hw_03

lisa liubovich

2024-06-06

1 Predicting a grade (CV in KNN)

Do by hand. A student wants to predict their grade for the Statistical Machine Learning course, using the KNN algorithm with $K = 3$. Six friends who took the course last year had the following mid-term test scores and grades.

Friend	1	2	3	4	5	6
Midterm	90	88	83	78	85	84
Course Grade	A	A	A	B	B	B

Estimate the prediction error rate of the algorithm, by means of:

(a) The validation-set method, using Friends 2, 3, 4, 5 as training and Friends 1, 6 as testing data.

Training Data: Friends 2, 3, 4, 5

Testing Data: 1, 6

Prediction for Friend 1 (Midterm = 90)

Distance to Friend 2: |90-88| = 2

Distance to Friend 3: |90-83| = 7

Distance to Friend 4: |90-78| = 12

Distance to Friend 5: |90-85| = 5

Nearest neighbors: 2, 5, and 7 → Grades: A, B, A

The predicted grade for friend 1 is an A.

Prediction for Friend 6 (midterm = 84)

Distance to Friend 2: |84-88| = 4

Distance to Friend 3: |84-83| = 1

Distance to Friend 4: |84-78| = 6

Distance to Friend 5: |84-85| = 1

Nearest neighbors: Friends 3, 5, and 2 → Grades: A, B, A

The predicted grade for friend 6 is an A.

Error rate: incorrect predictions/total test data = 1/2 = 0.5

(b) The leave-one-out cross-validation method.

Prediction for Friend 1 (Midterm = 90)

Training: 2, 3, 4, 5, 6

Distance to Friend 2: |90-88| = 2

Distance to Friend 3: |90-83| = 7

Distance to Friend 4: |90-78| = 12

Distance to Friend 5: |90-85| = 5

Distance to Friend 6: |90-84| = 6

Nearest neighbors: 2, 5, and 6 → Grades: A, B, B

The predicted grade for friend 1 is an B, which is incorrect.

Prediction for Friend 2 (Midterm = 88)

Training: Friends 1, 3, 4, 5, 6

Distance to Friend 1: |88-90| = 2

Distance to Friend 3: |88-83| = 5

Distance to Friend 4: |88-78| = 10

Distance to Friend 5: |88-85| = 3

Distance to Friend 6: |88-84| = 4

Nearest neighbors: Friends 1, 5, 6 → Grades: A, B, B

The predicted grade for friend 2 is B, which is Incorrect

Prediction for Friend 3 (Midterm = 83)

Training: Friends 1, 2, 4, 5, 6

Distance to Friend 1: 7

Distance to Friend 3: 5

Distance to Friend 4: 5

Distance to Friend 5: 2

Distance to Friend 6: 1

Nearest neighbors: Friends 2, 5, 6 → Grades: B, B, A

The predicted grade for friend 3 is B, which is incorrect

Prediction for Friend 4 (Midterm = 78)

Training: Friends 1, 2, 3, 5, 6

Distance to Friend 1: 12

Distance to Friend 2: 10

Distance to Friend 3: 5

Distance to Friend 5: 7

Distance to Friend 6: 6

Nearest neighbors: Friends 3, 5, 6 → Grades: A, B, B

The predicted grade for friend 4 is B, which is correct

Prediction for Friend 5 (Midterm = 85)

Training: Friends 1, 2, 4, 5, 6

Distance to Friend 1: 5

Distance to Friend 2: 3

Distance to Friend 3: 2

Distance to Friend 4: 7

Distance to Friend 6: 1

Nearest neighbors: Friends 2, 3, 6 → Grades: B, A, A

The predicted grade for friend 5 is A which is incorrect

Prediction for Friend 6 (Midterm = 84)

Training: Friends 1, 2, 3, 4, 5

Distance to Friend 1: 6

Distance to Friend 2: 4

Distance to Friend 3: 1

Distance to Friend 4: 6

Distance to Friend 5: 1

Nearest neighbors: Friends 2, 3, 5 → Grades: A, B, A

The predicted grade for friend 6 is A which is incorrect

Error rate: 5/6 = 0.833

(c) (Stat 627) Use knn.cv() function in package class to confirm your computation in (b). (You can start with reading the help file of knn.cv().)

```
library(class)

midterm_scores <- c(90, 88, 83, 78, 85, 84)
grades <- factor(c("A", "A", "A", "B", "B", "B"))

predicted_grades <- knn.cv(train = midterm_scores, cl = grades, k = 3)

error_rate <- mean(predicted_grades != grades)
print(error_rate)
```

```
## [1] 0.8333333
```

2 Ex.5.4.8. Cross-validation in linear regression on simulated data

(a) Generate a simulated data set as follows:

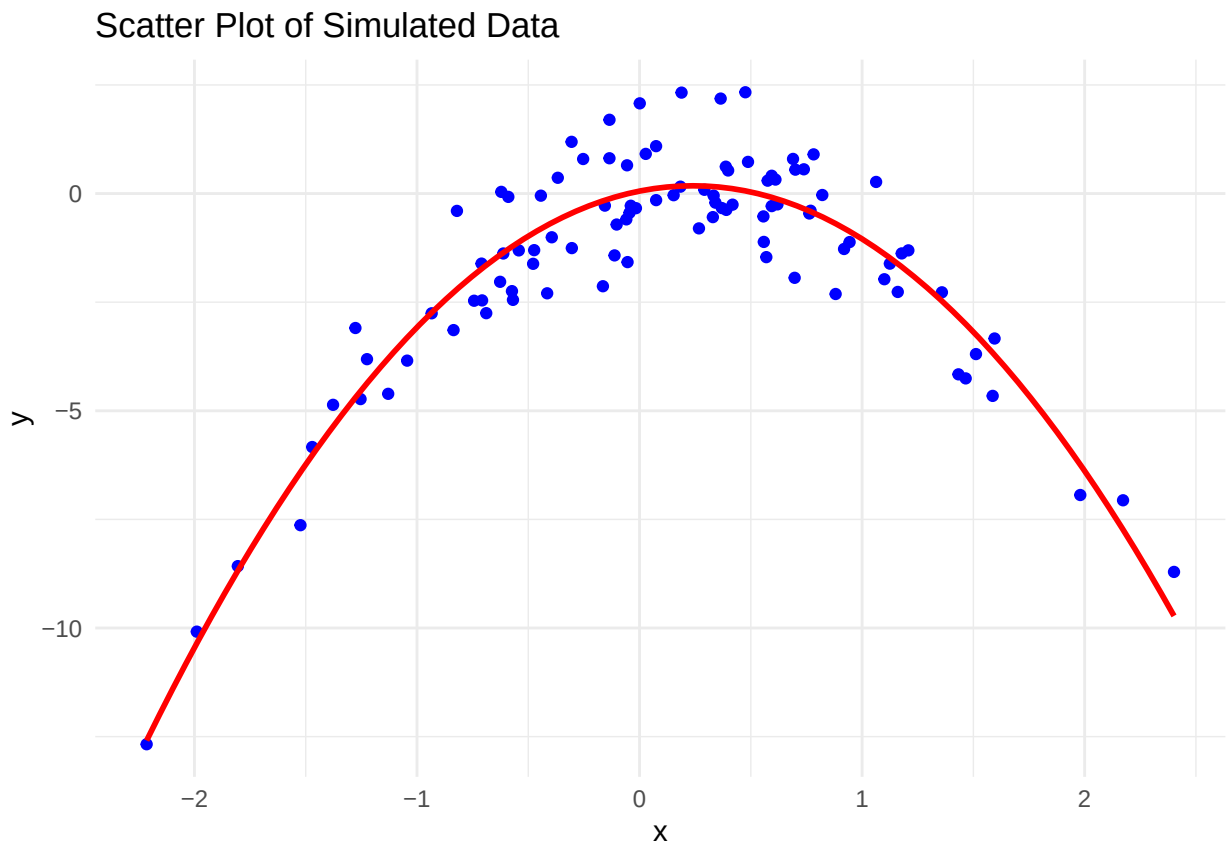
```
set.seed(1)
x <- rnorm (100)
y <- x - 2*x^2 + rnorm(100)
sim.df <- data.frame(x, y) # data.frame will be helpful in part (b), (c)
```

In this data set, what is n and p? Write out the model used to generate the data in equation form. Plot the data and interpret the plot.

n = 100, p = 1

Model used to generate data: y = x - $2x^2 + \varepsilon$, where ε is a random error term drawn for a normal distribution $\varepsilon \sim N(0,1)$.

```r
library(ggplot2)
ggplot(sim.df, aes(x = x, y = y)) +
  geom_point(color = "blue") +
  labs(title = "Scatter Plot of Simulated Data",
       x = "x",
       y = "y") +
  theme_minimal() +
  geom_smooth(method = "lm", formula = y ~ poly(x, 2), se = FALSE, color = "red")
```



The curve shows the quadratic trend of the data, which makes sense giving the squared term in the equation. The dispersion of points around the red line is relatively even but a bit wider towards the peak of the parabola, which indicates more significant noise in the data.

(b) Compute the LOOCV estimates of prediction error that result from fitting each of the following four regression models: (Hints: (1) LOOCV is the same as K-fold CV with K=(sample size). (2) Use function glm() fits Normal linear regression when you set family=gaussian. (3). Use function cv.glm() in pacakgeboot‘.)

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \varepsilon$$

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \varepsilon$$

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \varepsilon$$

```r
library(boot)
compute_loocv_error <- function(formula, data) {
  model <- glm(formula, data = data, family = gaussian)
  cv_result <- cv.glm(data, model, K = nrow(data))
  return(cv_result$delta[1])  # LOOCV error
}


formula_1 <- y ~ x
formula_2 <- y ~ x + I(x^2)
formula_3 <- y ~ x + I(x^2) + I(x^3)
formula_4 <- y ~ x + I(x^2) + I(x^3) + I(x^4)


loocv_error_1 <- compute_loocv_error(formula_1, sim.df)
loocv_error_2 <- compute_loocv_error(formula_2, sim.df)
loocv_error_3 <- compute_loocv_error(formula_3, sim.df)
loocv_error_4 <- compute_loocv_error(formula_4, sim.df)


loocv_error_1
```

```
## [1] 7.288162
```

```r
loocv_error_2
```

```
## [1] 0.9374236
```

```r
loocv_error_3
```

```
## [1] 0.9566218
```

```r
loocv_error_4
```

```
## [1] 0.9539049
```

(c) Which of these models have the smallest adjusted prediction mean squared error as estimated by LOOCV? Is this what you expected? Explain your answer.

Model 1 is a simple linear model, which likely underfits the data because the true relationship is quadratic. This explains the relatively high LOOCV error of 7.288162. Model 2 includes a quadratic term, which aligns well with the underlying data-generating process. This model captures the main structure of the data without overfitting, resulting in the smallest LOOCV error. Model 3 and Model 4 add cubic and quartic terms, respectively. These higher-order terms might slightly improve the fit to the data, but they also increase the complexity of the model, which can lead to overfitting and an increase in LOOCV error compared to the simpler quadratic model. This explains why their LOOCV errors (0.9566218 and 0.9539049) are slightly higher than that of Model 2.

Expected results: The result that Model 2 has the smallest LOOCV error is expected because the data was generated using a quadratic model. Therefore, including the x^2 term should capture the underlying relationship accurately. Model 1's poor performance is also expected because a linear model cannot capture the curvature of the true quadratic relationship.T he higher LOOCV errors for

Model 3 and Model 4 are due to the additional complexity from higher-order polynomial terms, which introduces more variance without significantly reducing bias for this data set. Therefore, the best fit is Model 2 balancing bias and variance effectively. The higher-order models do not provide significant improvement and might even slightly degrade predictive performance due to overfitting.

(d) Repeat step b but with $K = 10$-Fold validation and compare the prediction mean squared error. What do you notice compared to LOOCV?

- Use set.seed(12) before **each** cv.glm() call.

```
compute_10fold_cv_error <- function(formula, data) {
  set.seed(12)  # Ensure reproducibility for each model's CV
  model <- glm(formula, data = data, family = gaussian)
  cv_result <- cv.glm(data, model, K = 10)
  return(cv_result$delta[1])  # 10-Fold CV error
}

formula_1 <- y ~ x
formula_2 <- y ~ x + I(x^2)
formula_3 <- y ~ x + I(x^2) + I(x^3)
formula_4 <- y ~ x + I(x^2) + I(x^3) + I(x^4)


cv_error_1 <- compute_10fold_cv_error(formula_1, sim.df)
cv_error_2 <- compute_10fold_cv_error(formula_2, sim.df)
cv_error_3 <- compute_10fold_cv_error(formula_3, sim.df)
cv_error_4 <- compute_10fold_cv_error(formula_4, sim.df)


cv_error_1
```

```
## [1] 8.205558
```

```
cv_error_2
```

```
## [1] 0.9457335
```

```
cv_error_3
```

```
## [1] 1.032554
```

```
cv_error_4
```

```
## [1] 1.271199
```

Model 2 is the best-performing model based on both LOOCV and 10-fold CV errors. It has the lowest cross-validation errors, reflecting its suitability for capturing the true quadratic relationship in the data. The quadratic model balances bias and variance effectively, providing a good fit to the data without overfitting. Higher-order polynomial models (Models 3 and 4) do not improve performance and may introduce overfitting, as indicated by their higher cross-validation errors.

(e) (Stat-627) In part (c) (LOOCV), we did not use set.seed(). In part (d) (10-fold CV), we used set.seed(). The random seed is set so that we can get replicate the the same results for our homework practice. Why is the random seed relevant in the 10-fold CV but not in LOOCV?

The random seed is crucial for 10-fold CV due to the inherent randomness in how the data is split into folds. It ensures that the process can be repeated with the same results, which is key for reproducibility. On the other hand, LOOCV does not require a random seed because the data splitting process is deterministic and consistent across different runs.

3 Ex.5.4.5. Predicting defaults on loans

Use the Default data set in {ISLR2} package to create a logistic regression model for predicting the probability of variable default based on predictors income, balance, and student.

Use each of the following methods to estimate the *test error rate* of the logistic regression model and decide whether it will be improved if the dummy variable student is excluded from the prediction.

- Use a seed of 123 and a threshold of .5 where appropriate.

(a) The validation set approach with a 60% split. I.e. split the data set only once, 60% of the observation will be used for training, and the the remaining 40% will be used for validation/testing.

```
library(ISLR2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'lattice'

## The following object is masked from 'package:boot':
##
##     melanoma
```

```r
set.seed(123)

data(Default)

Default <- Default %>%
  mutate(student = ifelse(student == "Yes", 1, 0))


compute_test_error <- function(data, formula, method, threshold = 0.5, include_student = TRUE) {
  if (!include_student) {
    formula <- update(formula, . ~ . - student)
  }
  model <- glm(formula, data = data, family = binomial)
  probabilities <- predict(model, newdata = data, type = "response")
  predicted_classes <- ifelse(probabilities > threshold, 1, 0)
  confusion_matrix <- table(data$default, predicted_classes)
  test_error_rate <- 1 - sum(diag(confusion_matrix)) / sum(confusion_matrix)
  return(list(confusion_matrix = confusion_matrix, test_error_rate = test_error_rate))
}

split_index <- createDataPartition(Default$default, p = 0.6, list = FALSE)
training_data <- Default[split_index, ]
validation_data <- Default[-split_index, ]


formula <- default ~ income + balance + student

validation_results_with_student <- compute_test_error(validation_data, formula, method = "validatio
validation_results_without_student <- compute_test_error(validation_data, formula, method = "valida
print("Validation Set Approach:")
```

```
## [1] "Validation Set Approach:"
```

```r
print(paste("With student:", validation_results_with_student$test_error_rate))
```

```
## [1] "With student: 0.0260065016254063"
```

```r
print(paste("Without student:", validation_results_without_student$test_error_rate))
```

```
## [1] "Without student: 0.0257564391097774"
```

The error rate improves slightly without the student dummy variable.

(b) Leave-one-out cross-validation. (Your computer may take a really long time to run the code on LOOCCV due to the large sample size. Considering using a chunk option for cache, e.g., set {r, cache=TRUE}.)

```r
loocv_results_with_student <- compute_test_error(Default, formula, method = "loocv", include_studen
loocv_results_without_student <- compute_test_error(Default, formula, method = "loocv", include_stu
print("Leave-One-Out Cross-Validation (LOOCV):")
```

```
## [1] "Leave-One-Out Cross-Validation (LOOCV):"
```

```
print(paste("With student:", loocv_results_with_student$test_error_rate))
```

```
## [1] "With student: 0.0268"
```

```
print(paste("Without student:", loocv_results_without_student$test_error_rate))
```

```
## [1] "Without student: 0.0263"
```

The error rate improves slightly when excluding the student dummy variable.

(c) K-fold cross-validation for $K = 100$ and $K = 1000$.

```
k_fold_100_results_with_student <- train(formula, data = Default, method = "glm", trControl = train
k_fold_100_results_without_student <- train(update(formula, . ~ . - student), data = Default, metho
print("K-Fold Cross-Validation for K = 100:")
```

```
## [1] "K-Fold Cross-Validation for K = 100:"
```

```
print(paste("With student:", k_fold_100_results_with_student$results$Accuracy))
```

```
## [1] "With student: 0.973228782878288"
```

```
print(paste("Without student:", k_fold_100_results_without_student$results$Accuracy))
```

```
## [1] "Without student: 0.973808780878088"
```

```
k_fold_1000_results_with_student <- train(formula, data = Default, method = "glm", trControl = trai
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
k_fold_1000_results_without_student <- train(update(formula, . ~ . - student), data = Default, meth
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
print("K-Fold Cross-Validation for K = 1000:")
```

```
## [1] "K-Fold Cross-Validation for K = 1000:"
```

```
print(paste("With student:", k_fold_1000_results_with_student$results$Accuracy))
```

```
## [1] "With student: 0.974551515151515"
```

```
print(paste("Without student:", k_fold_1000_results_without_student$results$Accuracy))
```

```
## [1] "Without student: 0.975050505050505"
```

For the K = 100 and K = 1000, the error rate slightly improves without the inclusion student.

4 Cross-validation in LDA and QDA

Refer the R example handouts. Find the example of cross-validation in LDA and QDA. Is it LOOCV or K-fold CV?

Example 3.2 in the 3_4 Classification handout shows an example of LOOCV in LDA.

Example 3.5 in the 3_4 Classification handout shows an example of LOOCV in QDA.

5 Basses and sopranos (Jackknife)

An acoustic studio needs to estimate the range of voice fundamental frequencies that an adult singer can produce. A sample of $n = 10$ recordings contains frequencies 102, 115, 127, 127, 162, 180, 184, 205, 239, 240.

(a) Manually compute (by hand) the jackknife estimator of the population lowest fundamental frequency of a human voice. Compare your results with the natural range of human voice frequencies. (Use Google or Wiki.)

Sample Mean: (102+115+127+127+162+180+184+205+239+240)/10 = 168.10

Jackknife sample 1: remove 102

mean of remaining frequencies: (115+127+127+162+180+184+205+239+240)/9=171.44

Jackknife Sample 2: Remove 115

Mean of remaining frequencies: (102+127+127+162+180+184+205+239+240)/9=172.33

Jackknife Sample 3: Remove 127 (first occurrence)

(102+115+127+162+180+184+205+239+240)/9=167.89

Jackknife Sample 4: Remove 127 (second occurrence)

(102+115+127+162+180+184+205+239+240)/9=167.89

Jackknife Sample 5: Remove 162

(102+115+127+127+180+184+205+239+240)/9=166.88

Jackknife Sample 6: Remove 180

(102+115+127+127+162+184+205+239+240)/9=165.99

Jackknife Sample 7: Remove 184

(102+115+127+127+162+180+205+239+240)/9=165.1

Jackknife Sample 8: Remove 205

(102+115+127+127+162+180+184+239+240)/9=164.1

Jackknife Sample 9: Remove 239

(102+115+127+127+162+180+184+205+240)/9=159.88

Jackknife Sample 10: Remove 240

(102+115+127+127+162+180+184+205+239)/9=159.88

Jackknife estimator: 159.88

The natural range of human voice frequencies varies widely, but for adult males, it typically ranges from 85 Hz to 180 Hz, and for adult females, it ranges from 165 Hz to 255 Hz.

b. Use software to confirm your result.

```r
frequencies <- c(102, 115, 127, 127, 162, 180, 184, 205, 239, 240)

jackknife_estimator <- function(data) {
  n <- length(data)
  jackknife_means <- numeric(n)
  for (i in 1:n) {
    jackknife_means[i] <- mean(data[-i])
  }
  return(min(jackknife_means))
}

jackknife_result <- jackknife_estimator(frequencies)
jackknife_result
```

```
## [1] 160.1111
```

c. (Stat-627 only) Generalize the results. Assume a sample $X_1,\ldots,X_n$ of size n, where X_1,X_2 are the smallest two observations. Derive equations for the jackknife estimators of the population minimum. Use your result in (a) to verify your formula.

Mathematically, the jackknife estimator is

$$\hat{X}_{(1)} = \min_{i=1}^{n} \left(\frac{(n-1)\bar{X} - X_i}{n-1} \right)$$

where:

$$\bar{X} = \frac{1}{n} \sum_{j=1}^{n} X_j$$

is the sample mean.

Therefore, the jackknife estimator is calculated by:

$$\hat{X}_{(1)} = \min_{i=1}^{n} \left(\bar{X}_{(n)} - \frac{X_i - \bar{X}}{n-1} \right)$$

where:

$\bar{X}_{(n)}$ is the sample mean of all n observations.

See part a.

6 Ex.5.4.9. Bootstrap the mean of median house values in the Boston dataset.

We will now consider the Boston housing data set from the {MASS} library.

(a) Based on this data set, provide an estimate for the population meanμ of medv, which is the median value of owner-occupied homes in \$1,000s. Call this estimate $\hat{\mu}$

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```
data(Boston)

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

```
## [1] 22.53281
```

(b) Estimate the standard error of $\hat{\mu}$ (as we know, StdError($\bar{X}$) = $s/\sqrt{n}$, where s is the sample standard deviation. R function sd().)

```
s <- sd(Boston$medv)
se_mu_hat <- s / sqrt(length(Boston$medv))
se_mu_hat
```

```
## [1] 0.4088611
```

(c) Estimate the standard error of $\hat{\mu}$ using the bootstrap method. How does this compare to your answer from (b)?. Remember to set your seed to be reproducible.

```
set.seed(123)
B <- 1000

bootstrap_mu_hat <- replicate(B, mean(sample(Boston$medv, replace = TRUE)))
se_bootstrap_mu_hat <- sd(bootstrap_mu_hat)
se_bootstrap_mu_hat
```

```
## [1] 0.4185474
```

This standard error is slightly larger than the standard error in part b.

(d)Based on your bootstrap estimate from (c), provide a 95% confidence interval for μ. A popular approximation is $\hat{\mu} \pm 2$StdError($\hat{\mu}$).

Compare it to the results obtained using an R command t.test(Boston$medv).

```
lower_bound <- mu_hat - 2 * se_bootstrap_mu_hat
upper_bound <- mu_hat + 2 * se_bootstrap_mu_hat

t_test_result <- t.test(Boston$medv)
t_test_ci <- t_test_result$conf.int

print(lower_bound)
```

```
## [1] 21.69571
```

```
print(upper_bound)
```

```
## [1] 23.3699
```

```
print(t_test_ci)
```

```
## [1] 21.72953 23.33608
## attr(,"conf.level")
## [1] 0.95
```

(e) Now, estimate M, the population median of medv with the sample median $\hat{M}$.

```
M_hat <- median(Boston$medv)
M_hat
```

```
## [1] 21.2
```

(f) We now would like to estimate the standard error of $\hat{M}$, but unfortunately, there is no simple formula for computing the standard error of a sample median. Instead, estimate this standard error using the bootstrap method.

```
bootstrap_M_hat <- replicate(B, median(sample(Boston$medv, replace = TRUE)))
se_bootstrap_M_hat <- sd(bootstrap_M_hat)

se_bootstrap_M_hat
```

```
## [1] 0.3779944
```