Biostat 212A Homework 3

Due Feb 18, 2025 @ 11:59PM

Wenqiang Ge UID:106371961

2025-02-14

Table of contents

ISL Exercise 5.4.2 (10pts)

2. We will now derive the probability that a given observation is part of a bootstrap sample. Suppose that we obtain a bootstrap sample from a set of n observations.

   (a) What is the probability that the first bootstrap observation is *not* the jth observation from the original sample? Justify your answer.

   (b) What is the probability that the second bootstrap observation is *not* the jth observation from the original sample?

   (c) Argue that the probability that the jth observation is *not* in the bootstrap sample is $(1 - 1/n)^n$.

   (d) When $n = 5$, what is the probability that the jth observation is in the bootstrap sample?

   (e) When $n = 100$, what is the probability that the jth observation is in the bootstrap sample?

   (f) When $n = 10,000$, what is the probability that the jth observation is in the bootstrap sample?

   (g) Create a plot that displays, for each integer value of n from 1 to $100,000$, the probability that the jth observation is in the bootstrap sample. Comment on what you observe.

   (h) We will now investigate numerically the probability that a bootstrap sample of size $n = 100$ contains the jth observation. Here $j = 4$. We first create an array `store` with values that will subsequently be overwritten using the function `np.empty()`. We then

`np.empty()`

repeatedly create bootstrap samples, and each time we record whether or not the fifth observation is contained in the bootstrap sample.

```
rng = np.random.default_rng(10)
store = np.empty(10000)
for i in range(10000):
    store[i] = np.sum(rng.choice(100, replace=True) == 4)
          > 0
np.mean(store)
```

Comment on the results obtained.

Solution:

(a) The first bootstrap sample is selected randomly from the set of n observations, with replacement. The probability that the first sample is not the $j-th$ observation from the original sample is :

$P\ (first\ sample\ is\ not\ j\ )\ =\ \frac{n-1}{n}$

This is because there are $n-1$ other possible observations that could be selected out of the total n.

(b) The second bootstrap sample is selected randomly from the set of n observations, with replacement. The probability that the first sample is not the $j-th$ observation from the original sample is :

$P\ (second\ sample\ is\ not\ j\ )\ =\ \frac{n-1}{n}$

It has the same probability of the first sample is not the $j-th$ observation.

(c) Since each observation is selected independently with replacement, the probability that the $j-th$ observation is not selected in any specific sample is $(1-\frac{1}{n})$. Therefore, the probability that the $j-th$ observation is not in the bootstrap sample is:

$P\ (\ j-th\ observation\ is\ not\ in\ sample\ )\ =\ (1\ -\frac{1}{n})^n.$

(d)The probability that the $j-th$ observation is included in the bootstrap sample when $n = 5$:

$P\ (\ j-th\ observation\ is\ in\ sample\ )\ =\ 1-(\ 1-\frac{1}{5})^5\ \approx\ 0.672$

(e)The probability that the $j-th$ observation is included in the bootstrap sample when $n = 100$:

$P\ (\ j-th\ observation\ is\ in\ sample\ )\ =\ 1-(\ 1-\frac{1}{100})^{100}\ \approx\ 0.634$

(f)The probability that the $j-th$ observation is included in the bootstrap sample when $n = 10,000$:

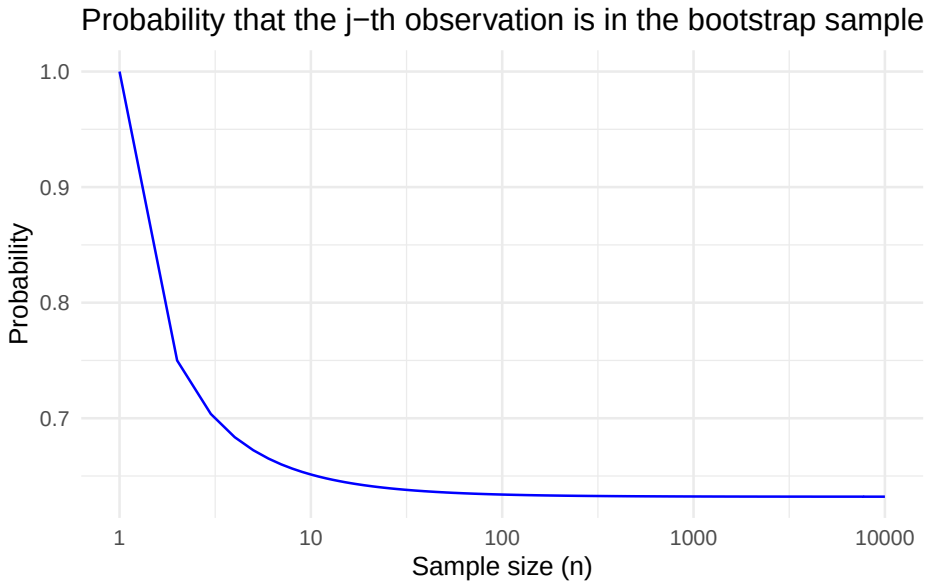
$$P\ (\ j-th\ observation\ is\ in\ sample\ )\ =\ 1-(\ 1-\tfrac{1}{10,000})^{10,000}\ \approx\ 0.632$$

(g)

```r
library(ggplot2)

n_values <- 1:10000
probabilities <- 1 - (1 - 1/n_values)^n_values
data <- data.frame(n = n_values, probability = probabilities)

ggplot(data, aes(x = n, y = probability)) +
  geom_line(color = "blue") +
  scale_x_log10() +
  labs(x = "Sample size (n)",
       y = "Probability",
       title = "Probability that the j-th observation is in the bootstrap sample") +
  theme_minimal()
```



For small sample sizes (n < 10), the probability is significantly lower. With very few observations, the chance of a specific observation being included in the bootstrap sample is quite low. As n increases, the probability gradually increases towards 1. The rate of this increase slows down as n gets larger, and for sufficiently large n, the probability converges near 1. The rapid increase at

the start might be due to the steep nature of the function for small n. Using a logarithmic scale for the x-axis can help visualize the change more effectively for smaller values of n.

(h)

```r
set.seed(10)
n <- 100
num_simulations <- 10000

store <- numeric(num_simulations)

for (i in 1:num_simulations) {
  sample <- sample(1:n, n, replace = TRUE)
  store[i] <- sum(sample == 4) > 0
}

probability <- mean(store)
print(probability)
```

```
[1] 0.6279
```

The simulation result gives a probability of approximately 0.6279. This means, based on the 10,000 simulations, the 5th observation is included in about 62.79% of the bootstrap samples.

This probability shows how often the 5th observation appears in the bootstrap samples.Since bootstrap sampling is done with replacement, each observation has a chance of being repeated in each sample, and hence the probability reflects how likely it is for a specific observation (in this case, the 5th) to appear in a given sample.

ISL Exercise 5.4.9 (20pts)

9. We will now consider the `Boston` housing data set, from the `ISLP` library.

   (a) Based on this data set, provide an estimate for the population mean of `medv`. Call this estimate $\hat{\mu}$.

(b) Provide an estimate of the standard error of $\hat{\mu}$. Interpret this result.

Hint: We can compute the standard error of the sample mean by dividing the sample standard deviation by the square root of the number of observations.

(c) Now estimate the standard error of $\hat{\mu}$ using the bootstrap. How does this compare to your answer from (b)?

(d) Based on your bootstrap estimate from (c), provide a 95 % confidence interval for the mean of `medv`. Compare it to the results obtained by using `Boston['medv'].std()` and the two standard error rule (3.9).

Hint: You can approximate a 95 % confidence interval using the formula $[\hat{\mu} - 2\text{SE}(\hat{\mu}), \hat{\mu} + 2\text{SE}(\hat{\mu})]$.

(e) Based on this data set, provide an estimate, $\hat{\mu}_{med}$, for the median value of `medv` in the population.

(f) We now would like to estimate the standard error of $\hat{\mu}_{med}$. Unfortunately, there is no simple formula for computing the standard error of the median. Instead, estimate the standard error of the median using the bootstrap. Comment on your findings.

(g) Based on this data set, provide an estimate for the tenth percentile of `medv` in Boston census tracts. Call this quantity $\hat{\mu}_{0.1}$. (You can use the `np.percentile()` function.)

`np.percentile()`

(h) Use the bootstrap to estimate the standard error of $\hat{\mu}_{0.1}$. Comment on your findings.

Solution:

(a)

```
library(ISLR)
library(MASS)

data("Boston")
```

```r
mu_hat <- mean(Boston$medv)
print(mu_hat)
```

```
[1] 22.53281
```

(b)

```r
# Calculate standard deviation of medv
s <- sd(Boston$medv)

# Calculate the number of observations
n <- length(Boston$medv)

# Calculate the standard error of the mean
SE_mu_hat <- s / sqrt(n)
print(SE_mu_hat)
```

```
[1] 0.4088611
```

If we were to draw repeated samples from the population, the sample means would vary by approximately 0.4089 units on average. A smaller standard error indicates a more precise estimate of the population mean.

(c)

```r
# Number of bootstrap samples
num_bootstrap <- 10000

# Set seed for reproducibility
set.seed(10)

# Create an empty vector to store the bootstrap sample means
bootstrap_means <- numeric(num_bootstrap)

# Perform bootstrap resampling
for (i in 1:num_bootstrap) {
  bootstrap_sample <- sample(Boston$medv, size = n, replace = TRUE)
  bootstrap_means[i] <- mean(bootstrap_sample)
}

# Estimate the standard error using the bootstrap
SE_bootstrap <- sd(bootstrap_means)
print(SE_bootstrap)
```

```
[1] 0.4070829
```

The bootstrap method provides a data-driven estimate of the standard error, rather than relying on the theoretical formula $\frac{s}{\sqrt{n}}$.

Both methods produce similar results, confirming that the standard error formula in (b) is accurate. The Bootstrap works well even when the underlying distribution of the data is unknown. Can be applied in cases where theoretical formulas are not available.

(d)

```
# Use the results from part (c) to calculate the 95% confidence interval

# Calculate the sample mean
mu_hat <- mean(Boston$medv)

# Standard error from the bootstrap samples
SE_bootstrap <- sd(bootstrap_means)

# Confidence interval using bootstrap
CI_lower_bootstrap <- mu_hat - 2 * SE_bootstrap
CI_upper_bootstrap <- mu_hat + 2 * SE_bootstrap

cat("95% Confidence Interval from Bootstrap: [", CI_lower_bootstrap, ", ",
    CI_upper_bootstrap, "]\n")
```

```
95% Confidence Interval from Bootstrap: [ 21.71864 ,  23.34697 ]
```

```
# Now, calculate the standard error using the sample's standard deviation
std_dev <- sd(Boston$medv)

# Confidence interval using the two standard error rule
CI_lower_sample <- mu_hat - 2 * SE_mu_hat
CI_upper_sample <- mu_hat + 2 * SE_mu_hat

cat("95% Confidence Interval from Sample Standard Error: [",
    CI_lower_sample, ", ", CI_upper_sample, "]\n")
```

```
95% Confidence Interval from Sample Standard Error: [ 21.71508 ,  23.35053 ]
```

Both methods yield very similar results,because tahe sample size is large enough for the standard error approximation to be accurate. The data distribution is approximately normal, making the standard error-based CI valid. The bootstrap method provides a more empirical estimate, without relying on normality assumptions. The standard error method is simpler and computationally efficient but assumes normality.

(e)

```r
# Estimate the median of medv
mu_med_hat <- median(Boston$medv)
print(mu_med_hat)
```

```
[1] 21.2
```

(f)

```r
# Create an empty vector to store the bootstrap sample medians
bootstrap_medians <- numeric(num_bootstrap)

# Perform bootstrap resampling for median
for (i in 1:num_bootstrap) {
  bootstrap_sample <- sample(Boston$medv, size = n, replace = TRUE)
  bootstrap_medians[i] <- median(bootstrap_sample)
}

# Estimate the standard error of the median
SE_med_bootstrap <- sd(bootstrap_medians)/ sqrt(num_bootstrap)
print(SE_med_bootstrap)
```

```
[1] 0.003792189
```

Unlike the mean, the standard error of the median cannot be computed using a simple formula. The bootstrap provides a data-driven method to estimate the uncertainty of the median. The result from bootstrap (0.0038) is small, suggesting a precise median estimate.

(g)

```r
# Estimate the 10th percentile of medv
mu_hat_0.1 <- quantile(Boston$medv, 0.1)
print(mu_hat_0.1)
```

```
  10%
12.75
```

(h)

```r
# Number of bootstrap samples
num_bootstrap <- 10000

# Create an empty vector to store the bootstrap sample 10th percentiles
bootstrap_mu_hat_0.1 <- numeric(num_bootstrap)

# Perform bootstrap resampling for 10th percentile
```

```r
for (i in 1:num_bootstrap) {
  bootstrap_sample <- sample(Boston$medv, size = length(Boston$medv),
                             replace = TRUE)
  bootstrap_mu_hat_0.1[i] <- quantile(bootstrap_sample, 0.1)
}

# Estimate the standard error of the 10th percentile
SE_mu_hat_0.1 <- sd(bootstrap_mu_hat_0.1)
print(SE_mu_hat_0.1)
```

```
[1] 0.5037677
```

The standard error of 0.5086 for the 10th percentile is a relatively moderate value, suggesting that there is a moderate amount of variability in the 10th percentile across the bootstrap samples. The smaller the standard error, the more consistent the 10th percentile is across different samples.

Least squares is MLE (10pts)

Show that in the case of linear model with Gaussian errors, maximum likelihood and least squares are the same thing, and C_p and AIC are equivalent.

Solution:

Least squares is MLE:

(1) Linear Model with Gaussian errors: $y = X\beta + \varepsilon$, where:

y: response vector X: design metrix β: coefficients vector

$\varepsilon \sim N(0, \sigma^2 I_n)$: error term

Log-likelihood function for y under the assumption of Gaussian error:

$$l(\beta, \sigma^2 | y) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y - X\beta)^T(y - X\beta)$$

To maximize the likelihood:

the first part involves σ^2, second parts related to the residual sum of squares, $(y - X\beta)^T(y - X\beta)$. So it is same with minimizing the sum of RSs.

Therefore, the maximum likelihood estimator (MLE) for β coincide with the least squares estimator (LSE) in the case of Gaussian error.

(2) Cp and AIC Equivalence

$C_p = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + 2p$, $AIC = -2l(\hat{\beta}) + 2p$, where $l(\hat{\beta})$ is the log-likelihood at maximum likelihood estimates of β.

For the linear model with Gaussian error:

$l(\hat{\beta}) = \frac{-n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y - X\hat{\beta})^T(y - X\hat{\beta})$, the $RSS = (y - X\hat{\beta})^T(y - X\hat{\beta})$

So $AIC = \frac{n}{\sigma^2}\left(\frac{1}{n}(y - X\hat{\beta})^T(y - X\hat{\beta})\right) + 2p$, which is equivalent to C_p.

Thus, in the case of Gaussian error, Cp and AIC are equivalents.

ISL Exercise 6.6.1 (10pts)

Conceptual

1. We perform best subset, forward stepwise, and backward stepwise selection on a single data set. For each approach, we obtain $p + 1$ models, containing $0, 1, 2, \ldots, p$ predictors. Explain your answers:

   (a) Which of the three models with k predictors has the smallest *training* RSS?

   (b) Which of the three models with k predictors has the smallest *test* RSS?

   (c) True or False:

   i. The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the $(k+1)$-variable model identified by forward stepwise selection.

   ii. The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the $(k + 1)$-variable model identified by backward stepwise selection.

   iii. The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the $(k + 1)$-variable model identified by forward stepwise selection.

   iv. The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the $(k+1)$-variable model identified by backward stepwise selection.

   v. The predictors in the k-variable model identified by best subset are a subset of the predictors in the $(k + 1)$-variable model identified by best subset selection.

Solution:

(a) **Best subset selection.**

Best subset selection considers all possible combinations of predictors. It will choose the set of k predictors that minimizes the training RSS, so it will likely have the smallest training RSS for any k predictors.

Forward stepwise selection starts with no predictors and adds the best predictors

one by one. While it tries to improve the fit with each step, it does not consider all possible combinations, so the training RSS might not be as small as the one obtained from best subset selection.

Backward stepwise regression starts with all predictors and removes the least important ones. While it considers all predictors initially, it may not reach the best combination for k predictors due to its greedy nature, so it might have a larger training RSS than best subset selection.

(b) **Forward stepwise selection.**

While best subset selection minimizes training RSS, it might overfit the training data because it explores all possible models. As a result, it might lead to a more complex model with higher variance and a larger test RSS.

Forward stepwise selection builds the model gradually by adding predictors, so it might be less likely to overfit compared to best subset selection. However, it still could lead to a relatively complex model, which might have a larger test RSS if it overfits.

Backward stepwise regression starts with all predictors and removes the least important ones, so it might end up with a simpler model that generalizes better than the other two. Since it begins with a full model and gradually reduces complexity, it may avoid overfitting and result in a smaller test RSS.

(c) **i: True**

In forward stepwise selection, the model starts with no predictors and progressively adds the best predictor at each step. Therefore, the $k - variable$ model will be a subset of the $(k+1) - variable$ model since at each step, the new model includes all the predictors of the previous model plus one more.

ii: False

The $k - variable$ model is not necessarily a subset of the $(k+1) - variable$ model, because the removal of predictors is based on their importance or performance in the model. For instance, the $k - variable$ model could have a different combination of predictors than the $(k + 1) - variable$ model, so it may not be a subset.

iii: False

The two methods are different in their approach, and the set of predictors selected by backward stepwise is not necessarily a subset of those selected by forward stepwise. The predictors chosen by each method may differ based on the initial conditions and the way each method evaluates predictor importance.

iv: False

Forward stepwise adds predictors gradually, while backward stepwise removes predictors from the full model. The $k - variable$ model selected by forward stepwise does not have to be a subset of the $(k + 1) - variable$ model selected

by backward stepwise, as the two methods may select different combinations of predictors.

v: True

The $k-variable$ model selected by best subset selection will always be a subset of the $(k+1)-variable$ model selected by best subset selection, as the $(k+1)-variablemodel$ is simply a larger subset of predictors. The $(k+1)-variablemodel$ is fitted based on previous model $(k-variable)$, so it must be the subset of previous model.

ISL Exercise 6.6.3 (10pts)

3. Suppose we estimate the regression coefficients in a linear regression model by minimizing

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p}|\beta_j|\leq s$$

for a particular value of s. For parts (a) through (e), indicate which of i. through v. is correct. Justify your answer.

(a) As we increase s from 0, the training RSS will:

i. Increase initially, and then eventually start decreasing in an inverted U shape.

ii. Decrease initially, and then eventually start increasing in a U shape.

iii. Steadily increase.

iv. Steadily decrease.

v. Remain constant.

(b) Repeat (a) for test RSS.

(c) Repeat (a) for variance.

(d) Repeat (a) for (squared) bias.

(e) Repeat (a) for the irreducible error.

Solution:

(a) **i: Increase initially, and then eventually start decreasing in an inverted U shape.**

When s is small (close to 0), the model will have fewer non-zero coefficients which leads to underfitting, and the training RSS will be relatively high.

As s increases, the model can accommodate more predictors, so it fits the data better, and the training RSS starts to decrease.

As s continues to increase, more coefficients are included, and the model may overfit the data. Eventually, the training RSS will start increasing again as the model becomes too complex.

(b) **ii. Decrease initially, and then eventually start increasing in a U shape.**

Training RSS follows an inverted U-shape, because increasing the number of predictors improves the fit to the training data initially, but eventually leads to overfitting.

Test RSS: The test RSS will likely follow a U-shape. Initially, as you increase s, the test RSS will decrease (since the model starts fitting the data better), but after a certain point, as the model overfits, the test RSS will start to increase.

(c) **ii. Decrease initially, and then eventually start increasing in a U shape.**

Variance refers to the variability of the model predictions across different datasets (e.g., in a cross-validation setting or in new datasets).

When $s = 0$, the model is overly constrained and underfits, leading to higher variance. As we increase s, the model becomes more flexible and is able to fit the data better, which initially reduces variance. However, if s increases too much and the model overfits, the variance will increase because the model becomes overly sensitive to the training data.

(d) **iii. Steadily decrease.**

Bias is the difference between the expected model prediction and the true value.

When $s = 0$, the model is highly constrained and underfits, which results in a high bias.

As s increases, the model becomes more flexible and starts fitting the data better, reducing the bias.

If s increases too much, the model starts to overfit, which means it fits the training data well but is less likely to generalize to new data, resulting in low bias.

Thus, squared bias will steadily decrease as s increases because the model is able to fit the true underlying relationship better.

(e) **v. Remain constant.**

Irreducible error is the noise in the data that cannot be explained by the model. It is constant and does not depend on the complexity of the model.

ISL Exercise 6.6.4 (10pts)

4. Suppose we estimate the regression coefficients in a linear regression model by minimizing

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

for a particular value of λ. For parts (a) through (e), indicate which of i. through v. is correct. Justify your answer.

(a) As we increase λ from 0, the training RSS will:
   i. Increase initially, and then eventually start decreasing in an inverted U shape.
   ii. Decrease initially, and then eventually start increasing in a U shape.
   iii. Steadily increase.
   iv. Steadily decrease.
   v. Remain constant.

(b) Repeat (a) for test RSS.

(c) Repeat (a) for variance.

(d) Repeat (a) for (squared) bias.

(e) Repeat (a) for the irreducible error.

Solution:

(a) **i: Increase initially, and then eventually start decreasing in an inverted U shape.**

When $\lambda = 0$, there is no penalty term, so the objective function is just RSS.

The model will likely overfit the data with more complex models, resulting in a very low training RSS.

As λ increases, the penalty on the coefficients increases, which forces the coefficients to shrink towards zero, making the model simpler. The model will not fit the training data as perfectly, so the training RSS will initially increase.

As λ continues to increase, the model will become increasingly constrained, and the coefficients will be forced to shrink more and more, making the model even simpler.

Eventually, the model will be too simple (potentially just a constant model), and the training RSS will start decreasing as it reduces the model complexity.

(b) **ii. Decrease initially, and then eventually start increasing in a U shape.**

When $\lambda = 0$, the model might overfit the data, and the test RSS will be relatively high.

As λ increases, the model starts to generalize better by reducing overfitting. This will reduce the test RSS initially.

As λ increases further, the model becomes too simple, and the test RSS may start increasing again, as it fails to capture the complexity of the data.

(c) **v. Steadily decrease.**

Variance refers to the variability of the model's predictions.

When $\lambda = 0$, the model may have high variance because it fits the training data too well, capturing noise as if it were signal.

As λ increases, the model is constrained, reducing its ability to fit the noise, and the variance of the predictions decreases.

As λ increases too much, the model becomes too simple (potentially underfitting), but the variance would remain low because it is no longer sensitive to fluctuations in the data.

(d) **iii. Steadily increase.**

When $\lambda = 0$, the model has no penalty, and it is free to fit the data exactly (which could result in low bias for the training data but high variance).

As λ increases, the model is forced to simplify, which increases bias because the model is less able to capture the true underlying relationship in the data.

As λ increases too much, the bias continues to increase because the model becomes too simple and may underfit the data.

(e) **v. Remain constant.**

The irreducible error is the noise in the data that cannot be modeled or explained by the predictors. This error is constant and does not depend on the model or the value of λ.

ISL Exercise 6.6.5 (10pts)

5. It is well-known that ridge regression tends to give similar coefficient values to correlated variables, whereas the lasso may give quite different coefficient values to correlated variables. We will now explore this property in a very simple setting.

   Suppose that $n = 2$, $p = 2$, $x_{11} = x_{12}$, $x_{21} = x_{22}$. Furthermore, suppose that $y_1 + y_2 = 0$ and $x_{11} + x_{21} = 0$ and $x_{12} + x_{22} = 0$, so that the estimate for the intercept in a least squares, ridge regression, or lasso model is zero: $\hat{\beta}_0 = 0$.

   (a) Write out the ridge regression optimization problem in this setting.

   (b) Argue that in this setting, the ridge coefficient estimates satisfy $\hat{\beta}_1 = \hat{\beta}_2$.

   (c) Write out the lasso optimization problem in this setting.

   (d) Argue that in this setting, the lasso coefficients $\hat{\beta}_1$ and $\hat{\beta}_2$ are not unique—in other words, there are many possible solutions to the optimization problem in (c). Describe these solutions.

Solution:

(a). Ridge regression:

To minimize: $\sum_{i=1}^{n} (y_i - x_{i1}\beta_1 - x_{i2}\beta_2)^2 + \lambda(\beta_1^2 + \beta_2^2)$

(b) since $x_{i1} = x_{i2}$, these two predictors are perfectly correlated, and the penalty equally affect both coefficients.
The optimization process can't distinguish between them, so $\hat{\beta}_1 = \hat{\beta}_2$.
During shrinking coefficients toward similar values, their values will converge to be the same, so $\hat{\beta}_1 = \hat{\beta}_2$.

(c). LASSO regression objective function:

To minimize: $\sum_{i=1}^{n} (y_i - x_{i1}\beta_1 - x_{i2}\beta_2)^2 + \lambda(|\beta_1| + |\beta_2|)$

(d). since L1 penalty encourage sparsity, so there's one case that $\hat{\beta}_1 = 0$ and $\hat{\beta}_2 \neq 0$. As a result, the model may choose either coefficient coefficient to be zero. For instance: $\beta_1 = 0$, $\beta_2 = a$; $\beta_1 = a$, $\beta_2 = 0$, or $|\hat{\beta}_1| + |\hat{\beta}_2| = a$.

ISL Exercise 6.6.11 (30pts)

11. We will now try to predict per capita crime rate in the `Boston` data set.

    (a) Try out some of the regression methods explored in this chapter, such as best subset selection, the lasso, ridge regression, and PCR. Present and discuss results for the approaches that you consider.

    (b) Propose a model (or set of models) that seem to perform well on this data set, and justify your answer. Make sure that you are evaluating model performance using validation set error, cross-validation, or some other reasonable alternative, as opposed to using training error.

    (c) Does your chosen model involve all of the features in the data set? Why or why not?

You must follow the typical machine learning paradigm to compare *at least* 3 methods: least squares, lasso, and ridge. Report final results as:

Solution:

Method	CV RMSE	Test RMSE
LS	5.227096	8.994735
Ridge	5.727724	9.00997
Lasso	5.728481	9.001981
PLS	5.179474	8.997268

(a)

```r
# Load necessary libraries
library(MASS)       # Contains the Boston dataset
library(leaps)      # For best subset selection
library(tibble)     # For organizing results
library(pls)         # PLS regression
library(glmnet)      # For Ridge and Lasso regression
library(caret)       # For cross-validation
library(ggplot2)     # For visualization

# Load Boston dataset
data("Boston")

# View summary of the dataset
summary(Boston$crim)  # Checking the crime rate statistics
```

```
   Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.00632  0.08204  0.25651  3.61352  3.67708 88.97620
```

```r
# Define the full model formula
full_formula <- crim ~ .
```

least squares

```r
# Define predictors (X) and target variable (y)
X <- Boston[, -which(names(Boston) == "crim")]  # Remove crim from predictors
y <- Boston$crim

# Split data into 80% training (non-test) and 20% test set
set.seed(123)  # For reproducibility
train_index <- createDataPartition(y, p = 0.8, list = FALSE)
train_data <- Boston[train_index, ]
test_data <- Boston[-train_index, ]

# Define 10-fold Cross-Validation
train_control <- trainControl(method = "cv", number = 10)

# Train Least Squares Regression (LS) model
ls_model <- train(crim ~ .,
                  data = train_data,
                  method = "lm",
                  trControl = train_control)
```

```r
# Get Cross-Validation RMSE
cv_rmse_ls <- ls_model$results$RMSE
cat("LS CV RMSE:", cv_rmse_ls, "\n")
```

```
LS CV RMSE: 5.227096
```

```r
# Fit the final LS model on the full training (non-test) data
final_ls_model <- lm(crim ~ ., data = train_data)
# Make predictions on the test set
predictions_ls <- predict(final_ls_model, newdata = test_data)

# Compute Test RMSE
test_rmse_ls <- sqrt(mean((test_data$crim - predictions_ls)^2))
cat("LS Test RMSE:", test_rmse_ls, "\n")
```

```
LS Test RMSE: 8.994735
```

CV RMSE is the average validation error from cross-validation and helps select the best model. CV RMSE (5.23) estimates the expected error on new (unseen) data if we re-train the model on different training samples. Test RMSE (8.99) is the actual error on the test set (20% unseen data). It is significantly higher than CV RMSE, which indicates overfitting.

Ridge regression:

```r
# Define predictors (X) and target variable (y)
X <- Boston[, -which(names(Boston) == "crim")]  # Remove crim from predictors
y <- Boston$crim

# Convert to matrix format for glmnet
X_matrix <- as.matrix(X)

# Split data into 80% training (non-test) and 20% test set
set.seed(123)  # For reproducibility
train_index <- createDataPartition(y, p = 0.8, list = FALSE)
train_X <- X_matrix[train_index, ]
test_X <- X_matrix[-train_index, ]
train_y <- y[train_index]
test_y <- y[-train_index]

# Standardize the predictors (Ridge requires standardization)
train_X_scaled <- scale(train_X)
test_X_scaled <- scale(test_X, center = attr(train_X_scaled, "scaled:center"),
                       scale = attr(train_X_scaled, "scaled:scale"))
```

```r
# Define lambda sequence (range of values for tuning)
lambda_seq <- 10^seq(4, -2, length = 100)

# Perform cross-validation to find the best lambda
set.seed(123)
cv_ridge <- cv.glmnet(train_X_scaled, train_y, alpha = 0, lambda = lambda_seq)

# Get the best lambda
best_lambda_ridge <- cv_ridge$lambda.min
cat("Best Lambda for Ridge:", best_lambda_ridge, "\n")
```

```
Best Lambda for Ridge: 0.1232847
```

```r
# Cross-validation RMSE
cv_rmse_ridge <- sqrt(min(cv_ridge$cvm))
cat("Ridge CV RMSE:", cv_rmse_ridge, "\n")
```

```
Ridge CV RMSE: 5.727724
```

```r
# Train Ridge model with best lambda
ridge_model <- glmnet(train_X_scaled, train_y, alpha = 0,
                      lambda = best_lambda_ridge)

# View the coefficients
coef(ridge_model)
```

```
14 x 1 sparse Matrix of class "dgCMatrix"
                     s0
(Intercept)  3.46735510
zn           0.77412706
indus       -0.42968556
chas        -0.09237806
nox         -1.14423499
rm          -0.07950546
age          0.23788138
dis         -1.64455413
rad          4.43848589
tax         -0.06271392
ptratio     -0.55801338
black       -0.70114986
lstat        0.88998195
medv        -1.19562757
```

```r
# Make predictions on test data
predictions_ridge <- predict(ridge_model, s = best_lambda_ridge,
                             newx = test_X_scaled)

# Compute Test RMSE
test_rmse_ridge <- sqrt(mean((test_y - predictions_ridge)^2))
cat("Ridge Test RMSE:", test_rmse_ridge, "\n")
```

```
Ridge Test RMSE: 9.00997
```

The results show that Ridge Regression did not outperform Least Squares (LS) Regression in this case. The CV RMSE for Ridge (5.727724) is higher than LS (5.227096), indicating that Ridge did not improve cross-validation performance. Additionally, the Test RMSE for Ridge (9.00997) is slightly worse than LS (8.994735), suggesting that Ridge did not significantly reduce overfitting. This could be due to low multicollinearity among predictors, a dataset that is not high-dimensional, or over-penalization from Ridge's regularization term.

LASSO:

```r
# Define predictors (X) and target variable (y)
X <- Boston[, -which(names(Boston) == "crim")]  # Remove crim from predictors
y <- Boston$crim

# Convert predictors to matrix format for glmnet
X_matrix <- as.matrix(X)

# Split data into 80% training (non-test) and 20% test set
set.seed(123)  # For reproducibility
train_index <- createDataPartition(y, p = 0.8, list = FALSE)
train_X <- X_matrix[train_index, ]
test_X <- X_matrix[-train_index, ]
train_y <- y[train_index]
test_y <- y[-train_index]

# Standardize the predictors (Lasso requires standardization)
train_X_scaled <- scale(train_X)
test_X_scaled <- scale(test_X, center = attr(train_X_scaled, "scaled:center"),
                       scale = attr(train_X_scaled, "scaled:scale"))
```

```r
# Define a sequence of lambda values for tuning
lambda_seq <- 10^seq(4, -2, length = 100)

# Perform cross-validation to find the best lambda
set.seed(123)
```

```r
cv_lasso <- cv.glmnet(train_X_scaled, train_y, alpha = 1, lambda = lambda_seq)

# Get the best lambda
best_lambda_lasso <- cv_lasso$lambda.min
cat("Best Lambda for Lasso:", best_lambda_lasso, "\n")
```

```
Best Lambda for Lasso: 0.02009233
```

```r
# Cross-validation RMSE
cv_rmse_lasso <- sqrt(min(cv_lasso$cvm))
cat("Lasso CV RMSE:", cv_rmse_lasso, "\n")
```

```
Lasso CV RMSE: 5.728481
```

```r
# Train Lasso model with best lambda
lasso_model <- glmnet(train_X_scaled, train_y, alpha = 1,
                      lambda = best_lambda_lasso)

# View the coefficients
print(coef(lasso_model))
```

```
14 x 1 sparse Matrix of class "dgCMatrix"
                     s0
(Intercept)  3.46735510
zn           0.76544240
indus       -0.34333197
chas        -0.07254647
nox         -1.17924331
rm          -0.03527903
age          0.17250010
dis         -1.66459412
rad          4.66220270
tax         -0.24006646
ptratio     -0.57915181
black       -0.66101761
lstat        0.87008853
medv        -1.25442541
```

```r
# Make predictions on test data
predictions_lasso <- predict(lasso_model, s = best_lambda_lasso,
                             newx = test_X_scaled)

# Compute Test RMSE
test_rmse_lasso <- sqrt(mean((test_y - predictions_lasso)^2))
```

```r
cat("Lasso Test RMSE:", test_rmse_lasso, "\n")
```

```
Lasso Test RMSE: 9.001981
```

Ridge and Lasso did not significantly improve generalization over Least Squares, suggesting that regularization had minimal effect, likely due to weak multicollinearity and the absence of clearly irrelevant predictors.

PLS:

```r
# Define predictors (X) and target variable (y)
X <- Boston[, -which(names(Boston) == "crim")]  # Remove crim from predictors
y <- Boston$crim

# Split data into 80% training (non-test) and 20% test set
set.seed(123)  # For reproducibility
train_index <- createDataPartition(y, p = 0.8, list = FALSE)
train_data <- Boston[train_index, ]
test_data <- Boston[-train_index, ]
```

```r
# Define 10-fold Cross-Validation
train_control <- trainControl(method = "cv", number = 10)

# Train PLS regression model and tune the number of components
set.seed(123)
pls_model <- train(crim ~ ., data = train_data, method = "pls",
                   trControl = train_control, tuneLength = 10)

# Get the optimal number of components
best_ncomp <- pls_model$bestTune$ncomp
cat("Optimal number of components for PLS:", best_ncomp, "\n")
```

```
Optimal number of components for PLS: 10
```

```r
# Cross-validation RMSE
cv_rmse_pls <- min(pls_model$results$RMSE)
cat("PLS CV RMSE:", cv_rmse_pls, "\n")
```

```
PLS CV RMSE: 5.179474
```

```r
# Fit the final PLS model using the best number of components
final_pls_model <- plsr(crim ~ ., data = train_data, ncomp = best_ncomp,
                        validation = "none")
```

```r
# View model summary
summary(final_pls_model)
```

```
Data:   X dimension: 406 13
    Y dimension: 406 1
Fit method: kernelpls
Number of components considered: 10
TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X       80.34    96.61    98.69    99.40    99.82    99.93    99.96    99.98
crim    39.84    40.14    41.20    43.68    45.90    48.21    48.69    49.05
      9 comps  10 comps
X       99.99     100.0
crim    49.23      49.3
```

```r
# Make predictions on the test set
predictions_pls <- predict(final_pls_model, newdata = test_data,
                           ncomp = best_ncomp)

# Compute Test RMSE
test_rmse_pls <- sqrt(mean((test_data$crim - predictions_pls)^2))
cat("PLS Test RMSE:", test_rmse_pls, "\n")
```

```
PLS Test RMSE: 8.997268
```

This suggests PLS generalized slightly better than the other models during cross-validation. It also effectively reduced dimensionality while keeping important information.

(b) Choose: LS and PLS

Since LS performed nearly as well as PLS, it remains a strong candidate. LS is simpler to interpret and computationally cheaper than PLS.

PLS reduces dimensionality while keeping the most important variance in the predictors. Since it achieved the lowest CV RMSE, it might generalize slightly better. a

(c)

Least Squares (LS) uses all features. LS does not perform feature selection, using all predictors.

No, PLS reduces dimensionality. Instead of selecting individual features, PLS creates new components that summarize the information from the original variables.

Bonus question (20pts)

Consider a linear regression, fit by least squares to a set of training data $(x_1, y_1), \ldots, (x_N, y_N)$ drawn at random from a population. Let $\hat{\beta}$ be the least squares estimate. Suppose we have some test data $(\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_M, \tilde{y}_M)$ drawn at random from the same population as the training data. If $R_{\text{train}}(\beta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta^T x_i)^2$ and $R_{\text{test}}(\beta) = \frac{1}{M}\sum_{i=1}^{M}(\tilde{y}_i - \beta^T \tilde{x}_i)^2$. Show that

$$\mathrm{E}[R_{\text{train}}(\hat{\beta})] < \mathrm{E}[R_{\text{test}}(\hat{\beta})].$$

Solution:

Bonus:

We define training error and test error as:

$R_{train}(\beta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta^T x_i)^2$; $R_{test}(\beta) = \frac{1}{M}\sum_{i=1}^{M}(\tilde{y}_i - \beta^T x_i)^2$

$\hat{\beta}$ is the least square estimate.

① least squares minimizes $R_{train}(\hat{\beta})$ directly on the training set.

② The test set is independent of training data. In addition, the $\hat{\beta}$ was chosen to minimize error on training set. So it's not optimized for the test set. Then, the test set error will be higher than training set.

③ Taking expectation over different training sets:

$E[R_{train}(\hat{\beta})] = E[\frac{1}{N}\sum_{i=1}^{N}(y_i - \beta^T x_i)^2]$, $E[R_{test}(\hat{\beta})] = E[\frac{1}{M}\sum_{i=1}^{M}(\tilde{y}_i - \beta^T x_i)^2]$

Since $E[R_{test}(\hat{\beta})] = E[R_{train}(\hat{\beta})]$ + model variance, and variances always non-negative, then $E[R_{train}(\hat{\beta})] < E[R_{test}(\hat{\beta})]$

Therefore, training error underestimates test error due to overfitting.