Bootstrap Confidence Intervals

Cheng Peng

West Chester University

Table of Contents

Introduction

The bootstrap is a resampling technique introduced by Efron (1979) that allows us to estimate the sampling distribution of a statistic by **repeatedly** sampling **with replacement** from the **observed data**.

The **Key Principle** is treat the empirical distribution function (EDF) of the sample as an approximation to the true population distribution:

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} I\left(X_i \leq x\right)$$

This key principle is backed by the well known **Glivenko–Cantelli Theorem**: EDF converges uniformly to CDF for i.i.d. data.

Key Explicit and Implicit Assumptions

Bootstrap samples represent the original population when the original sample itself is a good representation of the population. This is the fundamental bootstrap principle.

- Key Explicit Conditions
  - **Independence of observations**: The original data should be independent and identically distributed (i.i.d.) or at least exchangeable. The standard bootstrap sampling fails for time series

with strong autocorrelation, spatial data, clustered data (without special resampling methods).
    - **Sample size sufficiently large**: The original sample size n should be large enough to capture the population's variability. Small samples can still be bootstrapped, but the bootstrap distribution may be coarse and might miss extreme quantiles of the population
    - **Statistic is smooth and well-behaved**: Bootstrap works well for statistics that are smooth functions of the sample (e.g., mean, variance, regression coefficients). **However**, it can fail for non-smooth statistics (e.g., median with very small samples, extreme order statistics).
- Key Implicit Assumptions
    - **The empirical distribution function (EDF) is a good estimate of the population CDF** because bootstrap resampling approximates sampling from the EDF, not directly from the population. If the original sample is not representative (e.g., biased sampling), bootstrap will replicate the bias.
    - **The underlying population distribution has finite variance** which ensures consistency of bootstrap for the sample mean (and related statistics) in large samples.
    - **No extreme outliers driving the estimate** because bootstrap samples from the observed data, a single extreme outlier can appear multiple times in a bootstrap sample, distorting the bootstrap distribution more than the actual sampling distribution.

When Bootstrap Fails or Is Problematic

- **Heavy-tailed distributions with small n** — bootstrap can be inconsistent.
- **Non-smooth parameters** (e.g., estimating max of distribution).
- **Data with dependency structure** (unless block bootstrap or similar is used).
- **Insufficient sample size** for complex statistics.

Bootstrap Algorithm vs Bootstrap Sampling Distribution

The Basic Bootstrap Algorithm (also called the **nonparametric bootstrap**) is a resampling-based statistical method introduced by Bradley Efron in 1979. It provides a way to estimate the sampling distribution of **almost any statistic without making strong distributional assumptions**. It's power lies in its

simplicity: by letting the computer do the heavy lifting through resampling, we can make inferences that would be mathematically intractable using traditional methods.

Bootstrap Algorithm

The bootstrap algorithm is a resampling technique that uses a single dataset to estimate the sampling distribution of a statistic by repeatedly drawing samples with replacement from the original data and recalculating the statistic.

Mathematical Algorithm

Let $X = \{x_1, x_2, \ldots, x_n\}$ be the original sample:

- Compute the statistic of interest: $\hat{\theta} = T(X)$
- For $b = 1$ to B:
  - Draw a bootstrap sample X_b^* by sampling n observations with replacement from X
  - Compute $\hat{\theta}_b^* = T(X_b^*)$
- The bootstrap distribution is $\{\hat{\theta}_1^*, \hat{\theta}_2^*, \cdots, \hat{\theta}_B^*\}$

Pseudo-code

```
function basic_bootstrap(data, statistic, B=1000):
    n = length(data)
    theta_hat = statistic(data)
    bootstrap_dist = empty vector of size B

    for b in 1:B:
        bootstrap_sample = sample(data, size=n, replace=TRUE)
        bootstrap_dist[b] = statistic(bootstrap_sample)

    return (theta_hat, bootstrap_dist)
```

Types of Bootstrap Algorithms

- **Nonparametric Bootstrap** is the most common type (described above). It makes no assumptions about population distribution and resamples directly from observed data.
- **Parametric Bootstrap** assumes data comes from parametric family $F(x; \theta)$. We take an original sample from the population and estimate θ from data.

The **bootstrap samples** from $F(x;\hat{\theta}$ (instead of the ECDF $F_n(x)$) and calculate statistics from these **parametric** bootstrap samples.

- **Specialized Bootstrap** is for non-i.i.d. data or data with special probability structures. We will not explore this direction in depth.

Bootstrap Sampling Distributions

Bootstrap is a versatile, assumption-lean method for estimating sampling distributions of **sample statistics**, particularly valuable when theoretical distributions of these statistics are unknown or difficult to derive, but requires careful implementation and interpretation.

The following is a list frequently used bootstrap sampling distributions used in practice.

- **Distribution of bootstrap sample means**
- **Distribution of bootstrap sample medians**
- **Distribution of bootstrap sample variances or standard deviations**
- **Distribution of bootstrap correlation coefficients (Pearson's r)**
- **Distribution of bootstrap regression coefficients**
- **Distribution of bootstrap sample proportions**
- **Distribution of bootstrap ratios of sample statistics (e.g., cost-effectiveness ratios)**
- **Distribution of bootstrap statistics of any population parameters**

The above statistics can be estimated based on any formal estimation methods such as method of moments estimator (MME) and maximum likelihood estimator (MLE).

Applications of Bootstrap Sampling Distribution

The applications of Bootstrap Sampling Distribution span many areas of statistics and data science, primarily focused on **estimation**, **inference**, and **assessing uncertainty** when theoretical formulas are unavailable or unreliable. This course, we explore the applications in estimation and hypothesis testing.

Point and Interval Estimation

Bootstrap sampling distribution can be used for point and interval estimation. The

Hypothesis Testing

Types of Bootstrap Confidence Intervals

Bootstrap CIs include the percentile method (simple, range-preserving), BCa (corrects bias/skew, recommended default), bootstrap-t (most accurate with stable SE), and normal approximation (fast but assumes normality). BCa generally offers the best balance of accuracy and practicality for most applications.

Normal (Standard) Bootstrap CI

This bootstrap confidence interval **assumes** that bootstrap sampling distribution is approximately normally distributed.

Mathematical Algorithm

The algorithm is relatively straightforward and simple. We summarize it in the following 4 steps:

- Compute $\hat{\theta}$ from original sample
- Generate bootstrap distribution $\{\hat{\theta}_b^*\}_{b=1}^B$
- Estimate standard error: with $\bar{\theta}^* = \frac{1}{B}\sum_{b=1}^B \hat{\theta}_b^*$

$$\widehat{se}_{boot} = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\theta}_b^* - \bar{\theta}^*\right)^2}$$

- Confidence interval:

$$\hat{\theta} \pm z_{\alpha/2} \cdot \widehat{se}_{boot}$$

Pseudo-code

Instead of writing an R function to implement the normal (standard bootstrap) confidence interval, this is a good coding exercise to translate the pseudo code into a workable R function to construct the normal bootstrap confidence interval.

```
function normal_bootstrap_ci(data, statistic, B=1000, alpha=0.05):
    theta_hat, boot_dist = basic_bootstrap(data, statistic, B)
    se_boot = sd(boot_dist)
    z = qnorm(1 - alpha/2)
    lower = theta_hat - z * se_boot
```

```
    upper = theta_hat + z * se_boot
    return (lower, upper)
```

An R Example

In this example, we present a step-by-step approach for calculating the normal confidence interval using R code with a simulated data set.

```
# Normal Bootstrap CI for mean
set.seed(123)
example1.dat <- rnorm(30, mean=50, sd=10)  # simulate a normal random s
ample

# Manual implementation
B <- 1000
n <- length(example1.dat)
boot_means <- numeric(B)     # empty vector to store bootstrap sample m
eans

for(b in 1:B) {
  boot_sample <- sample(example1.dat, n, replace=TRUE)  # generating bo
otstrap sample
  boot_means[b] <- mean(boot_sample)            # calculate bootstrap m
eans
}

theta_hat <- mean(example1.dat)   # same mean from the original sample
se_boot <- sd(boot_means)
z <- qnorm(0.975)
ci_normal <- c(LCI = theta_hat - z*se_boot, xbar = theta_hat, UCL=theta
_hat + z*se_boot)
ci_normal

     LCI     xbar      UCL
46.02590 49.52896 53.03203
```

Percentile Bootstrap CI

Percentile bootstrap confidence intervals are the simplest form of bootstrap CIs. They are constructed by directly taking the corresponding quantiles (e.g., 2.5% and 97.5%) from the empirical distribution of bootstrap estimates. This method is range-preserving (e.g., stays within [0,1] for proportions) and works well when the bootstrap distribution is roughly symmetric and the estimator has little bias.

Mathematical Algorithm

- **Resample**: Draw B independent bootstrap samples (with replacement) of size n from the original data.
- **Compute statistic**: Generate bootstrap distribution $\{\hat{\theta}_b^*\}_{b=1}^B$

- **Empirical distribution**: Sort bootstrap statistics:

$$\hat{\theta}^*_{(1)} \leq \hat{\theta}^*_{(2)} \leq \ldots \leq \hat{\theta}^*_{(B)}$$

- **Percentiles**: For a $(1-\alpha$ confidence interval, take the $\alpha/2$ and $1-\alpha/2$ empirical percentiles of the bootstrap distribution.

$$\text{Percentile CI} = \left[\hat{\theta}^*_{(B\cdot\alpha/2)}, \hat{\theta}^*_{(B\cdot(1-\alpha/2))}\right]$$

Pseudo-code

The percentile bootstrap method is widely used in practice, especially when dealing with skewed bootstrap sampling distributions. The following pseudocode details the steps required to write an R function for constructing percentile bootstrap confidence intervals.

```
function percentile_ci(data, statistic, B=1000, alpha=0.05):
    theta_hat, boot_dist = basic_bootstrap(data, statistic, B)
    sorted_boot = sort(boot_dist)
    lower_idx = floor(B * alpha/2)
    upper_idx = ceiling(B * (1 - alpha/2))
    lower = sorted_boot[lower_idx]
    upper = sorted_boot[upper_idx]
    return (lower, upper)
```

An R Example

We will use built-in R functions `boot()` and `boot.ci()` to construct the percentile bootstrap confidence interval based on a simulated random sample from an exponential distribution. A manual calculation based on the formula of the percentile bootstrap confidence interval using the same data

```r
# Percentile Bootstrap CI
set.seed(123)
data <- rexp(25, rate=0.1)  # generating exponential data

# Using boot package
# library(boot)

# Define statistic function
mean_stat <- function(data, indices) {
  sample_data <- data[indices]
  return(mean(sample_data))
}

# Perform bootstrap
boot_result <- boot(data, mean_stat, R=1000)

# Percentile CI
boot.ci(boot_result, type="perc")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_result, type = "perc")

Intervals :
Level     Percentile
95%   ( 5.819, 12.504 )
Calculations and Intervals on Original Scale
```

```
# Manual calculation
boot_vals <- boot_result$t
alpha <- 0.05
ci_perc <- quantile(boot_vals, probs=c(alpha/2, 1-alpha/2))
ci_perc
```

```
     2.5%     97.5%
 5.862205 12.492416
```

Bias-Corrected and Accelerated (BCa) CI

BCa (Bias-Corrected and Accelerated) is an improved bootstrap CI that adjusts for both bias and skewness in the sampling distribution. It computes adjusted percentiles using two correction factors:

- **Bias correction (z_0)**: measures median bias by comparing bootstrap estimates to the original estimate.
- **Acceleration (a)**: measures how the standard error changes with the parameter value (estimated via jackknife).

It generally provides more accurate coverage than the simple percentile method, especially for skewed distributions or biased estimators, and remains transformation-respecting. It is often the recommended default for bootstrap confidence intervals when higher accuracy is needed.

Constructing of **BCa CI** requires **Jackknife** algorithm which is **resampling technique** used primarily for **bias correction** and **variance estimation**. It systematically recomputes a statistic by **leaving out one observation at a time** from the sample, creating multiple **pseudo-samples** of size $n - 1$.

In the next subsection, we present the two mathematical algorithms.

Mathematical Algorithms

Jackknife Resampling

- **Original sample**: $x_1, x_2, \ldots, x_n$ with statistic $\hat{\theta}$.
- **Jackknife samples**: Create n subsamples, each omitting one observation:

- **Compute statistic**: Calculate $\hat{\theta}_{(i)}$ for each jackknife sample.

BCa CI Algorithm

- Compute bias correction:

$$\hat{z}_0 = \Phi^{-1}\left(\frac{\#\{\hat{\theta}_b^* < \hat{\theta}\}}{B}\right)$$

- Compute acceleration $\hat{a}$ using jackknife:

- Compute adjusted percentiles:

- Confidence interval:

$$\left[\hat{\theta}^*_{(B\cdot\alpha_1)}, \hat{\theta}^*_{(B\cdot\alpha_2)}\right]$$

Pseudo-code

The following pseudo code for BCa CI contains **Jackknife Algorithm** for calculating the acceleration doefficient.

```
function bca_ci(data, statistic, B=1000, alpha=0.05):
    theta_hat, boot_dist = basic_bootstrap(data, statistic, B)

    # Bias correction
    prop_less = sum(boot_dist < theta_hat) / B
    z0 = qnorm(prop_less)

    # Acceleration (jackknife)
    n = length(data)
    jack_vals = numeric(n)
    for i in 1:n:
        jack_sample = data[-i]
        jack_vals[i] = statistic(jack_sample)
    theta_jack_mean = mean(jack_vals)
    numerator = sum((theta_jack_mean - jack_vals)^3)
    denominator = sum((theta_jack_mean - jack_vals)^2)
```

```
    a_hat = numerator / (6 * denominator^(3/2))

    # Adjusted percentiles
    z_alpha = qnorm(alpha/2)
    z_1alpha = qnorm(1 - alpha/2)

    alpha1 = pnorm(z0 + (z0 + z_alpha)/(1 - a_hat*(z0 + z_alpha)))
    alpha2 = pnorm(z0 + (z0 + z_1alpha)/(1 - a_hat*(z0 + z_1alpha)))

    sorted_boot = sort(boot_dist)
    lower = quantile(sorted_boot, probs=alpha1)
    upper = quantile(sorted_boot, probs=alpha2)

    return (lower, upper)
```

Practical Recommendations

BCa is the most reliable general-purpose bootstrap CI method for non-symmetric distributions but requires careful implementation and diagnostic checking. When in doubt, compare with **percentile** and **studentized intervals** (to be introduced in the next subsection) for consistency

- **When to Use**:
  - Non-symmetric distributions, small-to-medium samples (n < 200), biased estimators
  - Avoid for n < 20 or extremely heavy-tailed data
- **Implementation**:
  - Minimum $B = 1{,}000$ replicates ($\$,000\$ for publication/research)
  - Always check bootstrap distribution shape visually
  - Monitor stability: increase B until intervals change $< 1\%$
- **Key Diagnostics**:
  - Acceleration value $|acc| < 0.3$ (red flag if > 0.5)
  - Compare with percentile method - results should be similar for symmetric data
  - Plot histogram of bootstrap replicates for skewness assessment
- **Common Pitfalls**:
  - Too few replicates ⇒ unstable intervals
  - Discrete/discontinuous statistics ⇒ use smoothed bootstrap
  - Clustered/time series data ⇒ need specialized bootstrap variants
- **Reporting**:
  - **Always specify**: point estimate, CI, confidence level, B, n
  - **Example**: "15.2 (BCa 95% CI: [8.7, 21.9]; B=10,000, n=45)"

An R Example

In the following example with simulated data, we use the built-in R functions `boot()` and `boot.ci` with an argument to specify the type of bootstrap confidence interval.

```
# BCa Bootstrap CI for median
library(boot)

median_stat <- function(data, indices) {
  return(median(data[indices]))
}

set.seed(123)
data <- rgamma(30, shape=2, rate=0.5)
boot_result <- boot(data, median_stat, R=2000)

# BCa CI
bca_ci <- boot.ci(boot_result, type="bca")
bca_ci
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 2000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_result, type = "bca")

Intervals :
Level       BCa
95%   ( 1.954,  4.004 )
Calculations and Intervals on Original Scale
```

```
# Compare different methods
boot.ci(boot_result, type=c("norm", "perc", "bca"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 2000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_result, type = c("norm", "perc", "bca"))

Intervals :
Level      Normal             Percentile            BCa
95%   ( 1.904,  3.662 )   ( 2.057,  4.004 )   ( 1.954,  4.004 )
Calculations and Intervals on Original Scale
```

Studentized (Bootstrap-t) CI

The Studentized bootstrap-t method (also called bootstrap-t) is a bootstrap approach for constructing confidence intervals that accounts for the variability in the estimate's standard error across bootstrap samples.

Mathematical Algorithm

- For each bootstrap sample $b = 1, \ldots, B$:
  - Compute $\hat{\theta}_b^* = T(X_b^*)$
  - Estimate standard error $\widehat{se}_b^*$ using a second-level bootstrap
  - Compute: $t_b^* = \frac{\hat{\theta}_b^* - \hat{\theta}}{\widehat{se}_b^*}$
- Sort t_b^* values: $t_{(1)}^* \leq t_{(2)}^* \leq \ldots \leq t_{(B)}^*$
- Confidence interval:

$$\left[\hat{\theta} - t_{(1-\alpha/2)}^* \cdot \widehat{se}, \hat{\theta} - t_{(\alpha/2)}^* \cdot \widehat{se}\right]$$

Pseudo-code

```
function studentized_ci(data, statistic, B=1000, B2=100, alpha=0.05):
    theta_hat = statistic(data)
    n = length(data)
    t_star = numeric(B)

    for b in 1:B:
        # First-level bootstrap
        sample_b = sample(data, n, replace=TRUE)
        theta_b = statistic(sample_b)

        # Second-level bootstrap for SE
        theta_b2 = numeric(B2)
        for b2 in 1:B2:
            sample_b2 = sample(sample_b, n, replace=TRUE)
            theta_b2[b2] = statistic(sample_b2)
        se_b = sd(theta_b2)

        t_star[b] = (theta_b - theta_hat) / se_b

    # Estimate SE of original statistic
    se_hat = sd(replicate(B2, statistic(sample(data, n, replace=TRUE)))
)

    # Get critical values
    t_crit = quantile(t_star, probs=c(1-alpha/2, alpha/2))

    lower = theta_hat - t_crit[1] * se_hat
    upper = theta_hat - t_crit[2] * se_hat

    return (lower, upper)
```

An R Example

In the following example with simulated data, we also use the built-in R functions `boot()` and `boot.ci` with an argument to specify the type of bootstrap confidence interval.

```r
# Studentized Bootstrap CI
#library(boot)

# Statistic function with variance estimate
mean_var_stat <- function(data, indices) {
  sample_data <- data[indices]
  mean_val <- mean(sample_data)

  # Internal bootstrap for variance
  n_inner <- length(sample_data)
  inner_boot <- replicate(200, {
    mean(sample(sample_data, n_inner, replace=TRUE))
  })
  var_val <- var(inner_boot)

  return(c(mean_val, var_val))
}

set.seed(123)
data <- rlnorm(20, meanlog=2, sdlog=1)
boot_result <- boot(data, mean_var_stat, R=1000)
boot.ci(boot_result, type="stud")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_result, type = "stud")

Intervals :
Level    Studentized
95%   ( 8.40, 24.12 )
Calculations and Intervals on Original Scale
```

Practical Guidelines

Choosing the Right Method

Method	Use Case
Normal CI	When bootstrap distribution is approximately normal
Percentile CI	Simple method, transformation-respecting
BCa CI	More accurate, accounts for bias and skewness
Studentized CI	Good for location statistics, accounts for variance variability
Likelihood-based	When likelihood inference is appropriate

Number of Bootstrap Replicates

- Basic estimates: $B \geq 1000$
- Variance estimation: $B \geq 2000$

- Percentile/BCa intervals: $B \geq 2000 - 5000$
- Studentized intervals: $B \geq 1000$, with $B2 \geq 200$ for inner loop

Common Pitfalls

- **Too few bootstrap samples**: Leads to unstable intervals
- **Small sample sizes**: Bootstrap may not perform well with $n < 20$
- **Heavy dependence in data**: Requires specialized blocking methods
- **Discrete data**: Percentile methods may have poor coverage

R Packages for Bootstrap

- **boot**: Comprehensive bootstrap functions
- **bootstrap**: Basic bootstrap operations
- **resample**: Alternative implementation
- **infer**: Tidyverse approach to resampling

Overiew of Likelihood-Based Bootstrap CI (optional)

In classical parametric inference, we assume a model $f(y|\theta)$ and use the likelihood function $L(\theta)$ to estimate parameters and construct confidence intervals (e.g., using the likelihood ratio test and Wilks' theorem). However, when the sample size is small or the model is complex, asymptotic likelihood theory may not be reliable.

Likelihood-based bootstrap confidence intervals combine classical likelihood inference with bootstrap resampling to produce more accurate intervals, especially in small samples or non-standard situations.

This is a relatively more advanced topic. We will not discuss it in detail. Instead, we will provide an overview of two likelihood-based confidence intervals and outline the related algorithms. If you are interested in learning more, you can translate the algorithms into R code to implement them and compare the resulting confidence intervals with other types of confidence intervals you have learned previously.

Percentile Likelihood Bootstrap

The percentile likelihood bootstrap merges two approaches:

- **Likelihood principle**: Use the shape of the likelihood function to define plausibility of parameters.
- **Bootstrap principle**: Resample data to approximate the sampling distribution of estimates without relying solely on asymptotics.

The method can yield more accurate confidence intervals in finite samples while respecting the likelihood. The brief steps for implementing this bootstrap procedure is summarized in the following algorithm.

Algorithm

Step 1: Resample data with replacement

Step 2: For each bootstrap sample, let $\hat{\theta}^*$ be the MLE of θ. compute the likelihood ratio statistic:

$$\Lambda^*(\theta) = 2\left[\ell\left(\hat{\theta}^*\right) - \ell(\theta)\right]$$

Step 3: The percentile bootstrap confidence interval is:

$$\{\theta: \Lambda^*(\theta) \leq \chi^2_{1-\alpha}(1)\}$$

Bootstrap Calibrated Likelihood Ratio Intervals

These intervals improve upon standard Likelihood Ratio (LR) intervals by correcting their coverage properties through calibration, ensuring the nominal confidence level (e.g., 95%) is closer to the true coverage probability.

For a parameter θ, the standard LR statistic is:

$$W(\theta) = 2\left[\ell\left(\hat{\theta}\right) - \ell(\theta)\right]$$

where $\hat{\theta}$ is the MLE.

Asymptotically: $W(\theta) \sim \chi^2_1$ under $H_0: \theta = \theta_0$.

The standard $(1-\alpha)$ LR interval is:

$$\left\{\theta: W(\theta) \leq \chi^2_{1,1-\alpha}\right\}$$

The problem is that this χ^2 approximation may be poor with small samples and high dimensional inferences.

Algorithm

Step 1: Compute observed LR statistic $W_{\text{obs}}(\theta)$ for each θ

Step 2: Bootstrap resampling:

- Fit model, get $\hat{\theta}$
- Generate datasets from $F_{\hat{\theta}}$
- Recompute W^* for each

Step 3: Estimate coverage function:

$$\pi(c) = P^*(W^* \leq c)$$

Step 4: Find calibrated threshold c_α where

$$\pi(c_\alpha) = 1 - \alpha$$

Step 5: Invert test: Include θ if

$$W_{\text{obs}}(\theta) \leq c_\alpha$$