)
```

```
##
## Call:
## lm(formula = price ~ age + distance + stores, data = realestate_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -37.247  -5.396  -1.270   4.580  33.276
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.0451832  1.5423555  27.260  < 2e-16 ***
## age         -0.2222340  0.0432734  -5.136 4.98e-07 ***
## distance    -0.0052271  0.0005109 -10.231  < 2e-16 ***
## stores       1.3578257  0.2123640   6.394 5.94e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.887 on 310 degrees of freedom
## Multiple R-squared:  0.5489, Adjusted R-squared:  0.5445
## F-statistic: 125.7 on 3 and 310 DF,  p-value: < 2.2e-16
```

```r
# Predictions on training data using the best model
y_train_hat_stepwise <- predict(stepwise_model, newdata = realestate_train)

# Predictions on testing data using the best model
y_test_hat_stepwise <- predict(stepwise_model, newdata = realestate_test)

# Calculate training and testing errors (Mean Squared Error)
training_error_stepwise <- mean((realestate_train$price - y_train_hat_stepwise)^2)
testing_error_stepwise <- mean((realestate_test$price - y_test_hat_stepwise)^2)
training_error_stepwise
```

```
## [1] 77.96462
```

```r
testing_error_stepwise
```

```
## [1] 106.7575
```

Question 3 (Optimization)

a) [5 Points] Consider minimizing the following univariate function:

$$f(x) = \exp(1.5 \times x) - 3 \times (x+6)^2 - 0.05 \times x^3$$

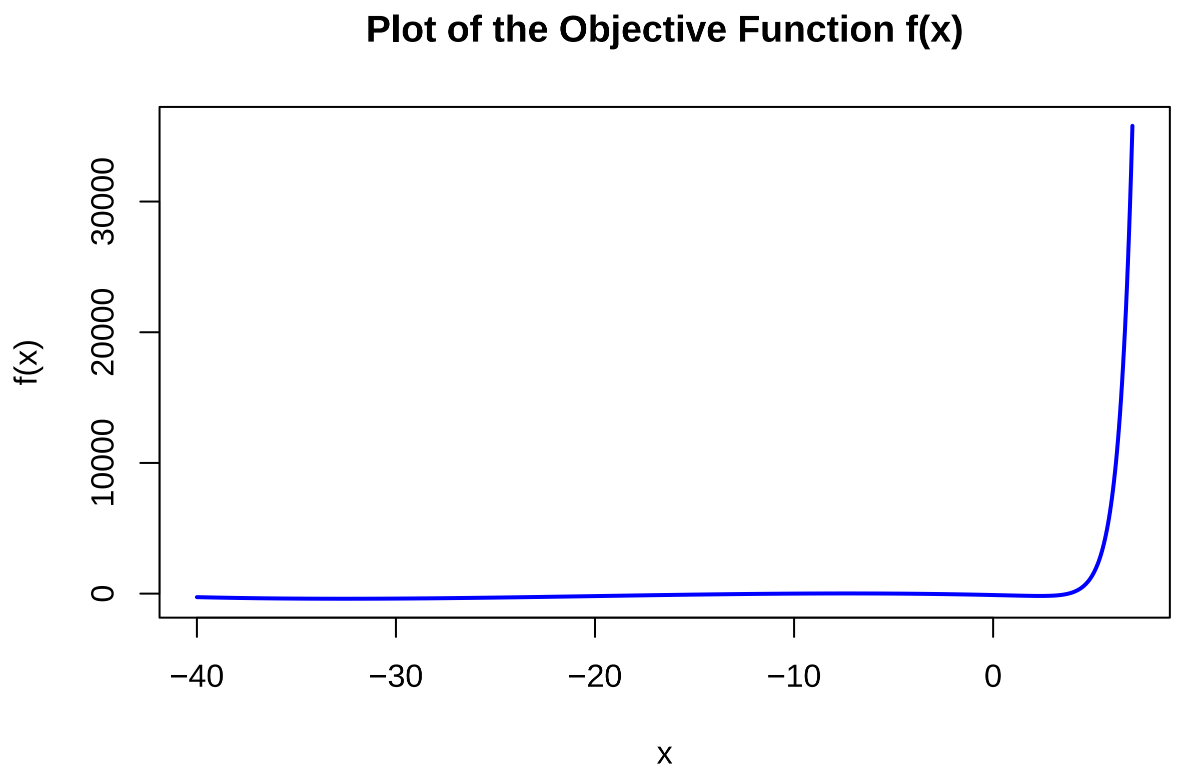
Write a function `f_obj(x)` that calculates this objective function. Plot this function on the domain $x \in [-40, 7]$.

```r
f_obj <- function(x) {
  exp(1.5 * x) - 3 * (x + 6)^2 - 0.05 * x^3
}

# Define the domain for x
x_vals <- seq(-40, 7, length.out = 1000)

# Calculate corresponding y values
y_vals <- f_obj(x_vals)

# Plot the function
plot(x_vals, y_vals, type = "l", col = "blue", lwd = 2,
     xlab = "x", ylab = "f(x)", main = "Plot of the Objective Function f(x)")
```



b) [10 Points] Use the `optim()` function to solve this optimization problem. Use `method = "BFGS"`. Try two initial points: -15 and 0. Report Are the solutions you obtained different? Why?

```
result_1 <- optim(-15, f_obj, method = "BFGS")
result_2 <- optim(0, f_obj, method = "BFGS")

print(result_1$par)
```

```
## [1] -32.64911
```

```
print(result_2$par)
```

```
## [1] 2.349967
```

They are different, The BFGS method is gradient-based, so it may get stuck in a local minimum depending on the starting point.

c) [10 Points] Consider a bi-variate function to minimize

$$f(x, y) = 3x^2 + 2y^2 - 4xy + 6x - 5y + 7$$

Derive the partial derivatives of this function with respect to x and y. And solve for the analytic solution of this function by applying the first-order conditions.

```r
library(Deriv)
f <- function(x, y) {
  3 * x^2 + 2 * y^2 - 4 * x * y + 6 * x - 5 * y + 7
}


df_dx <- Deriv(f, "x")
df_dy <- Deriv(f, "y")

system_of_eq <- function(vars) {
  x <- vars[1]
  y <- vars[2]
  c(df_dx(x, y), df_dy(x, y)) # Set both partial derivatives equal to zero
}


solution <- nleqslv::nleqslv(c(0, 0), system_of_eq)
print(solution$x)
```

```
## [1] -0.50  0.75
```

d) [10 Points] Check the second-order condition to verify that the solution you obtained in the previous step is indeed a minimum.

```r
df_dx <- Deriv(f, "x")
df_dy <- Deriv(f, "y")


d2f_dx2 <- Deriv(df_dx, "x")
d2f_dy2 <- Deriv(df_dy, "y")
d2f_dxdy <- Deriv(df_dx, "y")

x_critical <- -1/2
y_critical <- 3/4

hessian_matrix <- matrix(c(
  d2f_dx2(x_critical, y_critical), d2f_dxdy(x_critical, y_critical),
  d2f_dxdy(x_critical, y_critical), d2f_dy2(x_critical, y_critical)
), nrow = 2, byrow = TRUE)

cat("Hessian Matrix at the critical point (-1/2, 3/4):\n")
```

```
## Hessian Matrix at the critical point (-1/2, 3/4):
```

```r
print(hessian_matrix)
```

```
##      [,1] [,2]
## [1,]    6   -4
## [2,]   -4    4
```

```r
det_hessian <- det(hessian_matrix)
det_hessian > 0
```

```
## [1] TRUE
```

```r
d2f_dx2(x_critical, y_critical) > 0
```

```
## [1] TRUE
```

e) [5 Points] Use the `optim()` function to solve this optimization problem. Use `method = "BFGS"`. Set your own initial point. Report the solutions you obtained. Does different choices of the initial point lead to different solutions? Why?

```r
f_obj <- function(par) {
  x <- par[1]
  y <- par[2]
  3 * x^2 + 2 * y^2 - 4 * x * y + 6 * x - 5 * y + 7
}
initial_point_1 <- c(0, 0)
initial_point_2 <- c(-5, 5)
result_1 <- optim(initial_point_1, f_obj, method = "BFGS")
result_2 <- optim(initial_point_2, f_obj, method = "BFGS")
print(result_1$par)
```

```
## [1] -0.50  0.75
```

```r
print(result_2$par)
```

```
## [1] -0.50  0.75
```