C3_W2_Assignment

November 10, 2024

1 Loaded dice

Welcome to the second assignment in the course Probability and Statistics for Machine Learning and Data Science! In this quiz-like assignment you will test your intuition about the concepts covered in the lectures by taking the example with the dice to the next level.

This assignment can be completed with just pencil and paper, or even your intuition about probability, but in many questions using the skills you're developing as a programmer may help.

1.1 1 - Introduction

You will be presented with 11 questions regarding a several dice games. Sometimes the dice is loaded, sometimes it is not. You will have clear instructions for each exercise.

1.1.1 1.1 How to go through the assignment

In each exercise you there will be a question about throwing some dice that may or may not be loaded. You will have to answer questions about the results of each scenario, such as calculating the expected value of the dice throw or selecting the graph that best represents the distribution of outcomes.

In any case, **you will be able to solve the exercise with one of the following methods:**

- **By hand:** You may make your calculations by hand, using the theory you have developed in the lectures.
- **Using Python:** You may use the empty block of code provided to make computations and simulations, to obtain the result.

After each exercise you will save your solution by running a special code cell and adding your answer. The cells contain a single line of code in the format `utils.exercise_1()` which will launch the interface in which you can save your answer. **You will save your responses to each exercise as you go, but you won't submit all your responses for grading until you submit this assignment at the end.**

Let's go over an example! Before, let's import the necessary libraries.

1.2 2 - Importing the libraries

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import utils
```

1.3 3 - A worked example on how to complete this assignment.

Now let's go over one example question, so you understand how to go through the assignment.

1.3.1 3.1 Example question

Question: Given a 6-sided fair dice, you throw it two times and save the result. What is the probability that the sum of your two throws is greater than 5? (Give your result with 1 decimal place).

After the question, you will see the following block of code.

[5]:
```
# You can use this cell for your calculations (not graded)
0.7
```

[5]:
```
0.7
```

You may use it as you wish to solve the exercise. Or you can just ignore it and use pencil and pen to solve. It is up to you! **You will only save your final answer**.

1.3.2 3.2 Solving using simulations in Python

Let's solve this question in both ways. First, using Python. You may check the ungraded lab Dice Simulations that appears right before this assignment to help you simulate dice throws. Remember that, to get a good approximation, you need to simulate it a lot of times! You will see why this is true in the following weeks, but this is quite intuitive.

[6]:
```
# You can use this cell for your calculations (not graded)

# This list represents each dice side
dice = [1,2,3,4,5,6]

# The idea is to randomly choose one element from this list three times and sum␣
 ↪them.
# Each time we choose, it is as if we had thrown a dice and the side is the␣
 ↪chosen number.
# This list will store the sum for each iteration. The idea is to repeat this␣
 ↪experiment several times.
sum_results = []

number_iterations = 1000
```

```
# Setting a random seed just for reproducibility
np.random.seed(42)
# It will play this game number_iteration times
for i in range(number_iterations):
    # Throw the first dice
    throw_1 = np.random.choice(dice)
    # Throw the second dice
    throw_2 = np.random.choice(dice)
    # Sum the result
    sum_throw = throw_1 + throw_2
    # Append to the sum_result list
    sum_results.append(sum_throw)

# After recording all the sums, the actual probability will be very close to␣
 ↪the proportion among every sum greater than 10 in the sum_results list.
greater_5_count = 0

for x in sum_results:
    if x > 5:
        greater_5_count += 1

probability = greater_5_count/len(sum_results)
print(f"The probability by this simulation is: {probability}")
```

```
The probability by this simulation is: 0.719
```

So the result you would get, rounding in to decimal place, would be 0.7! Let's solve it "by hand".

1.3.3 3.3 Solving using the theory

When throwing two dice, there are 36 possible outcomes:

$$(1,1),(1,2),\ldots,(6,6)$$

You must count how many of them lead to a sum greater than 5. They are:

- If the first throw is 1, there are 2 possibilities for the second throw: 5 or 6.
- If the first throw is 2, there are 3 possibilities for the second throw: 4, 5 or 6.
- If the first throw is 3, there are 4 possibilities for the second throw: 3, 4, 5 or 6.
- If the first throw is 4, there are 5 possibilities for the second throw: 2, 3, 4, 5 or 6.
- If the first throw is 5, there are 6 possibilities for the second throw: 1, 2, 3, 4, 5 or 6.
- If the first throw is 6, there are 6 possibilities for the second throw: 1, 2, 3, 4, 5 or 6.

So, in total there are $2+3+4+5+6+6=26$, possibilities that sum greater than 5.

The probability is then $\frac{26}{36} \approx 0.72$. Rounding it to 1 decimal place, the result is also 0.7!

1.3.4 3.4 Saving your answer

Once you get your answer in hands, it is time to save it. Run the next code below to see what it will look like. You just add your answer as requested and click on "Save your answer!"

[7]:
```
utils.exercise_example()
```

```
FloatText(value=0.0, description='Probability:')
```

```
Button(button_style='success', description='Save your answer!',␣
↪style=ButtonStyle())
```

```
Output()
```

And that's it! Once you save one question, you can go to the next one. If you want to change your solution, just run the code again and input the new solution, it will overwrite the previous one. At the end of the assignment, you will be able to check if you have forgotten to save any question.

Once you finish the assignment, you may submit it as you usually would. Your most recently save answers to each exercise will then be graded.

1.4 4 - Some concept clarifications

During this assignment you will be presented with various scenarios that involve dice. Usually dice can have different numbers of sides and can be either fair or loaded.

- A fair dice has equal probability of landing on every side.
- A loaded dice does not have equal probability of landing on every side. Usually one (or more) sides have a greater probability of showing up than the rest.

Alright, that's all your need to know to complete this assignment. Time to start rolling some dice!

1.5 Exercise 1:

Given a 6-sided fair dice (all of the sides have equal probability of showing up), compute the mean and variance for the probability distribution that models said dice. The next figure shows you a visual represenatation of said distribution:

Submission considerations: - Submit your answers as floating point numbers with three digits after the decimal point - Example: To submit the value of 1/4 enter 0.250

Hints: - You can use np.random.choice to simulate a fair dice. - You can use np.mean and np.var to compute the mean and variance of a numpy array.

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[8]:
```
# Run this cell to submit your answer
utils.exercise_1()
```

```
FloatText(value=0.0, description='Mean:')
```

```
FloatText(value=0.0, description='Variance:')
```

```
Button(button_style='success', description='Save your answer!',␣
 ↪style=ButtonStyle())
```

```
Output()
```

1.6 Exercise 2:

Now suppose you are throwing the dice (same dice as in the previous exercise) two times and recording the sum of each throw. Which of the following `probability mass functions` will be the one you should get?

Hints: - You can use numpy arrays to hold the results of many throws. - You can sum to numpy arrays by using the `+` operator like this: `sum = first_throw + second_throw` - To simulate multiple throws of a dice you can use list comprehension or a for loop

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[9]:
```
# Run this cell to submit your answer
utils.exercise_2()
```

```
ToggleButtons(description='Your answer:', options=('left', 'center', 'right'),␣
 ↪value='left')
```

```
Button(button_style='success', description='Save your answer!',␣
 ↪style=ButtonStyle())
```

```
Output()
```

1.7 Exercise 3:

Given a fair 4-sided dice, you throw it two times and record the sum. The figure on the left shows the probabilities of the dice landing on each side and the right figure the histogram of the sum. Fill out the probabilities of each sum (notice that the distribution of the sum is symetrical so you only need to input 4 values in total):

Submission considerations: - Submit your answers as floating point numbers with three digits after the decimal point - Example: To submit the value of 1/4 enter 0.250

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[10]:
```
# Run this cell to submit your answer
utils.exercise_3()
```

```
FloatText(value=0.0, description='P for sum=2',␣
 ↪style=DescriptionStyle(description_width='initial'))
```

```
FloatText(value=0.0, description='P for sum=3:',␣
 ↪style=DescriptionStyle(description_width='initial'))
```

```
FloatText(value=0.0, description='P for sum=4:',␣
 ↪style=DescriptionStyle(description_width='initial'))
```

```
FloatText(value=0.0, description='P for sum=5:',␣
  ↪style=DescriptionStyle(description_width='initial'))
```

```
Button(button_style='success', description='Save your answer!',␣
  ↪style=ButtonStyle())
```

```
Output()
```

1.8 Exercise 4:

Using the same scenario as in the previous exercise. Compute the mean and variance of the sum of the two throws and the covariance between the first and the second throw:

Hints: - You can use np.cov to compute the covariance of two numpy arrays (this may not be needed for this particular exercise).

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[11]:
```
# Run this cell to submit your answer
utils.exercise_4()
```

```
FloatText(value=0.0, description='Mean:')
```

```
FloatText(value=0.0, description='Variance:')
```

```
FloatText(value=0.0, description='Covariance:')
```

```
Button(button_style='success', description='Save your answer!',␣
  ↪style=ButtonStyle())
```

```
Output()
```

1.9 Exercise 5:

Now suppose you are have a loaded 4-sided dice (it is loaded so that it lands twice as often on side 2 compared to the other sides):

You are throwing it two times and recording the sum of each throw. Which of the following `probability mass functions` will be the one you should get?

Hints: - You can use the `p` parameter of np.random.choice to simulate a loaded dice.

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[12]:
```
# Run this cell to submit your answer
utils.exercise_5()
```

```
ToggleButtons(description='Your answer:', options=('left', 'center', 'right'),␣
  ↪value='left')
```

```
Button(button_style='success', description='Save your answer!',␣
  ↪style=ButtonStyle())
```

```
Output()
```

1.10 Exercise 6:

You have a 6-sided dice that is loaded so that it lands twice as often on side 3 compared to the other sides:

You record the sum of throwing it twice. What is the highest value (of the sum) that will yield a cumulative probability lower or equal to 0.5?

Hints: - The probability of side 3 is equal to $\frac{2}{7}$

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[13]:
```
# Run this cell to submit your answer
utils.exercise_6()
```

```
IntSlider(value=2, continuous_update=False, description='Sum:', max=12, min=2)
```

```
Button(button_style='success', description='Save your answer!',␣
↪style=ButtonStyle())
```

```
Output()
```

1.11 Exercise 7:

Given a 6-sided fair dice you try a new game. You only throw the dice a second time if the result of the first throw is **lower** or equal to 3. Which of the following `probability mass functions` will be the one you should get given this new constraint?

Hints: - You can simulate the second throws as a numpy array and then make the values that met a certain criteria equal to 0 by using np.where

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[14]:
```
# Run this cell to submit your answer
utils.exercise_7()
```

```
ToggleButtons(description='Your answer:', options=('left-most', 'left-center',␣
↪'right-center', 'right-most'), …
```

```
Button(button_style='success', description='Save your answer!',␣
↪style=ButtonStyle())
```

```
Output()
```

1.12 Exercise 8:

Given the same scenario as in the previous exercise but with the twist that you only throw the dice a second time if the result of the first throw is **greater** or equal to 3. Which of the following `probability mass functions` will be the one you should get given this new constraint?

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[15]:
```
# Run this cell to submit your answer
utils.exercise_8()
```

```
ToggleButtons(description='Your answer:', options=('left-most', 'left-center',␣
↪'right-center', 'right-most'), …
```

```
Button(button_style='success', description='Save your answer!',␣
↪style=ButtonStyle())
```

```
Output()
```

1.13 Exercise 9:

Given a n-sided fair dice. You throw it twice and record the sum. How does increasing the number of sides `n` of the dice impact the mean and variance of the sum and the covariance of the joint distribution?

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[16]:
```
# Run this cell to submit your answer
utils.exercise_9()
```

```
As the number of sides in the die increases:
```

```
ToggleButtons(description='The mean of the sum:', options=('stays the same',␣
↪'increases', 'decreases'), value=…
```

```
ToggleButtons(description='The variance of the sum:', options=('stays the same',␣
↪'increases', 'decreases'), va…
```

```
ToggleButtons(description='The covariance of the joint distribution:',␣
↪options=('stays the same', 'increases',…
```

```
Button(button_style='success', description='Save your answer!',␣
↪style=ButtonStyle())
```

```
Output()
```

1.14 Exercise 10:

Given a 6-sided loaded dice. You throw it twice and record the sum. Which of the following statemets is true?

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[17]:
```
# Run this cell to submit your answer
utils.exercise_10()
```

```
RadioButtons(layout=Layout(width='max-content'), options=('the mean and variance␣
↪is the same regardless of whi…
```

```
Button(button_style='success', description='Save your answer!',␣
↪style=ButtonStyle())
```

```
Output()
```

1.15 Exercise 11:

Given a n-sided dice (could be fair or not). You throw it twice and record the sum (there is no dependance between the throws). If you are only given the histogram of the sums can you use it to know which are the probabilities of the dice landing on each side?

In other words, if you are provided with only the histogram of the sums like this one:

Could you use it to know the probabilities of the dice landing on each side? Which will be equivalent to finding this histogram:

[ ]:
```
# You can use this cell for your calculations (not graded)
```

[18]:
```
# Run this cell to submit your answer
utils.exercise_11()
```

```
RadioButtons(layout=Layout(width='max-content'), options=('yes, but only if one␣
 ↪of the sides is loaded', 'no, …
```

```
Button(button_style='success', description='Save your answer!',␣
 ↪style=ButtonStyle())
```

```
Output()
```

1.16 Before Submitting Your Assignment

Run the next cell to check that you have answered all of the exercises

[21]:
```
utils.check_submissions()
```

```
All answers saved, you can submit the assignment for grading!
```

Congratulations on finishing this assignment!

During this assignment you tested your knowledge on probability distributions, descriptive statistics and visual interpretation of these concepts. You had the choice to compute everything analytically or create simulations to assist you get the right answer. You probably also realized that some exercises could be answered without any computations just by looking at certain hidden queues that the visualizations revealed.

Keep up the good work!