Department of Electrical Engineering

California State University, Long Beach

Probability and Statistics with Applications to Computing

Duc Tran

Confidence Intervals

Submitted by

Devin Suy

2020 December 18

Introduction

The purpose of this laboratory was to utilize the Python scripting language along with a few additional numerical libraries to perform two different probability simulations. These simulations modeled the normal distribution, and the student's t-distribution in showcasing the relationship between sample size and confidence intervals. The simulations conducted were: modeling a normally distributed barrel of a million ball bearings and comparing sample means to a given population mean; estimating the population mean using the sample mean, comparing and contrasting between the normal distribution and student's t-distribution.

Methods

In this first experiment the following equation is implemented in Python:

$$\bar{X}_n = \frac{X_1 + X_2 + \cdots + X_n}{n}$$

Representing the sample mean, this equation is implemented using *statistics.mean()*, passing our generated normally distributed data from *np.random.normal()* for the given population parameters *mu=100* and *sigma=12* for our barrel of *N=1000000* ball bearings. Samples of size *n* are randomly taken using *random.sample()* for values of n from 1 to 200.

An ordered data dictionary is used to map each sample size *n* value, to its corresponding sample mean. This data dictionary is passed to *matplotlib.pyplot.scatter()* to plot our values, along with the 95 percent and 99 percent confidence intervals.

The following experiment uses the same methods to generate the normally distributed population, and is similar to the previous but with the addition of implementation for the following equation:

$$\sigma_n = \frac{\hat{S}}{\sqrt{n}}, \text{ where } \hat{S} = \left\{ \frac{(X_1 - \bar{X})^2 + (X_2 - \bar{X})^2 + \cdots + (X_n - \bar{X})^2}{n-1} \right\}^{1/2}$$

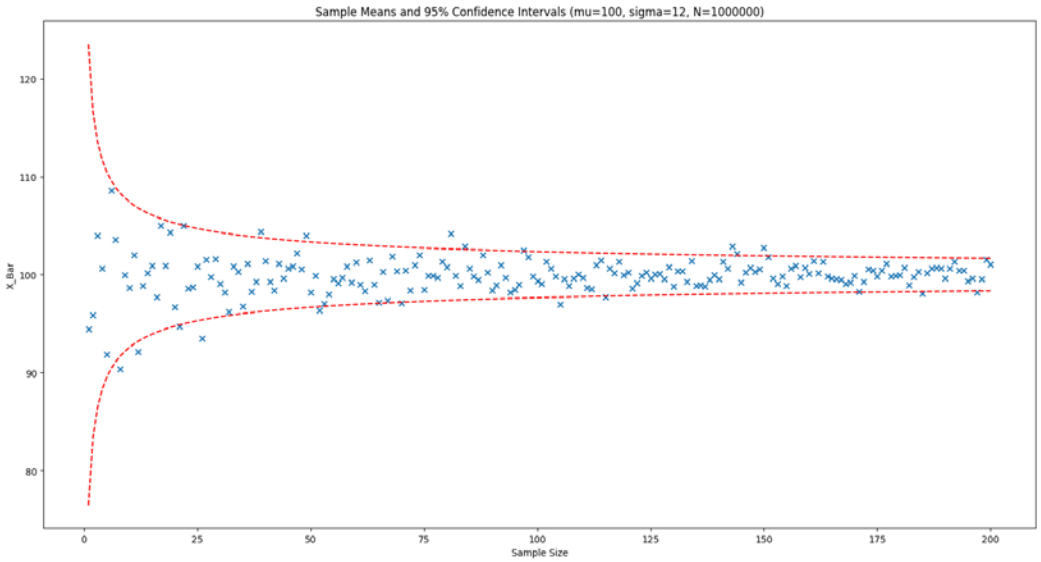
Representing the standard deviation of the sample mean, this is obtained using the *statistics.stdev()* function on our generated population values. Four ordered data dictionaries are created to map each sample size *n* to the amount of successful experiments in which the generated confidence

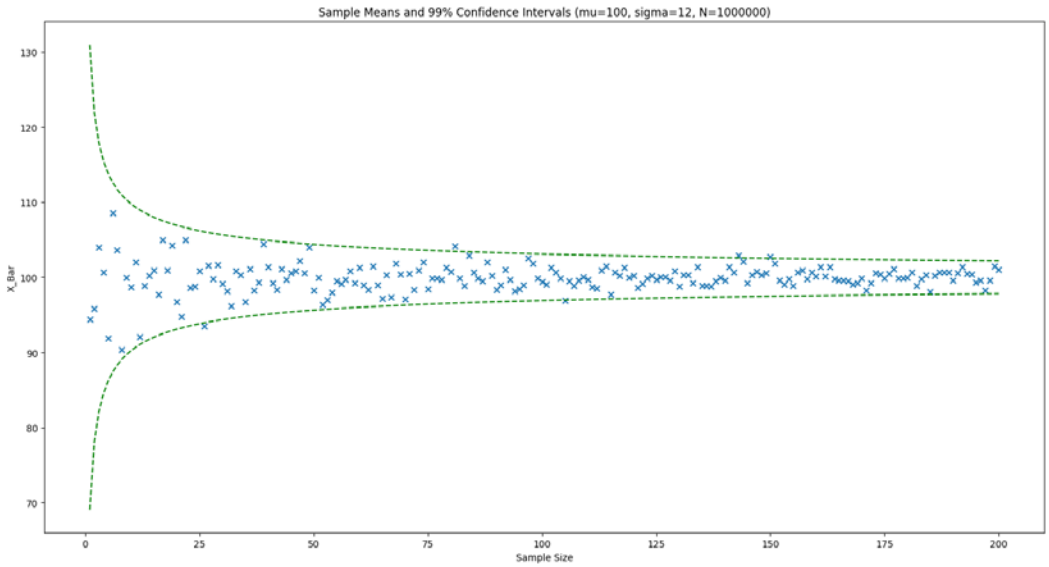
intervals do in fact include the provided population mean. Each trial of the experiment consists of sampling *n* values [5, 40, 120], calculating the sample mean and standard deviation, generating the 95 and 99 percent intervals for the normal distribution and t distributions, then checking if the population mean lies in such interval. Each time this is found to be true, the success count for the trial is updated in the respective data dictionary. The amounts of which are later divided by *num_trials=10000*, outputting the success percentages for each.

Both experiments utilize the utility functions *get_confidence_z()* which generates the confidences intervals given the mean, standard deviation, sample size, and desired confidence for the corresponding normal distribution. Similar implementation is performed in *get_confidence_z()* but for the t distribution.

Results and Discussion

Experiment One:





As shown by the above plots for the first experiment, we can observe two trends. For the 200 trials conducted the calculated population means, represented by the blue crosses, lie within their respective confidence intervals. About 5 percent of the blue crosses are outside of the first interval, and even less in the second plot as expected. Secondly, as sample size increases, we see that there appears to be slightly less values outside the intervals, despite its narrowing as larger sample sizes give an increasingly more accurate approximation of the population mean, particularly in comparison to sample size values < 30.

Experiment Two:

```
RESULTS
-------

Normal distribution, 95% confidence:
   n=5: Success rate 0.877
   n=40: Success rate 0.947
   n=120: Success rate 0.945

Normal distribution, 99% confidence:
   n=5: Success rate 0.939
   n=40: Success rate 0.987
   n=120: Success rate 0.989

T distribution, 95% confidence:
   n=5: Success rate 0.951
   n=40: Success rate 0.953
   n=120: Success rate 0.947

T distribution, 99% confidence:
   n=5: Success rate 0.989
   n=40: Success rate 0.99
   n=120: Success rate 0.99
```

Sample Size (n)	**95% Confidence** (Using Normal Distribution)	**99% Confidence** (Using Normal distribution)	**95% Confidence** (Using Student's t distribution)	**95% Confidence** (Using Student's t distribution)
n=5	0.877	0.939	0.951	0.989
n=40	0.947	0.987	0.953	0.99
n=120	0.945	0.989	0.947	0.99

This second experiment compared the success rates of varying sample sizes between the normal and student's t-distribution. As we have studied, the normal distribution is approached for a sufficiently large sample size *n*, often considered values of 30 or greater. This is further reflected in the data from this simulation as we see the t-distribution has a higher success rate and is more accurate for smaller sample sizes. As an approximation of the normal distribution itself, it is also shown that sample size continues to increase, the normal and t-distributions approximate the same values.

Conclusion

Performing this laboratory was a useful means of being able to simulate as well as compare and contrast the normal and student's t-distribution. We have previously studied the parameters of population and sample means and standard deviations, but it was interesting to be able apply these concepts to actual implementation and observe how they differ as well as how we can approximate the population parameters from a sufficient sample.

There are some limitations to this experiment, clearly for such a large population there will not be an easily obtained mean and standard deviation for the entire population, nor will it be as simple as values 100 and 12. Additionally much of the generation and sampling are dependent on python's implementation of pseudo randomization for our values. This can lead to differences in our results but a sufficiently large number of trials can help mitigate this.

By means of these simulations, we were able to demonstrate the relationship between sample size and the accuracy in approximation of the actual population mean. In practice, the t-distribution should be used for small sample sizes, not the normal distribution as it has better approximations as show in in our trials.

Appendix

Dependencies:

```
from math import sqrt
from collections import OrderedDict, defaultdict
from matplotlib import pyplot as plt
import numpy as np
import random
import statistics
```

Confidence Interval Implementation:

```
# Using normal distr, calculate and return the confidence interval
# given the mean, s_dev, sample size, and desired confidence %
def get_confidence_z(mu, sigma, n, confidence):
    confidence_vals = {.9 : 1.645, .95: 1.96, .9545: 2, .96: 2.05, .98: 2.33, .99
 : 2.58, .9973: 3}
    z_score = confidence_vals[confidence]

    return [
        mu - ((z_score * sigma) / sqrt(n)),
        mu + ((z_score * sigma) / sqrt(n))
    ]

# Using t distr, calculate and return the confidence interval
# given the mean, s_dev, sample size, and desired confidence %
def get_confidence_t(mu, sigma, n, confidence):
    # Values from t table using v=n-1 dof
    studT_95 = {5 : 2.78, 40: 2.02, 120 : 1.98}
    studT_99 = {5 : 4.6, 40: 2.71, 120: 2.62}

    if confidence == .95 and n in [5,40,120]:
        t_score = studT_95[n]
    elif confidence == .99 and n in [5,40,120]:
        t_score = studT_99[n]
    else: return -1

    return [
        mu - ((t_score * sigma) / sqrt(n)),
        mu + ((t_score * sigma) / sqrt(n))
    ]
```

Experiment 1:

```
# Assuming exact population parameters are given, randomly generate population
# of size N and sample values from n = [1:MAX_SAMPLE], calculate population
# means and plot along 95% and 99% confidence intervals
def sample_size_confidence(mu=100, sigma=12, N=1000000, MAX_SAMPLE=200):
    pop = list(np.random.normal(mu, sigma, N))

    # Sample values and map sample size -> to sample_mean, and
    # sample size -> to [interval_95, interval_99] for values n = [1 : 200]
    sample_data = OrderedDict()
    sample_interval_95 = OrderedDict()
    sample_interval_99 = OrderedDict()

    for n in range(1, MAX_SAMPLE+1):
        sample_interval_95[n] = get_confidence_z(mu, sigma, n, 0.95)
        sample_interval_99[n] = get_confidence_z(mu, sigma, n, 0.99)
        sample_data[n] = statistics.mean(random.sample(pop, n))

    # Generate plots, along 95 percent confidence interval, then 99
    plt.scatter(sample_data.keys(), sample_data.values(), marker="x")
    plt.plot(sample_interval_95.keys(), sample_interval_95.values(), 'r', linesty
le='dashed')
    plt.ylabel("X_Bar")
    plt.xlabel("Sample Size")
    plt.title("Sample Means and 95% Confidence Intervals (mu=" + str(mu) + ", sig
ma=" + str(sigma) + ", N=" + str(N) + ")")
    plt.show()
    plt.close()

    plt.scatter(sample_data.keys(), sample_data.values(), marker="x")
    plt.plot(sample_interval_99.keys(), sample_interval_99.values(), 'g', linesty
le='dashed', )
    plt.ylabel("X_Bar")
    plt.xlabel("Sample Size")
    plt.title("Sample Means and 99% Confidence Intervals (mu=" + str(mu) + ", sig
ma=" + str(sigma) + ", N=" + str(N) + ")")
    plt.show()

sample_size_confidence()
```

Experiment 2 Implementation:

```
def normal_studT(mu=100, sigma=12, N=1000000, num_trials=10000, n_vals=[5,40,120]
):
    pop = list(np.random.normal(mu, sigma, N))

    # Map the sample size n to the amount of successful trials
    z_success_95 = OrderedDict([(5, 0), (40,0), (120,0)])
    z_success_99 = OrderedDict([(5, 0), (40,0), (120,0)])
    t_success_95 = OrderedDict([(5, 0), (40,0), (120,0)])
    t_success_99 = OrderedDict([(5, 0), (40,0), (120,0)])

    # Conduct trials, logging the amount of "successes"
    for _ in range(num_trials):
        for n in n_vals:
            # Sample data and calculate parameters
            sample = random.sample(pop, n)
            sample_mean = statistics.mean(sample)
            sample_sdev = statistics.stdev(sample)

            # Calculate intervals [low, high]
            z_95 = get_confidence_z(sample_mean, sample_sdev, n, confidence=.95)
            z_99 = get_confidence_z(sample_mean, sample_sdev, n, confidence=.99)
            t_95 = get_confidence_t(sample_mean, sample_sdev, n, confidence=.95)
            t_99 = get_confidence_t(sample_mean, sample_sdev, n, confidence=.99)

            # Check if trial was success (whether or not the mu falls
            # within intervals), increment count if so
            if (z_95[0] <= mu) and (mu <= z_95[1]): z_success_95[n] += 1
            if (z_99[0] <= mu) and (mu <= z_99[1]): z_success_99[n] += 1
            if (t_95[0] <= mu) and (mu <= t_95[1]): t_success_95[n] += 1
            if (t_99[0] <= mu) and (mu <= t_99[1]): t_success_99[n] += 1

    # Output results of simulation
    print("RESULTS\n-------\n")
    print("Normal distribution, 95% confidence:")
    for n, success_count in z_success_95.items():
        print("   n=" + str(n) + ": Success rate " + str(round(success_count/1000
0, 3)))
    print("\nNormal distribution, 99% confidence:")
    for n, success_count in z_success_99.items():
        print("   n=" + str(n) + ": Success rate " + str(round(success_count/1000
0, 3)))
    print("\nT distribution, 95% confidence:")
    for n, success_count in t_success_95.items():
```

```
        print("   n=" + str(n) + ": Success rate " + str(round(success_count/1000
0, 3)))
    print("\nT distribution, 99% confidence:")
    for n, success_count in t_success_99.items():
        print("   n=" + str(n) + ": Success rate " + str(round(success_count/1000
0, 3)))

normal_studT()
```