HW1 - codes

February 14, 2025

0.1 IMPORTS

[2]:
```
import numpy as np
```

0.2 QUESTION ONE

Consider a simple Markov Chain structure X → Y → Z, where all variables are binary. You are required to: (a) Write a code (using your preferred programming language) that generates a distribution (not necessarily a valid BN one) over the 3 variables.

[3]:
```
def generate_random_distribution():
    """
    Generates a random joint probability distribution P(X, Y, Z) for binary␣
 ↪variables.
    A valid probability distribution should sum up to 1.

    So we can generate random values and normalize them to make them a␣
 ↪probability distribution.
    """

    # We use 2 as the variable is a binary variable and it could take 0 or 1
    '''
    We have three variables X, Y, Z. So we need to generate a 2x2x2 matrix:␣
 ↪000, 001, 010, 011, 100, 101, 110, 111

    '''
    P = np.random.rand(2, 2, 2)  # we generate Random values for P(X, Y, Z)
    P /= np.sum(P)  # Normalize to make it a probability distribution
    return P

random = generate_random_distribution()
# I want to print the individual values and their probabilities
for i in range(2):
    for j in range(2):
        for k in range(2):
            print(f"P(X={i}, Y={j}, Z={k}) = {random[i, j, k]}")
'''
Output should be something like:
```

```
[[[0.023 0.073] # P(X=0, Y=0, Z=0) = 0.023, P(X=0, Y=0, Z=1) = 0.073
  [0.053 0.031]] # P(X=0, Y=1, Z=0) = 0.053, P(X=0, Y=1, Z=1) = 0.031

 [[0.1   0.1  ] # P(X=1, Y=0, Z=0) = 0.1, P(X=1, Y=0, Z=1) = 0.1
  [0.1   0.1  ]]] # P(X=1, Y=1, Z=0) = 0.1, P(X=1, Y=1, Z=1) = 0.1
'''
print(f"The sum of the probabilities is {np.sum(random)}")
# output should be 1.0
```

```
P(X=0, Y=0, Z=0) = 0.14150538531589146
P(X=0, Y=0, Z=1) = 0.15714332162172975
P(X=0, Y=1, Z=0) = 0.08506851776783461
P(X=0, Y=1, Z=1) = 0.15031249911514194
P(X=1, Y=0, Z=0) = 0.01069206973887842
P(X=1, Y=0, Z=1) = 0.1470665786939664
P(X=1, Y=1, Z=0) = 0.16027735240348184
P(X=1, Y=1, Z=1) = 0.14793427534307552
The sum of the probabilities is 1.0
```

0.2.1 PLAYGROUND

NB: This is just me trying to understand my probability distribution, ignore the next cell

[4]:
```
# Marginalize of X
P_X = np.sum(random, axis=(1, 2))
print(f"P(X = 0) = {P_X[0]}")
print(f"P(X = 1) = {P_X[1]}")
print(f"The sum of the probabilities is {np.sum(P_X)}")

# Marginalize of Y
P_Y = np.sum(random, axis=(0, 2))
print(f"P(Y = 0) = {P_Y[0]}")
print(f"P(Y = 1) = {P_Y[1]}")
print(f"The sum of the probabilities is {np.sum(P_Y)}")

# Marginalize of Z
P_Z = np.sum(random, axis=(0, 1))
print(f"P(Z = 0) = {P_Z[0]}")
print(f"P(Z = 1) = {P_Z[1]}")
print(f"The sum of the probabilities is {np.sum(P_Z)}")

# Probability of Y given X
joint_XY = np.sum(random, axis=2) # P(X, Y) - remove the influence Z then use␣
 ↪conditional probability formula
P_Y_given_X = joint_XY / P_X[:, None] # P(Y | X) = P(X, Y) / P(X) # NOTE: Here␣
 ↪we are creating the conditional probability table between a child and its␣
 ↪parent
print(f"P(Y = 0 | X = 0) = {P_Y_given_X[0, 0]}")
```

```
print(f"P(Y = 1 | X = 0) = {P_Y_given_X[0, 1]}")
print(f"P(Y = 0 | X = 1) = {P_Y_given_X[1, 0]}")
print(f"P(Y = 1 | X = 1) = {P_Y_given_X[1, 1]}")
print(f"The sum of the probabilities is {np.sum(P_Y_given_X, axis=1)}")

# Probability of Z given Y
joint_YZ = np.sum(random, axis=0) # P(Y, Z) - remove the influence X then use
 ↪conditional probability formula
P_Z_given_Y = joint_YZ / P_Y[:, None] # P(Z | Y) = P(Y, Z) / P(Y) # NOTE: Here
 ↪we are creating the conditional probability table between a child and its
 ↪parent
print(f"P(Z = 0 | Y = 0) = {P_Z_given_Y[0, 0]}")
print(f"P(Z = 1 | Y = 0) = {P_Z_given_Y[0, 1]}")
print(f"P(Z = 0 | Y = 1) = {P_Z_given_Y[1, 0]}")
print(f"P(Z = 1 | Y = 1) = {P_Z_given_Y[1, 1]}")
print(f"The sum of the probabilities is {np.sum(P_Z_given_Y, axis=1)}")
```

```
P(X = 0) = 0.5340297238205978
P(X = 1) = 0.4659702761794022
The sum of the probabilities is 1.0
P(Y = 0) = 0.45640735537046606
P(Y = 1) = 0.5435926446295339
The sum of the probabilities is 1.0
P(Z = 0) = 0.3975433252260864
P(Z = 1) = 0.6024566747739136
The sum of the probabilities is 1.0
P(Y = 0 | X = 0) = 0.559236112928331
P(Y = 1 | X = 0) = 0.440763887071669
P(Y = 0 | X = 1) = 0.3385594671968014
P(Y = 1 | X = 1) = 0.6614405328031986
The sum of the probabilities is [1. 1.]
P(Z = 0 | Y = 0) = 0.3334684537045446
P(Z = 1 | Y = 0) = 0.6665315462954554
P(Z = 0 | Y = 1) = 0.45134140904081427
P(Z = 1 | Y = 1) = 0.5486585909591857
The sum of the probabilities is [1. 1.]
```

(b) Write a code that verifies whether a distribution is a valid BN distribution.

[5]:
```
import numpy as np

def check_bn_validity(P):
    """
    Checks if P(X, Y, Z) factorizes as P(X) P(Y|X) P(Z|Y)
    """
    P_X = np.sum(P, axis=(1, 2))  # Marginal P(X)

    # Conditional P(Y | X) = P(X, Y) / P(X)  where P(X, Y) = sum over Z
```

```python
    P_Y_given_X = np.sum(P, axis=2) / P_X[:, None]  # Shape (2,2)

    # Conditional P(Z | Y) = P(Y, Z) / P(Y) where P(Y, Z) = sum over X
    P_Z_given_Y = np.sum(P, axis=0) / np.sum(P, axis=0).sum(axis=1,␣
↪keepdims=True)  # Shape (2,2)

    # Reconstruct P(X, Y, Z) using the BN factorization
    P_reconstructed = np.zeros((2, 2, 2))
    for x in range(2):
        for y in range(2):
            for z in range(2):
                P_reconstructed[x, y, z] = P_X[x] * P_Y_given_X[x, y] *␣
↪P_Z_given_Y[y, z]

    return np.allclose(P, P_reconstructed, atol=1e-5)

# Example test with a random probability table
random_P = np.random.rand(2, 2, 2)
random_P /= np.sum(random_P)  # Normalize to ensure it's a valid joint␣
↪probability distribution

print(f"Is the random distribution a valid BN factorization?␣
↪{check_bn_validity(random_P)}")
```

```
Is the random distribution a valid BN factorization? False
```

(c) Using your code, generate 10000 distributions and compute the fraction of distributions that are valid BN distributions.

[6]:
```python
def estimate_valid_bn_fraction(num_samples=10000):
    """Generates num_samples distributions and computes the fraction that are␣
↪valid BN distributions."""
    valid_count = 0
    for _ in range(num_samples):
        P = generate_random_distribution()
        if check_bn_validity(P):
            valid_count += 1
    return valid_count / num_samples

valid_fraction = estimate_valid_bn_fraction()
print(f"Fraction of valid BN distributions: {valid_fraction:.4f}")
```

```
Fraction of valid BN distributions: 0.0000
```