CMPT 423/820

Assignment 1 Question 4

- Seyedeh Mina Mousavifar
- 11279515
- sem311

In this question, we'll do a little more work with Numpy, and we'll combine this with a bit of review from Lecture 02 -- Probability. Numpy can do the calculations here, but you the programmer have to manage really tiny details about the NDArray data structure. It's not ideal for convenient work with discrete probability distributions, but it is a little instructive on the use of Numpy.

Tasks

On Slide 18 of Lecture 02, there is a Joint Probability Distribution on 2 random variables A and W. This JPD will be the basis for some Numpy calculations. In parts 4 and following, below, the exercises create a new JPD $P_2(AW); this is not supposed to be equal to the JPD from part 1.

1. Define a NDArray to store the numeric values of the JPD.

In [178]:
```
import numpy as np

# based on Lecture 02, page 16
jpd = np.array([[0.144, 0.02, 0.016, 0.02], [0.576, 0.08, 0.064, 0.08]])
print('P_1(AW):\n', jpd)
```

```
P_1(AW):
 [[0.144 0.02  0.016 0.02 ]
 [0.576 0.08  0.064 0.08 ]]
```

1. Using the Numpy method `sum()` , do the following:
   A. Display the sum of all the values in the JPD. It should be (very close to) 1.
   B. Display the marginal $P(W)$ as an NDArray obtained from the JPD using `sum()` .
   C. Display the marginal $P(A)$ as an NDArray obtained from the JPD using `sum()` .

In [179]:
```
# A: sum of all the values in JPD
print('Sum of all the values in JPD=', np.sum(jpd))

# B: marginal P(W) - vertical sum of each column
pw = np.sum(jpd, axis = 0)
print('Marginal P(W)=', pw)

# C: marginal P(A) - horizontal sum of each row
pa = np.sum(jpd, axis = 1)
print('Marginal P(A)=', pa)
```

```
Sum of all the values in JPD= 0.9999999999999999
Marginal P(W)= [0.72 0.1  0.08 0.1 ]
Marginal P(A)= [0.2 0.8]
```

1. Using Numpy methods, show that A and W are in fact independent, that is $P_1(AW) = P(A)P(W)$

In [180]:
```
# matrix multiplication of P(A)*P(W)
product_aw_1 = np.matmul(pa.reshape(2,1), pw.reshape(1,4))
print('The product of P(A)P(B) is: \n', product_aw_1)

print()

# checking equality to original JPD
print('Does JPD P(AW) equals to P(A)P(W)?', np.array_equal(jpd, product_
aw_1))
```

```
The product of P(A)P(B) is: 
 [[0.144 0.02  0.016 0.02 ]
 [0.576 0.08  0.064 0.08 ]]

Does JPD P(AW) equals to P(A)P(W)? True
```

1. Define a NDArray for a new conditional $P(A|W) \neq P(A)$. Choose your own values here.

In [181]:
```
# considering keeping old marginal on W
my_jpd = np.array([[0.15, 0.03, 0.02, 0.05], [0.57, 0.07, 0.06, 0.05]])

# checking my array to sum to 1
print('Sum of all the values in my JPD=', np.sum(my_jpd))

# checking P(A|W) != P(A) by P(AW)!=P(A)P(W)
my_pw = np.sum(my_jpd, axis = 0)
my_pa = np.sum(my_jpd, axis = 1)
my_product_aw = np.matmul(my_pa.reshape(2,1), my_pw.reshape(1,4))
print('Does P1(A|W) equals to P(A) for my JPD?', np.array_equal(my_jpd,
my_product_aw))
```

```
Sum of all the values in my JPD= 0.9999999999999999
Does P1(A|W) equals to P(A) for my JPD? False
```

1. Using Numpy methods, calculate the new JPD $P_2(AW) = P(A|W)P(W)$ (keep the old marginal on W). Note: we'll use this new JPD in some of the parts that follow.

In [182]:
```
# P(A|W) by dividing each row of my_jpd to P(W) value columnwise
pw = pw.reshape(1,4)
paw_conditional = np.true_divide(my_jpd, pw)


# P2(AW) = P(A|W)P(W)
p2_aw = paw_conditional * pw
print('The new JPD P2(AW):\n', product_aw_2)

print()

# checking equality to original JPD
print('Does JPD P2(AW) equals to P(A|W)P(W)?', np.array_equal(my_jpd, pr
oduct_aw_2))
```

```
The new JPD P2(AW):
 [[0.15 0.03 0.02 0.05]
 [0.57 0.07 0.06 0.05]]

Does JPD P2(AW) equals to P(A|W)P(W)? True
```

1. Using Numpy methods, calculate the conditional $P(W|A)$ from the new JPD.

In [183]:
```
# P(W|A) from my JPD by dividing each column of my_jpd to P(W) value row
wise
my_pa = my_pa.reshape(2,1)
pwa_conditional = np.true_divide(my_jpd, my_pa)

print('from P2(AW), P(W|A):\n', pwa_conditional)
```

```
from P2(AW), P(W|A):
 [[0.6        0.12       0.08       0.2       ]
 [0.76       0.09333333 0.08       0.06666667]]
```

1. Using Numpy methods, show that Bayes' Rule is true (for the new JPD):

$$P(W|A) = \frac{P(A|W)P(W)}{P(A)}$$

In [184]:
```
# P(A|W) by dividing each row of my_jpd to P(W) value columnwise
paw_conditional = np.true_divide(my_jpd, pw)

# calculating Bayes' rule right side
bayes_right = (paw_conditional * pw)/my_pa

print('Is Bayes Rule true?', np.array_equal(pwa_conditional, bayes_right
))
```

```
Is Bayes Rule true? True
```

1. Produce a Boolean NDArray showing which values of the new JPD are greater than the old JPD $P_1(AW)$.

In [185]:
```
boolean = np.greater(jpd, p2_aw)

print('Boolean Array for difference between new JPD and old JPD:\n',bool
ean)
```

```
Boolean Array for difference between new JPD and old JPD:
 [[False False False False]
 [ True  True  True  True]]
```

1. Using Numpy methods, calculate the number of values in the new JPD that are smaller than the old JPD $P_1(AW)$.

In [186]:
```
small_count = np.sum(np.less(jpd,p2_aw))
print('The number of values in the new JPD that are smaller than the old
JPD is', small_count)
```

```
The number of values in the new JPD that are smaller than the old JPD i
s 4
```

What to hand in

Your version of this notebook named A1Q4.pdf, containing completed work above, and your name and student number at the top.

Evaluation:

- 10 marks: Your answers show that you have basic mastery of Numpy.