Problem2

December 16, 2021

1 Problem 2

We will do these calculations here in python, in order to avoid having to type them out in a calculator every time and also to avoid mistakes due to that.

[1]:
```
import numpy as np
```

[2]:
```
data_list = [
    0.162,
    0.144,
    0.074,
    0.220,
    0.194,
    0.062,
    0.044,
    0.100
]
p_abc = np.array(data_list).reshape(2,2,2)
a = 0
b = 1
c = 2
```

[3]:
```
print(p_abc)
```

```
[[[0.162 0.144]
  [0.074 0.22 ]]

 [[0.194 0.062]
  [0.044 0.1  ]]]
```

Sanity check:

[4]:
```
print(f"Sum of probabilities, should be 1: {p_abc.sum()}")
```

```
Sum of probabilities, should be 1: 1.0
```

1.1 calculate $p(a)$

Now we can easily compute the marginal $p(a)$ by summing over all values where $a == 0$ and $a == 1$.

[5]:
```
p_a = p_abc.sum(axis=(c,b), keepdims=True)
p_a
```

[5]:
```
array([[[0.6]],

       [[0.4]]])
```

1.2 calculate $p(c|a)$

We can do something similar for the conditional distributions:

We sum (marginalize) over b, since we do not care about b in $p(c|a)$ and then normalize for c in order to make it a conditional distribution.

[6]:
```
p_c_a = p_abc.sum(axis=b, keepdims=True)
p_c_a /= p_c_a.sum(axis=c, keepdims=True)
p_c_a
```

[6]:
```
array([[[0.39333333, 0.60666667]],

       [[0.595     , 0.405     ]]])
```

1.3 calculate $p(b|a, c)$

This time we must not marginalize over any variable, since $p(b|a, c)$ depends on all variables, but we still normalize over b to make it a probability distribution.

[7]:
```
p_b_ac = p_abc.copy()
p_b_ac /= p_b_ac.sum(axis=b, keepdims=True)
p_b_ac
```

[7]:
```
array([[[0.68644068, 0.3956044 ],
        [0.31355932, 0.6043956 ]],

       [[0.81512605, 0.38271605],
        [0.18487395, 0.61728395]]])
```

Now, that we have the marginal/conditional probability distributions, we can multiply them up and see is we recover $p(a, b, c)$:

1.4 recover $p(a, b, c)$

[8]:
```
p_abc_candidate = p_a * p_c_a * p_b_ac
```

[9]:
```
print("Original distribution")
print(p_abc)
print("\n\n\n")
print("Recovered distribution")
print(p_abc_candidate)
print("\n")
```

```
Original distribution
[[[0.162 0.144]
  [0.074 0.22 ]]

 [[0.194 0.062]
  [0.044 0.1  ]]]




Recovered distribution
[[[0.162 0.144]
  [0.074 0.22 ]]

 [[0.194 0.062]
  [0.044 0.1  ]]]
```

And finally we do a numerical comparison:

[10]:
```
is_correctly_recovered = np.allclose(p_abc, p_abc_candidate)
print(
    f"The original distribution has{' not' if not is_correctly_recovered else␣
 ↪''} been recovered correctly"
)
```

```
The original distribution has been recovered correctly
```