ECE 566 - Machine and Deep Learning

Homework #1
Due Date: 09/06/2022

Anas Puthawala

Introduction

This report will demonstrate some of the techniques utilized in an attempt to solve problems 1 and 2 of homework number 1. Any results (if found) will also be represented and explained. Overall, this report will serve as a guidance for the reader to understand the thought process that I underwent to solve the problems. The main ideas tackled in this assignment are 1. Observing the Central Limit Theorem in action and 2. Assessing the outcomes of the Monty Hall Problem

Summary

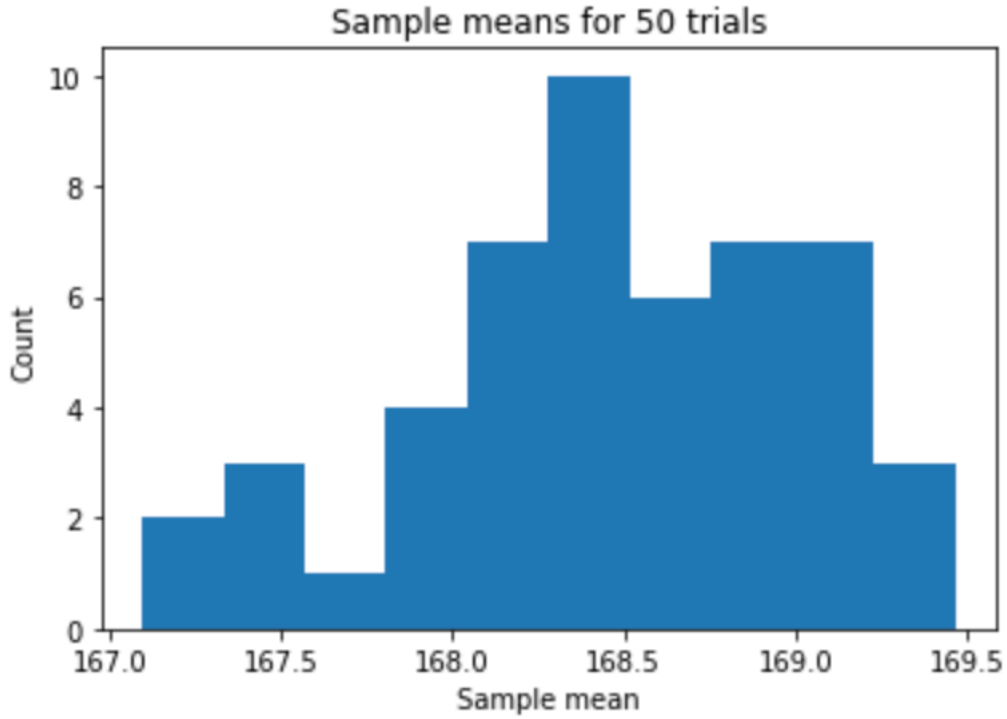
Problem # 1 - Central Limit Theorem
For this problem I simulated the CLT in action by using np.choice with the range between 140cm (4'7) and 198 cm (6'6) and calculating the sample mean for 1000 trials. I would iterate this entire process many times over (specifically n= 5, 10, 50, 100, 10000) and plot a histogram of the outcomes (which are sample means). What is **expected** is a gaussian distribution to be apparent when plotting values as n approaches infinity. That is the entire basis of the central limit theorem, that the sample means of a characteristic or trait will approach a gaussian distribution when sampling values from a population.

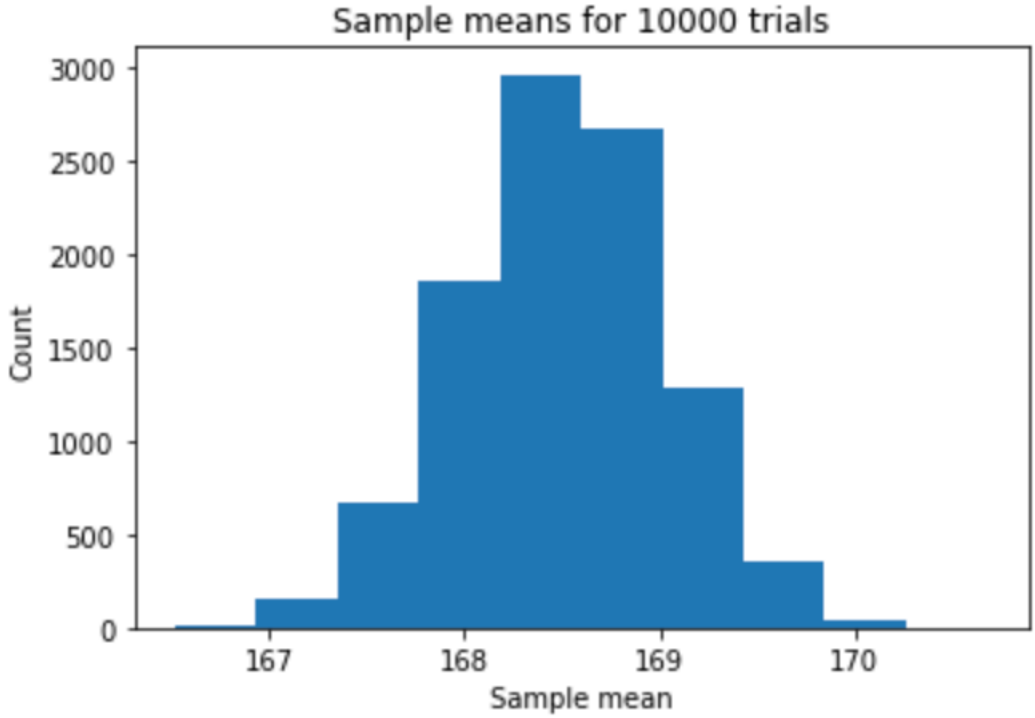
Problem # 2 - Monty Hall Problem
The monty hall problem is such that there's a gameshow with three doors and a car behind one of the three doors and goats behind the other two. Say for example, you pick a door and you don't know what's behind it. Now the gameshow host reveals a goat behind one of the other doors and asks if you want to A. switch your door to the remaining final door or B. keep your door. The objective behind this problem was to analyze the results obtained from simulating the Monty Hall problem with the two strategies (switching your door or not switching your door). It was found that switching the door results in winning the game 2/3 of the time and not switching results in winning the game 1/3 of the time.

Body

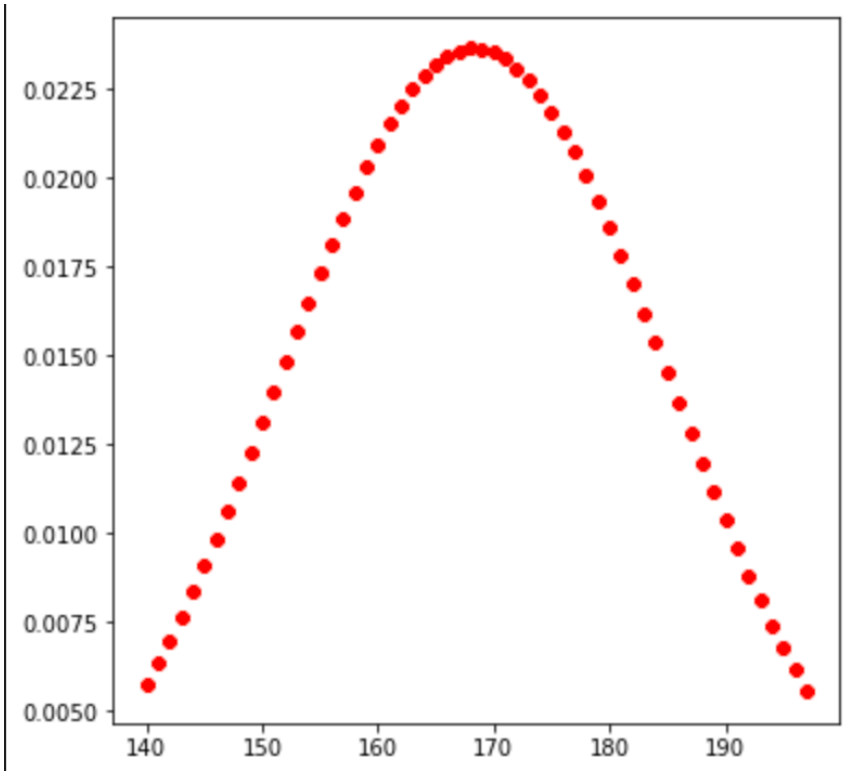
Problem # 1 - Central Limit Theorem
As briefly mentioned in the *Summary,* heights for many humans range usually from 4'7 to 6'6 or 6'7. After converting to centimeters, this height range was utilized to generate values using np.choice
The entire process is as follows:
1. Generate 1000 random values in the range of 140cm to 198cm
2. Calculate the mean of these 1000 values
3. Append this mean to an empty list called sample_means.

Steps 1-3 are **one** iteration, this process was repeated n=5,10,50,100,10000 times and a histogram was plotted using the matplotlib library and the list sample_means. Here are two figures generated from simulating the process 50 times and then 10000 times:





It's evident that as the iterations increase, the sample means begin to form a gaussian distribution. To conclude, I also generated a plot using the PDF of a gaussian distribution and it's evident that the mean is corresponding to what the mean is from the sample in which the iteration was ran 10000 times:



Problem # 2 - Monty Hall Problem

To simulate the Monty Hall Problem I developed two functions.
1. Single_trial
2. Run_trails

For Single_trial the purpose of the function is to simply run a single trial to analyze what the outcome is if I either switch doors or not. So if the switch variable (which must be passed in as a boolean when calling the function) is true then by checking a few if else statements (i.e. what the chosen door is, and what door should be revealed based on the chosen door and the remaining door) we're able to return a specific chosen door. The ultimate output of the function is a true or false, if the chosen_door is equal to 1 or not because in either switching case we make 1 equivalent to the door that has the prize behind it. Lastly, the run trials function simply iterates n number of trials (by calling the single_trial function) and if the outcome of the run_trials is True (i.e. you picked the correct door from single_trial) then simply increment a variable called 'wins' by 1. One final process is simply automating the process to run_trials a bunch of times (i.e. we want to collect data for 100 trials, 500 trials, 1000 trials) so I simply make a list of the amount of times I want to simulate and use a for loop to run the amount of trials and calculate the win percentage based on the variable 'switch' that I choose to set to either True (i.e. switch doors) or False (don't switch doors).

No visuals were needed for the interpretation of results for this problem and ultimately it was found that you would win 33% of the time if you did not switch and 66% of the time you would win if you did indeed switch doors when prompted.

Conclusions

To conclude, there were two main portions to this assignment consisting of assessing & analyzing the properties of the Central Limit Theorem and evaluating strategies for the Monty Hall Problem. Simulations were conducted using heights of humans with varying trial runs in an attempt to visually perceive the effects of the central limit theorem, i.e. witnessing by increasing the trial runs how the sample mean of various trials contort to a gaussian distribution.

As for the Monty Hall Problem there are two strategies:
1. Switching doors or
2. Not switching doors

By following a similar approach as the previous problem (simulations) we are able to ultimately deduce that switching doors when prompted gives you the upper hand and you will win the gameshow if you switch doors approximately 66% of the time whereas if you do not switch, you will win approximately 33% of the time.

References

- https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html
- https://en.wikipedia.org/wiki/Central_limit_theorem
- https://en.wikipedia.org/wiki/Monty_Hall_problem
- https://medium.com/swlh/simulate-the-monty-hall-problem-using-python-7b76b943640e

Appendices

Monty Hall Problem Code

```
def single_trial(switch:bool, ndoors=3):

    # Pick a random door out of the ndoors available
    chosen_door = random.randint(1, ndoors)
    available_doors = []
    #print(chosen_door)

    #case when switching is true
    if switch:
        # Reveal a goat
        if chosen_door == 2:
            revealed_door = 3
        else:
            revealed_door = 2


        # Make the switch by choosing any other door than the
one selected first
        # and the one that was just revealed
        for num in range(1, ndoors+1):
            if num not in (chosen_door, revealed_door):
                available_doors.append(num)

        #randomly choose an available door
        chosen_door = random.choice(available_doors)
```

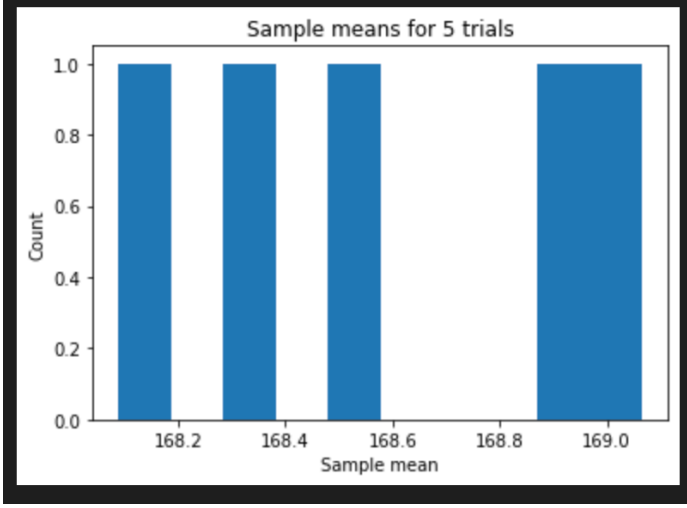
```
    #Case when switching is false
    elif not switch:
        # do not switch
        pass

    # You win if you picked door number 1
    return chosen_door == 1


def run_trials(trials, switch):

    wins = 0
    #Run the trials
    for i in range(trials):
        if single_trial(switch): #Checks to see if you win
            wins += 1 #If you win, increment by 1

    return wins

trials_list = [100, 500, 1000, 10000, 100000, 1000000]
for trials in trials_list:

print('-------------------------------------------------------
-----------------------')
    wins_no_switch = run_trials(trials, switch = False)
    print(f'Percentage of wins in {trials} trials without
switching: {(wins_no_switch/trials)*100}%')
    wins_switch = run_trials(trials, switch = True)
    print(f'Percentage of wins in {trials} trials with
switching: {(wins_switch/trials) * 100}%')

print('-------------------------------------------------------
-----------------------')
```

Central Limit Theorem Code

```
### Experiment:
# Measure the height (in cm) of 1000 randomly selected males
from the population, repeat this 5, 10, 50, 100, 10000 times to
effectively see the CLT in action

# The heights of 1000 men can be simulated using
np.random.choice and we'll choose the range to be from 140cm
(4'7) to 198cm (~6'6)
```

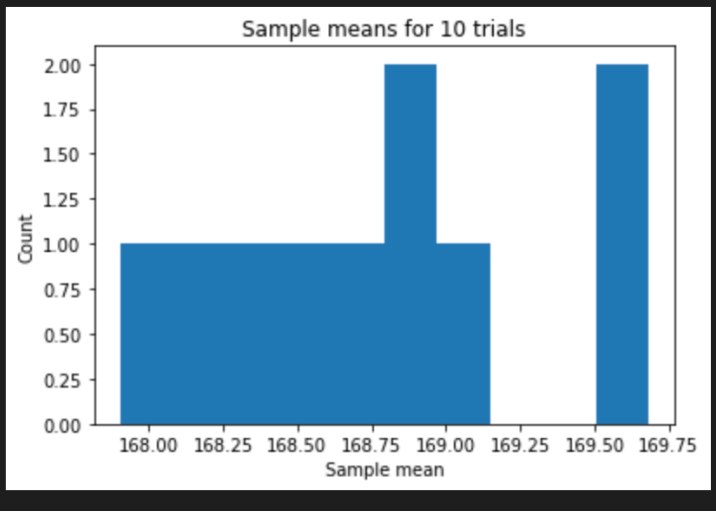
```
def simulate(n):

    sample_means = []
    sample_stddev = []
    sample_variance = []


    for _ in range(n):
        sample = np.random.choice(np.arange(140, 198), 1000)

        sample_means.append(np.mean(sample))


    print(f"The sample means' standard deviation is:
{np.std(sample_means)}\nThe sample means' variance is:
{np.var(sample_means)}\nThe sample means' median is:
{np.median(sample_means)}\nThe sample means' mean is:
{np.mean(sample_means)}")
    plt.hist(sample_means)
    #sample_stddev.append(np.std(sample))
    #sample_variance.append(np.var(sample))
    #plt.plot(sample_stddev)
    #plt.plot(sample_variance)
    plt.title(f'Sample means for {n} trials')
    plt.xlabel('Sample mean')
    plt.ylabel('Count')

simulate(n=5)
```

```
The sample means' standard deviation is: 0.3549234283616642
The sample means' variance is: 0.12597063999999736
The sample means' median is: 168.566
The sample means' mean is: 168.5926
```



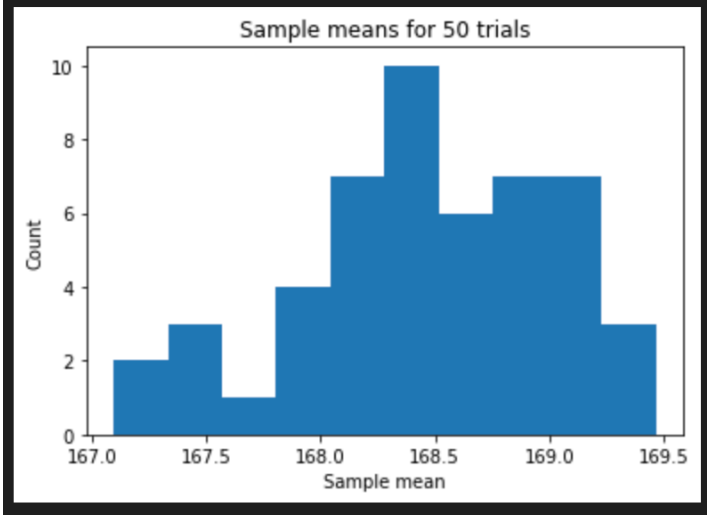
```
simulate(n=10)
```

```
The sample means' standard deviation is: 0.5501269035413507
The sample means' variance is: 0.3026396099999946
The sample means' median is: 168.8255
The sample means' mean is: 168.79829999999998
```



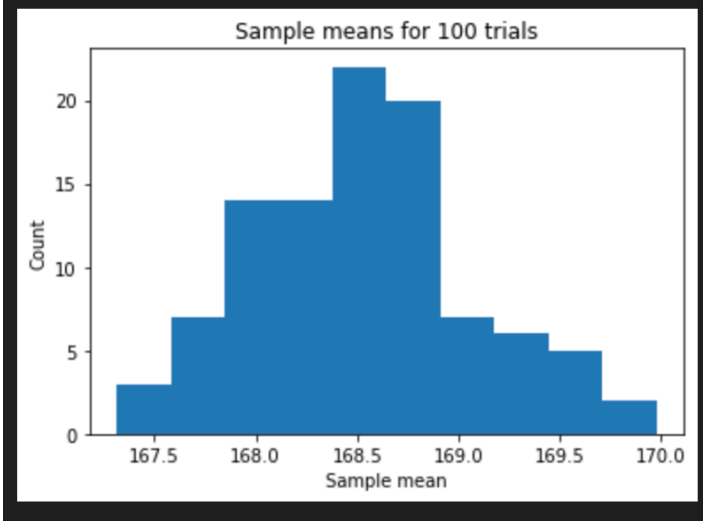

```
simulate(n=50)
```

```
The sample means' standard deviation is: 0.5589819106196543
The sample means' variance is: 0.31246077639999914
The sample means' median is: 168.49849999999998
The sample means' mean is: 168.46705999999998
```



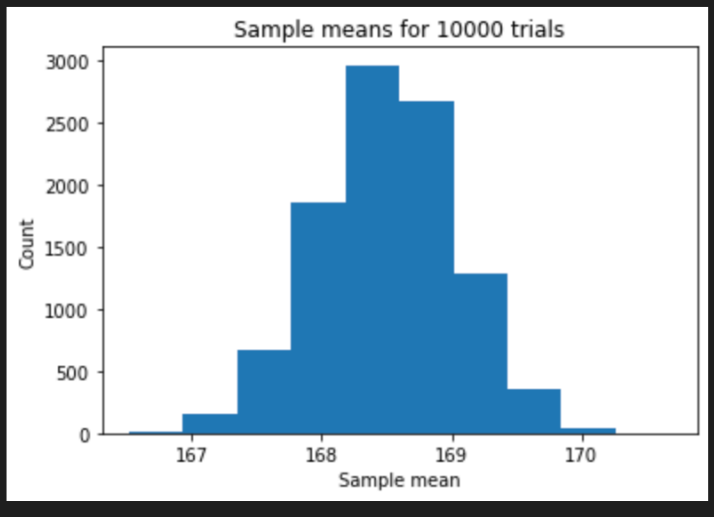
```
simulate(n=100)
```

```
The sample means' standard deviation is: 0.5455043151983305
The sample means' variance is: 0.2975749578999995
The sample means' median is: 168.5095
The sample means' mean is: 168.52911
```



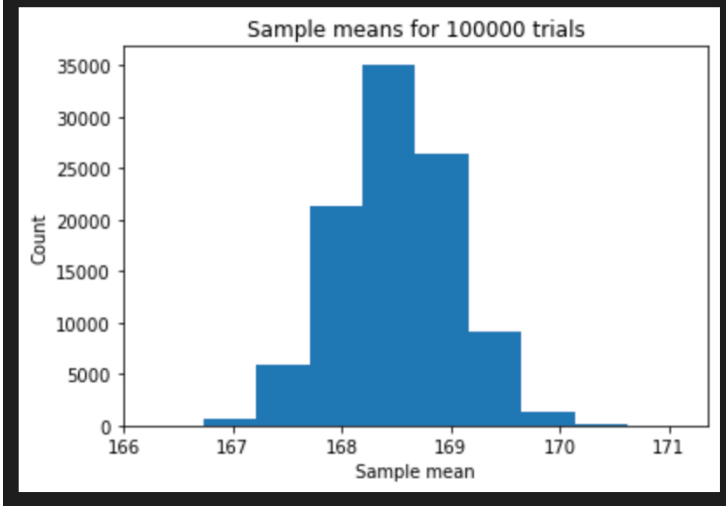

```
simulate(n=10000)
```

```
The sample means' standard deviation is: 0.532855856766687
The sample means' variance is: 0.28393536409056
The sample means' median is: 168.50549999999998
The sample means' mean is: 168.50432879999997
```



```
def theoretical_curve():
    ### Theoretical gaussian curve:
    # To get as close to theoretical gaussian curve we'll just
generate a BUNCH more data and see the results we get.
    print('Theoretical Curve')
    sample_means = []
    for _ in range(100000):
            sample = np.random.choice(np.arange(140, 198), 1000)

            sample_means.append(np.mean(sample))

    print(f"The sample means' standard deviation is:
{np.std(sample_means)}\nThe sample means' variance is:
{np.var(sample_means)}\nThe sample means' median is:
{np.median(sample_means)}\nThe sample means' mean is:
{np.mean(sample_means)}")
    plt.hist(sample_means)
    #sample_stddev.append(np.std(sample))
    #sample_variance.append(np.var(sample))
    #plt.plot(sample_stddev)
    #plt.plot(sample_variance)
    plt.title(f'Sample means for {100000} trials')
    plt.xlabel('Sample mean')
    plt.ylabel('Count')
#Theoretical mean? variance?
### Representation

theoretical_curve()
```

```
Theoretical Curve
The sample means' standard deviation is: 0.5304893998054969
The sample means' variance is: 0.2814190033059964
The sample means' median is: 168.501
The sample means' mean is: 168.49957506
```



```
def pdf(x):
    mean = np.mean(x)
    std = np.std(x)
    y_out = 1/(std * np.sqrt(2 * np.pi)) * np.exp( - (x - 
mean)**2 / (2 * std**2))
    return y_out

# To generate an array of x-values
x = np.random.choice(np.arange(140, 198),1000)

# To generate an array of
# y-values using corresponding x-values
y = pdf(x)

# Plotting the bell-shaped curve
plt.figure(figsize = (6, 6))
#plt.plot(x, y, linestyle = 'dotted')

plt.scatter(x, y, marker = 'o', s = 25, color = 'red')
plt.show()
```