Simulations with Conditional and Total Probability - Lab

Introduction

In this lab, we shall run simulations for simple total probability problems. We shall solve these problems by hand and also perform random sampling from a defined probability distribution repeatedly to see if our calculated results match the results of random simulations.

Objectives

You will be able to:

- Use knowledge of conditional probabilities, total probabilities, and the product rule to run random simulations using NumPy

Exercise 1

Part 1

Suppose you have two bags of marbles. The first bag contains 6 white marbles and 4 black marbles. The second bag contains 3 white marbles and 7 black marbles. Now suppose you put the two bags in a box. Now if you close your eyes, grab a bag from the box, and then grab a marble from the bag, **what is the probability that it is black**?

In [1]:
```
# Your answer here
# P(BLACK) = P(CHOOSE BAG 1) * P(BLACK|CHOOSE BAG 1) + P(CHOOSE BAG 2) * P(BLACK|
p_b = ((1/2)*(4/10)) + ((1/2)*(7/10))
p_b

# The probability is 11/20 or 0.55
```

Out[1]:
```
0.55
```

Part 2

Run a simple simulation to estimate the probability of drawing a marble of a particular color. Run the code and verify that it agrees with your computation done earlier.

Perform following tasks:

- Create dictionaries for bag1 and bag2 holding marble color and probability values:
  - **bag1 = {'marbles' : np.array(["color1", "color2"]), 'probs' : np.array([P(color1), P(color2)])}**
- Create a dictionary for the box holding the bags and their probability values:
  - **box = {'bags' : np.array([bag1, bag2]), 'probs' : np.array([P(bag1),P(bag2)])}**

- Show the content of your dictionaries

In [2]:
```
import numpy as np
bag1 = {'marbles' : np.array(["black", "white"]), 'probs' : np.array([4/10, 6/10]
bag2 = {'marbles' : np.array(["black", "white"]), 'probs' : np.array([7/10, 3/10]
box  = {'bags' : np.array([bag1, bag2]), 'probs' : np.array([1/2, 1/2])}

bag1, bag2, box

# ({'marbles': array(['black', 'white'], dtype='<U5'),
#   'probs': array([0.4, 0.6])},

#  {'marbles': array(['black', 'white'], dtype='<U5'),
#   'probs': array([0.7, 0.3])},

#  {'bags': array([{'marbles': array(['black', 'white'], dtype='<U5'), 'probs': a
#          {'marbles': array(['black', 'white'], dtype='<U5'), 'probs': array([0.
#         dtype=object), 'probs': array([0.5, 0.5])})
```

Out[2]:
```
({'marbles': array(['black', 'white'], dtype='<U5'),
  'probs': array([0.4, 0.6])},
 {'marbles': array(['black', 'white'], dtype='<U5'),
  'probs': array([0.7, 0.3])},
 {'bags': array([{'marbles': array(['black', 'white'], dtype='<U5'), 'probs': a
rray([0.4, 0.6])},
         {'marbles': array(['black', 'white'], dtype='<U5'), 'probs': array([0.
7, 0.3])}],
        dtype=object),
  'probs': array([0.5, 0.5])})
```

Create a function `sample_marble(box)` that randomly chooses a bag from the box and then randomly chooses a marble from the bag

In [3]:
```
def sample_marble(box):
    # randomly choose a bag
    bag = np.random.choice(box['bags'], p = box['probs'])
    # randomly choose a marble
    return np.random.choice(bag['marbles'], p = bag['probs'])

sample_marble(box)
# 'black' OR 'white'
```

Out[3]:
```
'black'
```

Create another function `probability_of_colors(color, box, num_samples)` that gets a given number of samples from `sample_marbles()` and computes the fraction of marbles of a desired color

In [4]:
```
def probability_of_color(color, box, num_samples=1000):
    # get a bunch of marbles
    marbles = np.array([sample_marble(box) for ii in range(num_samples)])
    # compute fraction of marbles of desired color
    return np.sum(marbles == color) / num_samples
```

Now let's run our function in line with our original problem, i.e. the probability of seeing a black marble by sampling form the box 100000 times.

In [5]:
```
# probability_of_color("black", box, num_samples=100000)
probability_of_color("black", box, num_samples=100000)
# very close to 0.55

# your answer should be very close to 0.55
```

Out[5]:
```
0.54956
```

Exercise 2

Suppose now we add a third color of marble to the mix. Bag 1 now contains 6 white marbles, 4 black marbles, and 5 gray marbles. Bag 2 now contains 3 white marbles, 7 black marbles, and 5 gray marbles.

The probability of grabbing the first bag from the box is now TWICE the probability of grabbing the second bag.

What is the probability of drawing a gray marble from the bag according to law of total probabilities?

Copy and paste the code from the exercise above and modify it to estimate the probability that you just computed and check your work.

In [6]:
```
# Change above code here
# Since there is a 1/3 probability of drawing a gray marble from each bag
# (2 bags in total), it seems like the desired probability should be 1/3
# Let's check this using the law of total probability.
# Let G be the event that we select a gray marble,
# and  $B_1$ and  $B_2$ be the events that we select Bags 1 and 2 from the box, r
# We then have P(G) = P(G| B_1)P(B_1)+P(G| B_2)P(B_2)=(1/3)*(2/3)+(1/3)*(1/3)
#                    = (2/9)+(1/9)=(3/9)=(1/3)
```

In [7]:
```
# probability_of_color("gray", box, num_samples=100000)

import numpy as np
bag1 = {'marbles' : np.array(["black", "white", "gray"]), 'probs' : np.array([4/1
bag2 = {'marbles' : np.array(["black", "white", "gray"]), 'probs' : np.array([7/1
box  = {'bags' : np.array([bag1, bag2]), 'probs' : np.array([2/3, 1/3])}

def sample_marble(box):
    # randomly choose a bag
    bag = np.random.choice(box['bags'], p = box['probs'])
    # randomly choose a marble
    return np.random.choice(bag['marbles'], p = bag['probs'])

def probability_of_color(color, box, num_samples=1000):
    # get a bunch of marbles
    marbles = np.array([sample_marble(box) for ii in range(num_samples)])
    # compute fraction of marbles of desired color
    return np.sum(marbles == color) / num_samples

# Very close to 0.33
```

Summary

In this lab, you looked at some more examples of simple problems using the law of total probability. You also ran some simulations to solve these problems using continuous random sampling. You learned that you get a result very close to the mathematical solution when using random sampling. This difference is due to randomness, and as your sample size grows you'll see that the difference becomes very small! For more complex problems with larger datasets, having an understanding of the underlying probabilities can help you solve a lot of optimization problems as you'll learn later.