Project 5 - Confidence Intervals

EE 381 - Probability and Statistics with Applications to Computing

Brian Nguyen
015188925
April 19, 2019

Contents

1 Effect of Sample Size on Confidence Interval

1.1 Introduction

In this project, we are exploring the relation of $\overline{X}$ to the population mean μ. The scenario is taking a small sample of bearings (size n) and calculating the sample mean $\overline{X}$ and the sample standard deviation $\hat{S}$. We are creating two graphs with sample means: one with 95 percent confidence intervals and another with 99 percent confidence intervals. The following parameters are provided:

- Total number of bearings: $N = 1,500,000$
- Population mean: $\mu = 55$ grams
- Population standard deviation: $\sigma = 5$grams
- Sample sizes: $n = 1, 2, ..., 200$

In this case, we are using a Normal distribution. The 95% and the 99% confidence intervals are defined as $\mu \pm 1.96\frac{\sigma}{\sqrt{n}}$ and $\mu \pm 2.58\frac{\sigma}{\sqrt{n}}$ respectively.

1.2 Methodology

For each of the confidence intervals, I created two methods of the upper and lower bounds:

```
def n95_upper(n, mu, sig):
    return mu + 1.96 * (sig / np.sqrt(n))

def n95_lower(n, mu, sig):
    return mu - 1.96 * (sig / np.sqrt(n))

def n99_upper(n, mu, sig):
    return mu + 2.58 * (sig / np.sqrt(n))

def n99_lower(n, mu, sig):
    return mu - 2.58 * (sig / np.sqrt(n))
```

In order to plot these values, I created three lists that stores different values:

- `X_bar_list`: stores values of $\overline{X}$
- `upper_list`: stores upper bound values
- `lower_list`: stores lower bound values

In a `for` loop from 0 to `n`, I generated values for $\overline{X}$ by using the following methods:

- `X = B[random.sample(range(N), i)]`
- `X_bar = np.mean(X)`

where `B = np.random.normal(mu_x_gram, sig_x_gram, N)`. In addition, I generated the upper and lower bound functions by using the functions in the previous page. I stored them lists `upper_list` and `lower_list`.

In order to plot the values of $\overline{X}$, I used the range of `n` and the values stored in `X_bar_list` to generate a scatter plot with the function `plt.scatter(b, X_bar_list)`, where `b` is the range from 0 to n. The values of `upper_list` and `lower_list` are also used to plot the upper bound and lower bound graphs.

1.3 Results and Conclusion

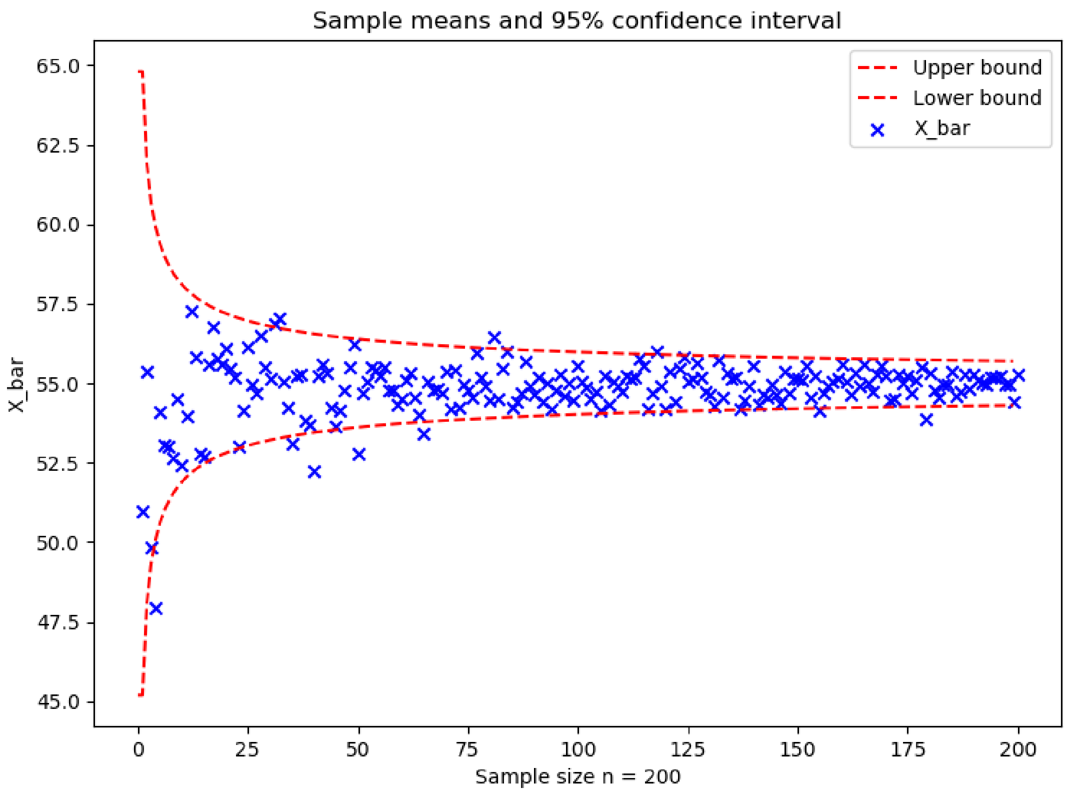

Figure 1: 95% confidence interval with a sample size of $n = 200$

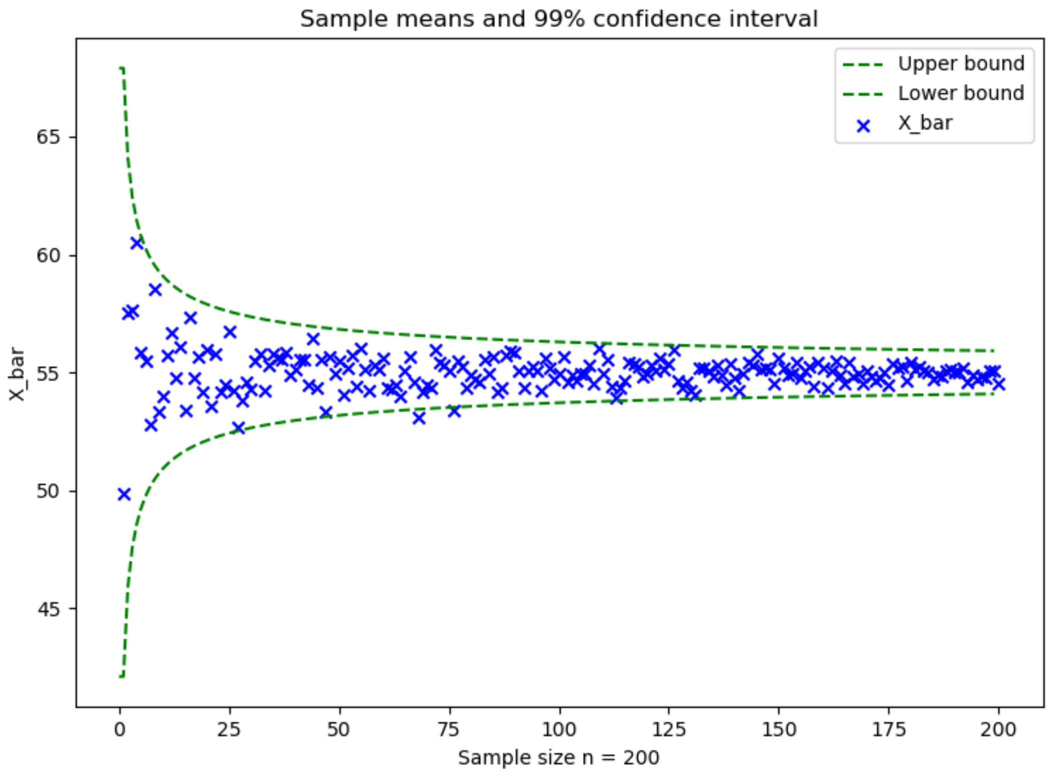

Figure 2: 99% confidence interval with a sample size of $n = 200$

1.4 Appendix

```
"""
Name:     Brian Nguyen
Class:    EE 381 - Probability & Stats
Start:    4/9/19
End:      4/19/19
"""


import numpy as np
import random
import matplotlib.pyplot as plt

N = 1_500_000   # population size
mu_x_gram = 55  # mean
sig_x_gram = 5  # standard deviation
```

```
def n95_upper(n, mu, sig):
    return mu + 1.96 * (sig / np.sqrt(n))


def n95_lower(n, mu, sig):
    return mu - 1.96 * (sig / np.sqrt(n))


def n99_upper(n, mu, sig):
    return mu + 2.58 * (sig / np.sqrt(n))


def n99_lower(n, mu, sig):
    return mu - 2.58 * (sig / np.sqrt(n))


def confidence(n):

    """
    :param n: size of sample
    :return: graphs depicting 95% and 99% confidence intervals
    """

    # 95% CONFIDENCE INTERVAL
    X_bar_list = []     # list to store values of X bar
    upper_list = []     # list to store values of upper bound graph
    lower_list = []     # list to store values
    B = np.random.normal(mu_x_gram, sig_x_gram, N)
    for i in range(n):
        if i == 0:
            i += 1
        X = B[random.sample(range(N), i)]
        X_bar = np.mean(X)
        X_bar_list.append(X_bar)    # store value of X_bar into list
        f1 = n95_upper(i, mu_x_gram, sig_x_gram)    # compute value for 95% c
        f2 = n95_lower(i, mu_x_gram, sig_x_gram)    # compute value for 95% c
        upper_list.append(f1)   # store upper limit value into list
        lower_list.append(f2)   # store lower limit value into list
    b = list(range(1, n + 1))
    plt.scatter(b, X_bar_list, color='b', marker='x', label='X_bar')
    plt.title('Sample means and 95% confidence interval')
    x_label = 'Sample size n = %d' % n
    plt.xlabel(x_label)
    plt.ylabel('X_bar')
    plt.plot(upper_list, 'r--', label='Upper bound')
```

```
    plt.plot(lower_list, 'r--', label='Lower bound')
    plt.legend()
    plt.show()
    # ---------------------------------------------------------------
    # 99% CONFIDENCE INTERVAL
    X_bar_list = []  # list to store values of X bar
    upper_list = []  # list to store values of upper bound graph
    lower_list = []  # list to store values
    for i in range(n):
        if i == 0:
            i += 1
        X = B[random.sample(range(N), i)]
        X_bar = np.mean(X)
        X_bar_list.append(X_bar)    # store value of X_bar into list
        f1 = n99_upper(i, mu_x_gram, sig_x_gram)    # compute value for 95% c
        f2 = n99_lower(i, mu_x_gram, sig_x_gram)    # compute value for 95% c
        upper_list.append(f1)      # store upper limit value into list
        lower_list.append(f2)      # store lower limit value into list
    b = list(range(1, n + 1))
    plt.scatter(b, X_bar_list, color='b', marker='x', label='X_bar')
    plt.title('Sample means and 99% confidence interval')
    x_label = 'Sample size n = %d' % n
    plt.xlabel(x_label)
    plt.ylabel('X_bar')
    plt.plot(upper_list, 'g--', label='Upper bound')
    plt.plot(lower_list, 'g--', label='Lower bound')
    plt.legend()
    plt.show()


def main():
    n = int(input('Enter size n: '))
    confidence(n)


main()
```

2 Using the Sample Mean to Estimate the Population Mean

2.1 Introduction

In this section, we are using different sizes of n to check the probability of success that the mean μ falls in between the lower and upper bounds of each confidence intervals. It should be noted that depending on the size of n, we would have to either use a Normal distribution for large samples ($n \geq 30$) or a Student's t distribution for small samples ($n < 30$).

In the previous experiment, we used the 95% and 99% confidence intervals $\mu \pm 1.96\frac{\sigma}{\sqrt{n}}$ and $\mu \pm 2.58\frac{\sigma}{\sqrt{n}}$ for large sizes of n; however, in this experiment, we are also dealing with small sizes of n.

The 95% and 99% confidence interval depends on $[-t_c, t_c]$ such that $P\{-t_c < z < t_c\} = 0.95$ and $P\{-t_c < z < t_c\} = 0.99$ respectively. The values $[-t_c, t_c]$ depend on two values: the probability value and the degrees of freedom v.

The 95% confidence interval is defined as $\overline{X} \pm t_{0.975}\frac{\hat{S}}{\sqrt{n}}$ and the 99% confidence interval is defined as $\overline{X} \pm t_{0.995}\frac{\hat{S}}{\sqrt{n}}$

This experiment is repeated for $M = 10,000$ times

2.2 Methodology

For the Normal distributions, I defined two variables `successes_n95` and `successes_n99` to count the number of times the mean μ falls in between the confidence intervals. Inside of the `for` loop from range 0 to $M = 10,000$, I calculated $\overline{X}$ and used that value in four different functions:

```
def n_lower_limit_95(X_bar, S, n):
    return X_bar - 1.96 * (S / np.sqrt(n))


def n_upper_limit_95(X_bar, S, n):
    return X_bar + 1.96 * (S / np.sqrt(n))


def n_lower_limit_99(X_bar, S, n):
    return X_bar - 2.78 * (S / np.sqrt(n))


def n_upper_limit_99(X_bar, S, n):
    return X_bar + 2.78 * (S / np.sqrt(n))
```

The returned values are then assigned to variables `ll_n95, ul_n95, ll_n99`, and `ul_n99` respectively. For each of the confidence intervals, if the mean $\mu = 55$ is between the values, then the experiment is considered a success.

A similar approach was used for the Student t-distributions. Inside of the `for` loop, I calculated $\overline{X}$ using the functions

```
def t_lower_limit(X_bar, S, n, t):
    return X_bar - t * (S / np.sqrt(n))


def t_upper_limit(X_bar, S, n, t):
    return X_bar + t * (S / np.sqrt(n))
```

The value of t is dependent on the value of n. To determine its value, we would need to look at the chart that lists the percentile values for the Student's t Distribution with v degrees of freedom. The t_c values for $n = 5, 40, 120$ are:

- $n = 5$
  - $t_{.975} = 2.78$
  - $t_{.995} = 4.60$
- $n = 40$
  - $t_{.975} = 2.02$
  - $t_{.995} = 2.71$
- $n = 120$
  - $t_{.975} = 1.98$
  - $t_{.995} = 2.62$

Just like the previous example, the returned values are assigned to variables `ll_n95, ul_n95, ll_n99`, and `ul_n99`. For each of the confidence intervals, if the mean $\mu = 55$ is between the values, then the experiment is considered a success.

2.3 Results and Conclusion

Sample size (n)	95% Confidence Interval (Using Normal Distribution)	99% Confidence Interval (Using Normal Distribution)	95% Confidence Interval (Using Student t's Distribution)	99% Confidence Interval (Using Student t's Distribution)
5	87.33%	90.93%	95.16%	99.10%
40	94.41%	96.75%	95.03%	99.04%
120	95.08%	99.40%	95.07%	99.19%

Tabe 1: Success rate (percentage) for different sample sizes

2.4 Appendix

2.4.1 Normal Distribution

```
"""
Name:    Brian Nguyen
Class:   EE 381 - Probability & Stats
Start:   4/19/19
End:     4/19/19
"""

import numpy as np
import random


N = 1_500_000    # population size
mu_x_gram = 55   # mean
sig_x_gram = 5   # standard deviation


def n_distribution(n):
    successes_n95 = 0
    successes_n99 = 0
    M = 10_000
    B = np.random.normal(mu_x_gram, sig_x_gram, N)
    for i in range(M):
        X = B[random.sample(range(N), n)]
        S = np.std(X, ddof=1)
        X_bar = np.mean(X)

        # compute values for limits
        ll_n95 = n_lower_limit_95(X_bar, S, n)
        ul_n95 = n_upper_limit_95(X_bar, S, n)
        ll_n99 = n_lower_limit_99(X_bar, S, n)
        ul_n99 = n_upper_limit_99(X_bar, S, n)
        if mu_x_gram >= ll_n95 and mu_x_gram <= ul_n95:
            successes_n95 += 1
        if mu_x_gram >= ll_n99 and mu_x_gram <= ul_n99:
            successes_n99 += 1
    successes_n95 = successes_n95 / M * 100
    successes_n99 = successes_n99 / M * 100
    print("Normal 95% confidence at n =", n, ":", successes_n95)
    print("Normal 99% confidence at n =", n, ":", successes_n99)
```

```
def n_lower_limit_95(X_bar, S, n):
    return X_bar - 1.96 * (S / np.sqrt(n))


def n_upper_limit_95(X_bar, S, n):
    return X_bar + 1.96 * (S / np.sqrt(n))


def n_lower_limit_99(X_bar, S, n):
    return X_bar - 2.78 * (S / np.sqrt(n))


def n_upper_limit_99(X_bar, S, n):
    return X_bar + 2.78 * (S / np.sqrt(n))


def main():
    n_distribution(5)
    n_distribution(40)
    n_distribution(120)

main()
```

2.4.2 Student's t Distribution

```
"""
Name:    Brian Nguyen
Class:   EE 381 - Probability & Stats
Start:   4/19/19
End:     4/19/19
"""


import numpy as np
import random

N = 1_500_000   # population size
mu_x_gram = 55  # mean
sig_x_gram = 5  # standard deviation

def t_distribution(n):
    successes_t95 = 0
    successes_t99 = 0
    M = 10_000
    B = np.random.normal(mu_x_gram, sig_x_gram, N)
    for i in range(M):
        # all t values from the student's t chart
        if n == 5:
            t_95 = 2.78
            t_99 = 4.60
        elif n == 40:
            t_95 = 2.02
            t_99 = 2.71
        elif n == 120:
            t_95 = 1.98
            t_99 = 2.62
        else:
            break
        X = B[random.sample(range(N), n)]
        S = np.std(X, ddof=1)
        X_bar = np.mean(X)

        # compute values for limits
        ll_t95 = t_lower_limit(X_bar, S, n, t_95)
        ul_t95 = t_upper_limit(X_bar, S, n, t_95)
        ll_t99 = t_lower_limit(X_bar, S, n, t_99)
        ul_t99 = t_upper_limit(X_bar, S, n, t_99)
        if mu_x_gram >= ll_t95 and mu_x_gram <= ul_t95:
```

```
            successes_t95 += 1
        if mu_x_gram >= ll_t99 and mu_x_gram <= ul_t99:
            successes_t99 += 1
    successes_t95 = successes_t95 / M * 100
    successes_t99 = successes_t99 / M * 100
    print("Student t 95% confidence at n =", n, ":", successes_t95)
    print("Student t 99% confidence at n =", n, ":", successes_t99)


def t_lower_limit(X_bar, S, n, t):
    return X_bar - t * (S / np.sqrt(n))


def t_upper_limit(X_bar, S, n, t):
    return X_bar + t * (S / np.sqrt(n))


def main():
    t_distribution(5)
    t_distribution(40)
    t_distribution(120)


main()
```