Practicing with joint probability mass functions

Contents


```
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(rc={"figure.dpi":100, 'savefig.dpi':300})
sns.set_context('notebook')
sns.set_style("ticks")
```

Objectives

- To practice with the joint probability mass function.

Joint probability mass function of two discrete random variables

Take two discrete random variabls X and Y. Say that X takes 5 values, $0, 1, \ldots, 4$, and Y takes 10 values, $0, 1, \ldots, 9$. Then, you can think of the joint probability mass function of X and Y as the 5×10 matrix:

$$A_{ij} = p(X = i, Y = j).$$

Let's make up such a matrix to play with:

```
# This is to make sure that we all get the same results
np.random.seed(12345)
# First I draw a matrix with random entries in [0,1]
A = np.random.rand(5, 10)
print(A)
```

Why are the entries random? Why isn't each index the value of the normalization constant?

```
[[0.93  0.316 0.184 0.205 0.568 0.596 0.965 0.653 0.749 0.654]
 [0.748 0.961 0.008 0.106 0.299 0.656 0.81  0.872 0.965 0.724]
 [0.642 0.717 0.468 0.326 0.44  0.73  0.994 0.677 0.791 0.171]
 [0.027 0.8   0.904 0.025 0.492 0.526 0.596 0.052 0.895 0.728]
 [0.818 0.5   0.81  0.096 0.219 0.259 0.468 0.459 0.71  0.178]]
```

```
# And then I scale it so that the sum of all elements is one:
A = A / np.sum(A)
print(f"A = {A}")
print(f"Sum of A_ij = {np.sum(A):.2f}")
```

```
A = [[0.034 0.012 0.007 0.007 0.021 0.022 0.035 0.024 0.027 0.024]
 [0.027 0.035 0.    0.004 0.011 0.024 0.029 0.032 0.035 0.026]
 [0.023 0.026 0.017 0.012 0.016 0.027 0.036 0.025 0.029 0.006]
 [0.001 0.029 0.033 0.001 0.018 0.019 0.022 0.002 0.033 0.026]
 [0.03  0.018 0.029 0.003 0.008 0.009 0.017 0.017 0.026 0.006]]
Sum of A_ij = 1.00
```

Now we have a matrix that corresponds to a proper joint probability mass function. Is this arbitrary/random as it doesn't follow any distribution?

Okay, this is great we can sample from this. Let's find now the probability mass function of just X. Remember that you need to marginalize:

$$p(x) = \sum_y p(x, y) = \sum_y A_{xy}.$$

This is easy:

```
p_x = np.sum(A, axis=1) # Axis = 1 tells sum to sum only the second axis
print(f"pmf of just X: {p_x}")
```

```
pmf of just X: [0.212 0.224 0.217 0.184 0.164]
```

Verify that this is indeed a pmf:

```
print(f"sum of p_x = {np.sum(p_x):.2f}")
```

```
sum of p_x = 1.00
```

With this you can easily find the expectation of X:

```
E_X = np.sum(np.arange(5) * p_x)
print(f"E[X] = {E_X:.2f}")
```

Note how expectation is calculated using python syntax without using the rv built-in

```
E[X] = 1.87
```

Similarly for the variance of X:

```
E_X2 = np.sum(np.arange(5) ** 2 * p_x)
V_X = E_X2 - E_X ** 2
print(f"V[X] = {V_X:.2f}")
```

```
V[X] = 1.89
```

Let's do the same for Y:

```
p_y = np.sum(A, axis=0)
print(f"pmf of just Y: {p_y}")
E_Y = np.sum(np.arange(10) * p_y)
print(f"E[Y] = {E_Y:.2f}")
E_Y2 = np.sum(np.arange(10) ** 2 * p_y)
V_Y = E_Y2 - E_Y ** 2
print(f"V[Y] = {V_Y:.2f}")
```

```
pmf of just Y: [0.115 0.12  0.086 0.028 0.073 0.101 0.139 0.099 0.149 0.089]
E[Y] = 4.70
V[Y] = 8.98
```

Alright, we have found all the individual statistics. Let's now find the covariance of the two random variables. Remember the formula:

$$\mathbb{C}[X, Y] = \sum_{x,y} (x - \mathbb{E}[X])(y - \mathbb{E}[Y])p(x, y).$$

Expectation is computed over a sum, where the summand arguments are function of x and y respectively

Here we go:

```
# We will loop over all the possible values
C_XY = 0.0 # Keeping track of the sum
for x in range(5):
    for y in range(10):
        C_XY += (x - E_X) * (y - E_Y) * A[x, y] # the += means add to the left hand side
print(f"C[X, Y] = {C_XY:.2f}")
```

```
C[X, Y] = -0.38
```

We see that X and Y are negatively correlated. If only we could sample from them to visualize it... How can we do this? We cannot just sample X and then Y without thinking about it. We need to sample X and Y together. Basically, we need to sample a set of index (i, j) with probability A_{ij}. This is like sampling from a categorical with $5 \times 10 = 50$ different labels $c_0 = (1, 1), c_1 = (1, 2), \ldots, c_{49} = (5, 10)$ each with a probability $A_{00}, A_{01}, \ldots, A_{4,9}$

a probability $A_{00}, A_{01}, \ldots, A_{4,9}$.

This is what the code below does. You can look at the details, if you are so inclined.

```
import scipy.stats as st

# A.flatten() is the matrix flattened out as a row
XY = st.rv_discrete('Joinnt XY', values=(range(50), A.flatten()))
# Let's now write a function that samples X and Y using a sample from XY
def sample_X_and_Y():
    """Samples X and Y once."""
    k = XY.rvs()
    # This is integer division
    i = k // 10
    # This is the remainder
    j = k % 10
    return i, j
```

values are states, followed by corresponding probabilities

This code essentially samples a random index from the array. Recall that i is row and j is column, and x and y can take on states of 0-4 and 0-9 respectively, so these are just the indices that are sampled according to the probabilities in the array.

Let's try it out - take 10 samples:

```
for n in range(10):
    x, y = sample_X_and_Y()
    print(f"x = {x:d}, y = {y:d}")
```

```
x = 2, y = 5
x = 0, y = 8
x = 3, y = 6
x = 4, y = 5
x = 2, y = 7
x = 0, y = 7
x = 2, y = 2
x = 1, y = 8
x = 4, y = 0
x = 4, y = 2
```

Now, let's write some code to find the expectation of a function $f(X, Y) = X + Y$. It is:

$$\mathbb{E}[f(X,Y)] = \sum_{x,y} f(x,y)p(x,y) = \sum_{x,y} f(x,y)A_{xy}.$$

The array A is the joint pdf in this situation

Let's write a function that calculates the expectation of any f and then use it for $f(X, Y) = X + Y$.

```
def expectation(f, A):
    """Returns the expectation of the function f(X, Y).

    Arguments:
    f -- A function f(X, Y).
    A -- A matrix containing the joint probability mass
         function of X and Y. Say that A has dimensions
         n x m. We assume that X takes values
         1, 2, ..., n and that Y takes values
         1, 2, ..., m.
    """
    n, m = A.shape
    res = 0.
    for i in range(n):
        for j in range(m):
            res += f(i - 1, j - 1) * A[i, j]
    return res
```

This has been identified as incorrect by the instructor. The indices of f() within the nested loop should be i+1, j+1 since the looping starts at zero but the values that X and Y take start at 1 (see comment above in this function definition)

Notice that `expectation` is a Python function that takes as an input the Python function of which we want to find the expectation. This is cool!

Here is how to use `expectation`. First, you need to define the function the expectation of which you want to find.

```
def f(x, y):
    return x + y
```

We can calculate the expectation like this:

```
E_f = expectation(f, A)

print(f"E[f(X, Y)] = {E_f:.2f}")
```

```
E[f(X, Y)] = 4.56
```

Now let's find the variance of $f(X,Y)$. We can use the formula:

$$\mathbb{V}[f(X,Y)] = \mathbb{E}\left[f^2(X,Y)\right] - \{\mathbb{E}[f(X,Y)]\}^2.$$

We need to define a function that corresponds to the square of $f(X,Y)$. Here it is:

```
def g(x, y):
    return f(x, y) ** 2
```

So, now we can do:

```
V_f = expectation(g, A) - E_f ** 2

print(f"V[f(X, Y)] = {V_f:.2f}")
```

```
V[f(X, Y)] = 10.11
```

Questions

- Modify the code above to find the variance of $3X + 5Y$.
- Write code that finds the expectation of the function $f(X,Y) = XY^3$.

```
# Your code here
```

If you are feeling bored, learn some functional programming...

I have to show you a neat way to do this using functional programming ideas. Feel free to skip if you find it wierd... It is not required for the rest of the course.

First, here is how you can define a function super-quickly in Python:

```
f = lambda x, y: x + y
```

Now, I am going to define a function that squares any function. Pay attention, I will define a function that takes as an argument any function (with any arguments) and it returns another function which is just the square of the first. Here it is in one line:

```
square = lambda f: lambda *args: f(*args) ** 2
```

A function of a function

Here is what I can do with this:

```
f2 = square(f)

print(f"f(1,2) = {f(1,2)}, f(1, 3) ** 2 = {f2(1,2)}")
```

```
f(1,2) = 3, f(1, 3) ** 2 = 9
```

So, now can find the variance like this:

```
V_f = expectation(square(f), A) - expectation(f, A) ** 2

print(f"V[f(X, Y)] = {V_f:.2f}")
```

```
V[f(X, Y)] = 10.11
```

By Ilias Bilionis (ibilion[at]purdue.edu)