index

January 8, 2022

1 Introduction to Probability - Lab

1.1 Introduction

Now that you know what sets are, we can go on and work with two sets that are of key importance when talking about probability: the event space and the sample space. These two concepts are foundational for calculating probabilities when assuming each event in the event space *has the same probability of happening*. Typical examples are rolling a dice (if the dice is "fair", the chance of throwing each number between 1 and 6 is 1/6) and flipping a coin (1/2 heads vs tails). You'll get a better sense of how all of this works in this lab.

1.2 Objectives

You will be able to:

- Calculate probabilities by using relative frequency of outcomes to event space
- Use Python to demonstrate the axioms of probability

1.3 Sample space, event space and the law of relative frequency

a. Let's throw a dice once First, create a set `roll_dice` that holds the sample space of rolling a 6-sided dice once.

[2]:
```
roll_dice = set(range(1,7))
roll_dice
```

[2]:
```
{1, 2, 3, 4, 5, 6}
```

Now, let's assume that the event space is defined by "throwing a number higher than 4". This means that we consider the outcome "successful" if a 5 or a 6 is thrown. Create a set that holds these values.

[4]:
```
event = set(range(5,7))
event
```

[4]:
```
{5, 6}
```

Now use the formula $P(E) = \frac{|E|}{|S|}$ (This formula is called "Laplace's formula" and strongly related to the law of relative frequency) to calculate the probability.

[5]:
```
prob_5_6 = len(event)/len(roll_dice)
prob_5_6  # 0.3333333333333333
```

[5]:
```
0.3333333333333333
```

Using this formula, it should be clear that the answer is 1/3 or 0.3333….

b. Now, let's simulate rolling dice to see how the law of relative frequency works. The law of relative frequency can be used to prove certain probabilities. But how does this work exactly? You're about to find out!

$$P(E) = \lim_{n \to \infty} \frac{S(n)}{n}$$

As you can see in the formula, the law states that when repeating an experiment n times, where n is very big, and you divide the number of "good" outcomes by the sample space (here we call it event E), you get to the probability of the event E. It should be clear that we get a more accurate number for P(E) when n grows.

Let's see how this works. First, let's randomly generate values between 1 and 6. You can use `numpy` (imported as `np`) to generate random integers between 1 and 6 by setting the correct arguments. The `np.random` module is a very useful tool for this. We helped you with the code here, but you'll get more practice and a thorough explanation later on!

[8]:
```
import numpy as np
np.random.randint(1,7)
# you will get a random value between 1 and 6. See how it changes when you rerun
```

[8]:
```
6
```

Now, let's repeat this experiment 10 times, then 1000 times, then 1 million times, then 100 million times. You can do this by specifying the argument `size` within the numpy function used above. Store the values in the pre-defined variables below.

[18]:
```
np.random.seed(12345) # to ensure reproducibility of results

dice_10 = np.random.randint(1,7,size= 10)
dice_1k = np.random.randint(1,7,size= 10**3)
dice_1m = np.random.randint(1,7,size=10**6)
dice_100m = np.random.randint(1,7,size=10**8)
```

Next, let's count the number of "events". Remember that an event here is defined as throwing a 5 or a 6. Store them in the values below.

[19]:
```
event_10 = np.sum(dice_10>4)
event_1k = np.sum(dice_1k>4)
event_1m = np.sum(dice_1m>4)
event_100m = np.sum(dice_100m>4)
```

Next, you'll divide the number of events for each n by the respective values for n. What do you see?

[21]:
```
prob_10 = event_10/10
prob_1k = event_1k / 10**3
prob_1m = event_1m / 10**6
prob_100m = event_100m / 10**8
prob_10, prob_1k, prob_1m, prob_100m  # 0.5 0.331 0.333657 0.33329752
```

[21]:
```
(0.5, 0.331, 0.333657, 0.33329752)
```

You see that the probability converges to 0.3333333… for higher values of n.

1.4 The Probability Axioms

You're working at the United Nations, and want to get a better sense of the world population.

You come across some numbers and find the list of probabilities of being an inhabitant for each of the seven continents (rounded up to 3 digits):

- P(Africa) = 0.161
- P(Antarctica) = 0.000
- P(Asia) = 0.598
- P(Europe) = 0.10
- P(North-America) = 0.079
- P(Australia) = 0.005
- P(South-America) = 0.057

store these values using the variable names below:

[22]:
```
P_afr = 0.161
P_ant = 0.000
P_as = 0.598
P_eur = 0.10
P_na = 0.079
P_aus = 0.005
P_sa = 0.057
```

Now create the sample space set names `continents`. Store the sample space in a numpy array.

[23]:
```
continents = np.array([P_afr, P_ant, P_as, P_eur, P_na, P_aus, P_sa])
print(continents)
```

```
[0.161 0.    0.598 0.1   0.079 0.005 0.057]
```

We want to make sure that the three probability axioms are fulfilled because they assure us that (Ω, E, P) is a **probability space**:

- if we have a sample space S (or Ω)
- if we have an event space E and a probability measure P,
- **and** the three probability axioms introduced in the previous lesson are fulfilled,

The third axiom is fairly ad hoc, and you will basically have to deduce from the context whether individual events are independent. It is fairly straightforward, however, that people cannot be inhabitants of two continents at the same time, so for now, we will assume that we're good for axiom three.

However, we can use the numpy array `continents` to verify if axiom 1 and 2 are fulfilled. Create a function `check_axioms` with a single input, `sample_space`, that returns the message "We're good!" if both axiom 1 and 2 are fulfilled, and "Not quite!" if that's not the case.

[26]:
```
def check_axioms(sample_space):
    a1 = np.sum(sample_space >= 0)
    a2 = np.sum(sample_space <= 1)
    a3 = int(np.sum(sample_space) == 1)
    if (a1 == len(sample_space)) & (a2 == len(sample_space)) & (a3 == 1):
        return "We're Good!"
    else:
        return "Not quite!"
```

Now test your newly created function out on the sample space `continents`:

[27]:
```
check_axioms(continents)
```

[27]:
```
"We're Good!"
```

You want to make sure your test returns `"Not quite!"` for the following numpy arrays. Go ahead and test away!

[28]:
```
test_1 = np.array([0.05, 0.2, 0.3, 1.01])
test_2 = np.array([0.05, 0.5, 0.6, -0.15])
test_3 = np.array([0.043,0.05,.02,0.3,0.2])
```

[29]:
```
print(check_axioms(test_1))
print(check_axioms(test_2))
print(check_axioms(test_3))
```

```
Not quite!
Not quite!
Not quite!
```

Great! We tested it and seems like our set `continents` is a true probability space.

1.5 Some more practice on the sample and event spaces

In this exercise, we'll look at possible outcomes when throwing a dice twice. For your convenience, we created the NumPy array below.

Next, we'll compute a couple or probabilities associated with doing this.

[33]:
```
import numpy as np
sample_dice = np.array([(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6),
                        (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6),
                        (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6),
                        (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6),
                        (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6),
                        (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)])
```

Look at the shape of the array to reassure we haven't made any mistakes.

[34]:
```
np.shape(sample_dice) # should be equal to (36,2)
```

[34]:
```
(36, 2)
```

Use Python to obtain the following probabilities:

a. What is the probability of throwing a 5 at least once? First, use `sample_dice` to get "True" values for each time a 5 occurs.

[36]:
```
set_5 = sample_dice == 5
# Your output should be an array of shape (36, 2) with booleans instead of␣
 ↪numbers
set_5
```

[36]:
```
array([[False, False],
       [False, False],
       [False, False],
       [False, False],
       [False,  True],
       [False, False],
       [False, False],
       [False, False],
       [False, False],
       [False, False],
       [False,  True],
       [False, False],
       [False, False],
       [False, False],
       [False, False],
       [False, False],
       [False,  True],
       [False, False],
       [False, False],
       [False, False],
       [False, False],
       [False, False],
       [False,  True],
       [False, False],
```

```
       [ True, False],
       [ True, False],
       [ True, False],
       [ True, False],
       [ True,  True],
       [ True, False],
       [False, False],
       [False, False],
       [False, False],
       [False, False],
       [False,  True],
       [False, False]])
```

Next, make sure that you get a value `True` for each pair where at least one 5 was thrown.

[42]:
```
true_5 = np.any(set_5, axis = 1)

# Your output should be an array of shape (36,) and have booleans.
              # You should obtain "True" if at least one of the previous pairs␣
 ↪was true.
              # Hint: Use np.any()
print(true_5)
```

```
[False False False False  True False False False False False  True False
 False False False False  True False False False False False  True False
  True  True  True  True  True  True False False False False  True False]
```

Applying the `sum()` function to this result you can get to the total number of items in the event space. Divide this number by the total number in the sample space to obtain the probability of throwing a 5 at least once.

[44]:
```
prob_5 = np.sum(true_5) / len(sample_dice)
print(prob_5)
```

```
0.3055555555555556
```

b. What is the probability of throwing a 5 or 6 at least once?

[45]:
```
set_5 = sample_dice == 5
set_6 = sample_dice == 6
```

[46]:
```
set_5_6 = set_5 + set_6
```

[47]:
```
set_any_5_6 = np.any(set_5_6, axis = 1)
print(set_any_5_6)
```

```
[False False False False  True  True False False False False  True  True
 False False False False  True  True False False False False  True  True
  True  True  True  True  True  True  True  True  True  True  True  True]
```

[48]:
```
prob_5_6 = set_any_5_6.sum() / len(sample_dice)
print(prob_5_6)
```

```
0.5555555555555556
```

c. What is the probability of the outcome having a sum of exactly 8?

[56]:
```
sum_dice = np.sum(sample_dice, axis = 1)
sum_8 = np.sum(sum_dice == 8)
sum_8
```

[56]:
```
5
```

[57]:
```
prob_sum_8 = sum_8 / len(sum_dice)
print(prob_sum_8)
```

```
0.1388888888888889
```

1.6 Now let's try creating your own event space!

A teaching assistant is holding office hours so students can make appointments. She has 6 appointments scheduled today, 3 by male students, and 3 by female students.

Create a NumPy array of all possible orders of three male and three female students in which the teaching assistant can see students by appointment (please note: only consider gender when creating event space). Create this NumPy array in the same way as we did in the "throwing a dice twice" exercise. It will be quite a bit of typing, as your resulting NumPy array will have a shape (20,6)!

[58]:
```
# From GitHub
sample_mf= np.array([("M","M","M","F","F","F"), ("M","M","F","M","F","F"),␣
 ↪("M","M","F","F","M","F"),
                     ("M","M","F","F","F","M"), ("M","F","M","M","F","F"),␣
 ↪("M","F","M","F","F","M"),
                     ("M","F","M","F","M","F"), ("M","F","F","M","F","M"),␣
 ↪("M","F","F","M","M","F"),
                     ("M","F","F","F","M","M"), ("F","F","F","M","M","M"),␣
 ↪("F","F","M","F","M","M"),
                     ("F","F","M","M","F","M"), ("F","F","M","M","M","F"),␣
 ↪("F","M","F","F","M","M"),
                     ("F","M","F","M","M","F"), ("F","M","F","M","F","M"),␣
 ↪("F","M","M","F","M","F"),
                     ("F","M","M","F","F","M"), ("F","M","M","M","F","F") ])
```

[60]:
```
sample_mf.shape # get the shape of sample_mf
```

[60]:
```
(20, 6)
```

[61]:
```
sample_length= len(sample_mf)
print(sample_length)
```

```
20
```

1. Calculate the probability that at least 2 out of the first 3 appointments are with female students First, select the first 3 appointment slots and check for "F".

[63]:
```
first_3_F = sample_mf[:,:3] == "F"
print(first_3_F)
```

```
[[False False False]
 [False False  True]
 [False False  True]
 [False False  True]
 [False  True False]
 [False  True False]
 [False  True False]
 [False  True  True]
 [False  True  True]
 [False  True  True]
 [ True  True  True]
 [ True  True False]
 [ True  True False]
 [ True  True False]
 [ True False  True]
 [ True False  True]
 [ True False  True]
 [ True False False]
 [ True False False]
 [ True False False]]
```

[64]:
```
num_F = first_3_F.sum(axis = 1)
print(num_F)
```

```
[0 1 1 1 1 1 1 2 2 2 3 2 2 2 2 2 2 1 1 1]
```

[72]:
```
F_2plus = sum(num_F > 1)
print(F_2plus)
```

```
10
```

[73]:
```
prob_F_2plus = F_2plus/sample_length
print(prob_F_2plus)
```

```
0.5
```

2. Calculate the probability that after 4 appointment slots, all the female students have had an appointment

[87]:
```
a1 = sample_mf[:,4:] == ["M", "M"]
a2 = np.sum(a1, axis = 1)
a3 = np.sum(a2 > 1)
prob = a3 / sample_length
print(prob)

## From GitHub

git = np.sum((sample_mf[:,4:] == ['M','M']).all(axis=1))/sample_length
print(git)
```

```
0.2
0.2
```

You noticed that coming up with the sample space was probably the most time-consuming part of the exercise, and it would really become unfeasible to write this down for say, 10 or, even worse, 20 appointments in a row. You'll learn about methods that make this process easier soon!

1.7 The Addition Law of Probability

At a supermarket, we randomly select customers, and make notes of whether a certain customer owns a Visa card (event A) or an American Express credit card (event B). Some customers own both cards. You can assume that:

- P(A) = 0.5
- P(B) = 0.4
- both A and B = 0.25.

1) Compute the probability that a selected customer has at least one credit card.

2) Compute the probability that a selected customer doesn't own any of the mentioned credit cards.

3) Compute the probability that a customer *only* owns VISA card.

(You can use Python here, but you don't have to)

[91]:
```
P_A = 0.5
P_B = 0.4
P_A_and_B = 0.25
a1 = P_A + P_B - P_A_and_B
print(a1)
a2 = 1 - a1
print(a2)
a3 = P_A - P_A_and_B
print(a3)
```

```
0.65
0.35
```

```
0.25
```

1.8 Summary

In this lab, you got to practice your knowledge on the foundations of probability through working on problems regarding the law of relative frequency, the probability axioms, and the addition law of probability.