IX1501 HT24 Project 2

Sophie Malmberg & Adalet Adiljan

October 2024

Bootstrapping for an estimation of probability

sophiema@kth.se
adalat@kth.se

1 Introduction

The goal of this project was to estimate a probability given a sample of "10 independent and identically distributed (i.i.d.) random variables "[1]. The underlying distribution of these variables was unknown, as was their true mean value μ. The main objective was to estimate the probability that the sample mean, adjusted by the unknown mean, falls within a given range. To achieve this, we used bootstrapping [1], a resampling technique that enabled us to make inferences about the population from the sample data.

The project consisted of two tasks. The first task was to "explain how bootstrapping method can be used to estimate p" [1], including writing pseudo code describing the solution. The second task consisted of estimating the probability p by writing a program using the Python programming language. This report explains how the bootstrap method can be applied to estimate the probability p, using resampling to generate an empirical distribution of the sample mean.

1.1 Bootstrapping

Bootstrapping is a method that can be used in order to approximate distributions given a small amount of sample data, when there is insufficient information about the underlying distribution of the sample data [1]. This is done by resampling the original data and creating new samples containing a random combination of the original data. These samples can then be used to create a confidence interval describing the probability of the original mean value μ [2].

The arithmetic mean is the average of a set of numbers, which is calculated by adding all the values and then dividing by the number of values. For each sample of size n, and values $x_i \in X$, the mean value is defined as the following:

$$\mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Bootstrapping is suitable for cases where the underlying distribution is unknown or when the sample size is too small to reliably apply classical statistical methods [1].

2 Summary of the findings and results

The result shows that the bootstrap method can be used in order to approximate the probability of the sample mean falling within a specified range. The results shows that by generating multiple resamples and analyzing distribution of bootstrap sample means, we can effectively capture the variability inherent in the data and make informed statistical conclusions about the population from which the sample was drawn.

3 Description of the methodology

The primary objective of the tasks was to approximate the unknown value of p given that we don't have enough of knowledge about the given sample. The bootstrapping approach not only allowed us to gather a comprehensive distribution of the means, but also contributed to estimating the confidence interval to estimate the unknown probability p. We were given the samples consisting of 10 observations as follows [1]:

$$56, 101, 78, 67, 93, 87, 64, 72, 80, 69$$

The probability p was defined according to the following formula [1], with $a = -4, n = 10$ and $b = 6$:

$$p = P\left(a < \frac{1}{n}\sum_{i=1}^{n} X_i - \mu < b\right)$$

3.1 Task 1

The first task was solved by using the description of the bootstrap method given in the assignment [1]. While considering the description in the given assignment, a step-by-step process for implementing the bootstrapping method in Python begins with an understanding of the principles outlined in a comprehensive introduction to bootstrap sampling [5]. By following the pseudo code instructions in this given task, it thereof served as a stepping stone for task 2, since the pseudo code clarifies logic and the flow for task 2. The original sample consisting of numerical observations was given. Starting with defining the original sample, this will contain a list of the given numerical observations.

The first step involves to calculate the mean of the original sample which serves as a reference point for comparing the means of the randomized samples. The second step consisted of setting the parameters and the constraints to analyze the sample means, such as the lower constraint 'a' and the upper constraint 'b' to establish the bounds for our analysis. We also set the size of each bootstrap sample `sampleSize` and determine the total number of bootstrap samples to generate `sampleAmount`. Finally, we initialize an empty list to store the means of the randomized samples and a counter to track the number of times each sample mean falls within the specified constraints.

Further, a loop was initialized to generate the specific number of bootstrap samples denoted as `sampleAmount`. Within each iteration of the loop, a random sample is drawn from the original sample with replacement which allows duplicates of the values being chosen. Given the generated random samples from the bootstrap, we are also needed to calculate the mean of the subset of randomly drawn data:

$$\text{randomizedSampleMean} = \frac{1}{\text{sampleSize}} \sum_{j=1}^{\text{sampleSize}} y_j$$

`sampleSize` corresponds to the randomly selected data points in the sample, y_j the individual values in the bootstrapped sample which are finally summarized to compute the average of the random sample. The mean value of each list by the sample with replacement is calculated and appended to the 'results' list.

Given the defined constraint, we then check with a condition if the calculated sample mean falls within the defined constraints $a < (\text{randomizedSampleMean} - \text{mean}) < b$. If this condition is satisfied, the probability counter will be incremented.

3.2 Task 2

The second task was solved by writing a Python program. The program creates a large amount of samples ($n = 1000$), each containing a randomized combination of the given sample values. The mean value $\mu_n$of each randomized sample is calculated and compared to the original mean value μ. If the difference $\mu_n - \mu$ fulfills the given conditions, the value of a probability counter is increased by one. This probability counter value is then divided by the total amount of samples, resulting in the probability p, according to the formula below.

$$\text{probability} = \frac{\text{probabilityCounter}}{\text{sampleAmount}}$$

The bootstrapping code is based on the example code given in the assignment [1]. We also created a histogram (see figure 1) displaying the results of the Python program.

4 Numerical results

The Python program that was written during task 2 approximates the probability as the following:

$$p = P\left(a < \frac{1}{n}\sum_{i=1}^{n} X_i - \mu < b\right) = 0.755$$

This means that 75.5% of the bootstrapped sample means falls within the [-4, 6] interval when it is centered around the original sample mean. Along with printing out the p value, we considered the confidence interval of 95% to capture the middle distribution of sample means that lie with 95% confidence:

$$CI_{95\%} = [CI_{\text{low}}, CI_{\text{high}}] = [68.8, 85.1]$$

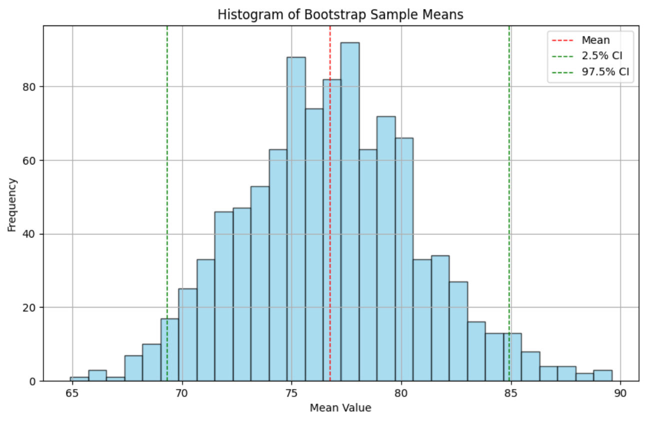

Figure 1: Histogram of Bootstrap Sample Means

The histogram (figure 1) shows the range of mean values which was computed from bootstrap samples on the x-axis, ranging from 65 to 85, where each bar represents the frequency of bootstrap means within specific intervals. The y-axis represents the frequency of which mean is being yield at, where $\mu \approx 77$ is the average mean value. This value can be compared to the mean value $\mu = 76.7$ of the original sample. The dashed red line marks the calculated mean of the entire resampled data, while the two dashed green lines marks the boundaries of the 95 percent confidence interval implying that the true population mean lies within this range.

5 Analysis and discussion

The results showed us that the method provides a powerful and non-parametric approach to estimate the sample means variability. By the given histogram, it highlights how the central tendency is preserved which showcases the reliability of bootstrapping to estimate p. Since we are given a small sample size, it could lead to a reduced precision of the estimation since it causes a broader interval, so therefore, increasing the sample size could narrow this interval more.

After exploring and experimenting with the bootstrapping method, it's clear that while it offers notable advantages for scenarios which the population is unknown, there are certain limitations that may be relevant in real-life scenarios, especially when working with data analysis. For example, if we get to work with sample size of 10, it may lead to overgeneralizing the true mean. In practical situations like large-scale applications, a more larger sample size could better capture any hidden or underlying population characteristics.

Another example is if the data is somehow correlated in a important way such as in financial data where bootstrapping may not be the best method to capture the true mean since it requires the observations to be independent from each other. Furthermore, working with small samples [3] could lead to biased estimates when for example working with rare events, since it way fail to capture the range of variability. Since bootstrapping has practical applications in the real world, we came to realize that its effectiveness is highly relied upon how representative the data actually is of the real distribution.

Compared to other statistical methods, the bootstrapping approach is advantageous because of its non-parametric nature. It doesn't rely on the assumption of normality which makes it ideal for situations where traditional parametric methods like t-testing [6] might fail due to non-normal data distribution. Thanks to its flexibility, it is often applicable in real world scenarios [4].

6 Code

```
# TASK 1

# Define the original sample data as array:
sample = arr[ list of observations ]

# set the parameters:

Integer a = lower constraint
Integer b = upper constraint

Integer sampleSize = length(sample)
Integer sampleAmount = number of samples to randomize
result = arr[]# Creating an empty list to store bootstrap sample means
probabilityCounter = 0 #Counter to increment everytime the condition is fulfilled

#Calculate the mean of the original sample:
mean = calculateMean(sample)

FOR i = 1 to sampleAmount DO
    #Generate a random sample with replacement
    randomizedSample = randomChoice (sample, sampleSize)

    #Calculate the mean of the random sample:
    randomizedSampleMean = calculateMean(randomizedSample)

    #Add randomizedSampleMean to result:
    result.append(randomizedSampleMean)

    IF a < (randomizedSampleMean - mean) < b THEN
        probabilityCounter = probabilityCounter + 1

    #Calculate the confidence interval using the percentiles from the result
    confidenceInterval = calculatePercentiles(result, arr[2.5, 97,5])

    #Calculate the probability of fulfilling the given condition:
    probability = probabilityCounter / sampleAmount

------------------------------------------------------------
# TASK 2
import numpy as np
from numpy import random
import matplotlib.pyplot as plt

a = -4  # the lower constraint
b = 6   # the upper constraint
sample = [56, 101, 78, 67, 93, 87, 64, 72, 80, 69]  # the original sample observations
mean = np.mean(sample)  # the mean of the original sample
sampleSize = 10  # the size of each sample
sampleAmount = 1000  # the number of randomized samples to generate
result = []  # list to store the means of the randomized samples
probabilityCounter = 0  # counter for samples that fulfill the given conditions

for i in range(sampleAmount):
    # Generate a random sample with replacement
    randomizedSample = random.choice(sample, sampleSize, replace=True)
    randomizedSampleMean = np.mean(randomizedSample)  # Calculate the mean
    #of the randomized sample
    result.append(randomizedSampleMean)  # Store the mean in the result list
    # Check if the condition is satisfied
    if a < (randomizedSampleMean - mean) < b:
        probabilityCounter += 1

# Calculate the confidence interval using np.percentile
confidenceInterval = np.percentile(result, [2.5, 97.5])

# Calculate the probability of fulfilling the given conditions
probability = probabilityCounter / sampleAmount

#Calculate the mean of the result
mean = np.mean(result)

# Create a histogram for the bootstrap sample means
plt.figure(figsize=(10, 6))
plt.hist(result, bins=30, color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Histogram of Bootstrap Sample Means')
plt.xlabel('Mean Value')
plt.ylabel('Frequency')
plt.axvline(np.mean(result), color='red', linestyle='dashed',
linewidth=1, label='Mean')
plt.axvline(confidenceInterval[0], color='green', linestyle='dashed',
linewidth=1, label='2.5% CI')
plt.axvline(confidenceInterval[1],
color='green', linestyle='dashed',
linewidth=1, label='97.5% CI')
plt.legend()
plt.grid()
plt.show()

#Prints the result
print("Confidence interval: ", confidenceInterval)
print("Probability: ", probability)
print("Mean: ", mean)
```

7 References

[1]: IX1501 HT24 Project, Canvas.kth.se, 2024. https://canvas.kth.se/courses/49301/pages/ projektuppgift-2
(Accessed Oct. 4, 2024).

[2]: 11.2.1 - Bootstrapping Methods, Pennsylvania State University. https://online.stat.psu.edu/stat500/lesson/11/11.2/11.2.1
(Accessed Oct. 4, 2024).

[3]: Limitations of Bootstrapping Method. https://frmi.netlify.app/quantitative_analysis/13_simulation_and_bootstrapping/7_limitations_of_bootstrapping_method
(Accessed Oct.6, 2024).

[4]: Advantages of Bootstrapping Statistics, Learn Statistics Easily. https://statisticseasily.com/glossario/what-is-bootstrapping-statistical-overview/
(Accessed Oct.6, 2024)

[5]: Introduction to Bootstrap Sampling in Python, AskPython. https://www.askpython.com/python/examples/bootstrap-sampling-introduction
(Accessed Oct.8, 2024)

[6]: An Introduction to t Tests, Scribbr. https://www.scribbr.com/statistics/t-test/
(Accessed Oct.8, 2024)