CMPT 423/820

Assignment 1 Question 4 -- Solution and Grading

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 [1]:
```
import numpy as np

lecture2_JPD = np.array([[0.144, 0.02, 0.016, 0.02], [0.576, 0.08, 0.064
```

2. 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 [2]:
```
# A
print("The sum over the whole JPD is:", lecture2_JPD.sum())

# B
p_W = lecture2_JPD.sum(0)
print("The marginal P(W)", p_W)

# C
p_A = lecture2_JPD.sum(1)
print("The marginal P(A)", p_A)
```

```
The sum over the whole JPD is: 0.9999999999999999
The marginal P(W) [0.72 0.1  0.08 0.1 ]
The marginal P(A) [0.2 0.8]
```

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

In [3]:
```
# Numpy doesn't have a single tool for this, so we have to
# force the issue.  Create 2 arrays with the contents of P_A and P_W
# then multiply!

p_W_for_prod = np.tile(p_W, 2).reshape((2,4))
p_A_for_prod = np.tile(p_A, 4).reshape((4,2)).T

prod_AW = p_A_for_prod * p_W_for_prod

# calculate a difference array, which we'll sum up
diff = lecture2_JPD - prod_AW

# now check the sum
if abs(diff.sum()) < 0.001:
    print('The product is the same as the JPD, so independent!')
else:
    print('The product is not the same the JPD, so not independent!')
```

```
The product is the same as the JPD, so independent!
```

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

In [4]:
```
new_P_A_W = np.array([[0.2, 0.8], [0.1, 0.9], [0.5, 0.5], [0.8,0.2]])
```

5. 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 [5]:
```
new_prod_AW = new_P_A_W.T * p_W_for_prod
print("The new JPD is:")
print(new_prod_AW)
print("The sum over the new JPD is:", new_prod_AW.sum())
```

```
The new JPD is:
[[0.144 0.01  0.04  0.08 ]
 [0.576 0.09  0.04  0.02 ]]
The sum over the new JPD is: 1.0
```

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

In [6]:
```
# derive the marginal from the new JPD
new_p_A = new_prod_AW.sum(1)

print("The new marginal P(A)", new_p_A)
p_A_for_prod = np.tile(new_p_A, 4).reshape((4,2)).T

new_P_W_A = new_prod_AW / p_A_for_prod
print("The new conditional P(W|A)")
print(new_P_W_A)
```

```
The new marginal P(A) [0.274 0.726]
The new conditional P(W|A)
[[0.52554745 0.03649635 0.1459854  0.2919708 ]
 [0.79338843 0.12396694 0.05509642 0.02754821]]
```

~~7. Using Numpy methods, calculate the conditional $P(W|A)$ from the new JPD.~~

8. 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 [7]:
```
# well, that's the same as above.
lhs = new_P_W_A
rhs = (new_P_A_W.T * p_W) / np.tile(new_p_A, 4).reshape((4,2)).T

diff = lhs - rhs
if abs(diff.sum()) < 0.0001:
    print("Bayes Rule holds")
else:
    print("Bayes Rule does not hold")
```

```
Bayes Rule holds
```

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

In [8]:
```
greater = new_prod_AW > lecture2_JPD
print(greater)
```

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

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

In [9]:
```
number_smaller = new_prod_AW.size - greater.sum()
print("There are ", number_smaller, "entries in the new JPD smaller than
```

```
There are  5 entries in the new JPD smaller than in the old JPD.
```

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.

Grading:

1. The original JPD should have 8 entries, and sum to 1.
    - The numbers don't really matter, as long as they sum to 1.
    - It's valid to use the transpose of the shown table.
    - Deduct the mark if the table is not 2D, or doesn't sum to 1.
2. Use of `sum()` on a JPD.

    - The arguemnt to `sum()` indicates an axis. This is the easiest way to do get the answers.
    - Deduct the mark if the answer uses more Python than Numpy. It's okay to pull out a row, and call `sum()`.
      It's not okay to write a loop over the whole NDArray.
3. Independence by multiplying.
    - To get this to work, arrays have to be the same shape, or some other technique.
    - Deduct the mark if the answer is incorrect, or if Numpy was not used for part of it.
4. Defining the new conditional.
    - The conditional has to be 2D. The orientation doesn't matter.
    - The short axis **must sum to 1** The other axis probably won't but it's okay if it does.
5. Calculating the JPD from the $P(W)$ and $P(A|W)$.

    - More use of array-multiplication. The same idea as part 3.
    - Here, there is no right answer.
    - Deduct the mark if the resulting JPD doesn't sum to 1, or is the wrong shape.
6. Calculating a conditional.
    - More NDArray work, this time with division.
    - The resuting conditional distribution has to be 2D, and sum to 1 along the long axis.
    - Deduct the mark if the result is not appropriate as a conditional, or the wrong shape.
7. Copy paste error.
    - One free mark!
8. Wait, we did this before in part 6.
    - Deduct the mark if the equality is not shown somehow.
9. Creating a Boolean NDArray.
    - Using the relational expression is the whole point!
    - Deduct the mark if a relational expression is not used. Either `>` or `<` is valid.
10. Counting Booleans.
    - Boolean `True` counts as integer 1.
    - Deduct the mark if `sum()` us not used on the Booelan NDarray.