Marginals

January 24, 2022

1 Problem 2

We will calculate the required distributions here in python, to avoid calculation errors:

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

First we recreate the given distribution

[2]:
```
p_dcs = np.array([
    0.06,
    0.13,
    0.04,
    0.23,
    0.06,
    0.17,
    0.04,
    0.27
]).reshape(2,2,2)
```

Next we can do a sanity check, to see if we have entered the values incorrectly:

[3]:
```
print(f"Sum of probabilities (should be 1): {p_dcs.sum()}")
```

```
Sum of probabilities (should be 1): 1.0
```

We can also define constants for the letters corresonding to the dimensions of the distribution, to make marginalization more readable:

[4]:
```
d = 0
c = 1
s = 2
```

1.1 Task a) $p(c)$

This can be easily done, by summing over everything but c:

[5]:
```
p_c = p_dcs.sum(axis=(d,s))
print(p_c)
```

```
[0.42 0.58]
```

So we have $p(c = 0) = 0.42$ and $p(c = 1) = 0.58$

1.2 Task b) $p(d)$

This can be done exactly the same way as before, but now summing over everything but d:

[6]:
```
p_d = p_dcs.sum(axis=(c,s))
print(p_d)
```

```
[0.46 0.54]
```

So we have $p(d = 0) = 0.46$ and $p(d = 1) = 0.54$

1.3 Task c) $p(d, c)$

This can also be done as before, except, now we have to only sum over s, since we want to keep c and d:

[7]:
```
p_dc = p_dcs.sum(axis=s)
print(p_dc)
```

```
[[0.19 0.27]
 [0.23 0.31]]
```

So we have: $p(c = d, c = 0) = 0.19$, $p(d = 0, c = 1) = 0.27$, $p(d = 1, c = 0) = 0.23$, and $p(d = 1, c = 1) = 0.31$.

1.4 Task d) $p(s|d, c)$

This is going to look similar to the full joint $p(d, c, s)$, except, that it is normalized differently, since it is only a probability distribution in s and **not** in c and d. So we can calculate this by dividing this by corresponding marginal from $p(d, c)$:

We could use the results from above, but for numpy to know what to do, it makes sense to keep the dimensons (so keep $p(d, c)$ three dimensional, since $p(d, c, s)$ is also three dimensional). This is why we recompute it:

[8]:
```
p_dc_keepdims = p_dcs.sum(axis=s, keepdims=True)
print(p_dc_keepdims)
```

```
[[[0.19]
  [0.27]]

 [[0.23]
  [0.31]]]
```

Note that this is the same distribution as above, just in a different format for numpy broadcasting to work nicely.

Now we can devide by this:

[9]:
```
p_s_dc = p_dcs / p_dc_keepdims
print(p_s_dc)
```

```
[[[0.31578947 0.68421053]
  [0.14814815 0.85185185]]

 [[0.26086957 0.73913043]
  [0.12903226 0.87096774]]]
```

This gives us the following results:

[10]:
```
d_max, c_max, s_max =  p_s_dc.shape

for d_value in range(d_max):
    for c_value in range(c_max):
        for s_value in range(s_max):
            print(f"p(d = {d_value}, c = {c_value}, s = {s_value}) =␣
 ↪{p_s_dc[d_value, c_value, s_value]}")
```

```
p(d = 0, c = 0, s = 0) = 0.3157894736842105
p(d = 0, c = 0, s = 1) = 0.6842105263157895
p(d = 0, c = 1, s = 0) = 0.14814814814814814
p(d = 0, c = 1, s = 1) = 0.8518518518518519
p(d = 1, c = 0, s = 0) = 0.2608695652173913
p(d = 1, c = 0, s = 1) = 0.7391304347826088
p(d = 1, c = 1, s = 0) = 0.12903225806451613
p(d = 1, c = 1, s = 1) = 0.870967741935484
```

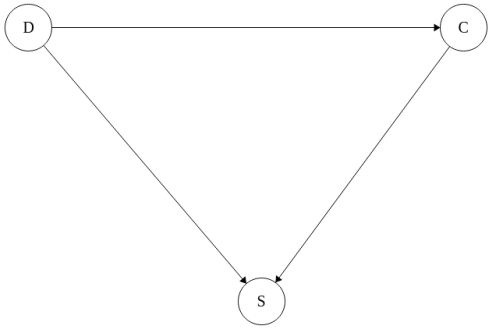
Note, that this is a probability distribution in s only, which can be seen since summing over s gives 1 in every case:

[11]:
```
print(p_s_dc.sum(axis=s))
```

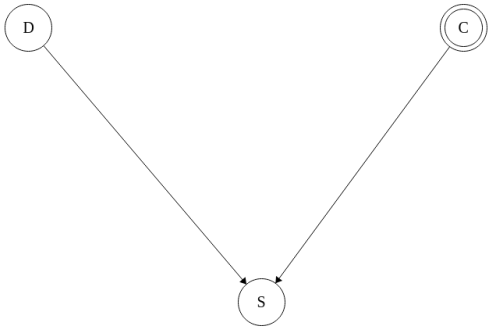
```
[[1. 1.]
 [1. 1.]]
```

1.5 Task d) using *do* calculus

Here now the graphical model becomes important: The original graphical model is:



By using the do-operator on whether the chief surgeon does the surgery, we have do delete the arrow from d to c. This results in the following new graph (where a double circle refers to the usage of the do-operator):



This means, that the original factorization

$$p(d, c, s) = p(d) \cdot p(c|d) \cdot p(s|d, c)$$

now changes to

$$p(d, c, s) = p(d) \cdot p(c) \cdot p(s|d, c)$$

since c is now independent of d. This way we can easily calculate the new joint, since we already have all the required distributions. As above, however, we will have to recalculate them with dimensions kept, for numpy to work:

[12]:
```
p_c_keepdims = p_dcs.sum(axis=(d,s), keepdims=True)
p_d_keepdims = p_dcs.sum(axis=(c,s), keepdims=True)
# p_s_dc can be left as is, since it is already three dimensional
```

Now we can calculate the new joint after using the do-operator by simply multiplying as given in the formula above (this is automatically done correcly by numpy, since we have the correct dimensions):

[13]:
```
p_dcs_new = p_d_keepdims * p_c_keepdims * p_s_dc
print(p_dcs_new)
```

```
[[[0.06101053 0.13218947]
  [0.03952593 0.22727407]]

 [[0.05916522 0.16763478]
  [0.0404129  0.2727871 ]]]
```

Since this is mathematically not necessarily a probability distribution yet, we have to check if it sums to 1 and force it to if necessary:

[14]:
```
print(p_dcs_new.sum())
```

```
1.0
```

This seems to be fine, so we can leave it as it is.

[15]:
```
d_max, c_max, s_max =  p_dcs_new.shape

for d_value in range(d_max):
    for c_value in range(c_max):
        for s_value in range(s_max):
            print(f"p(d = {d_value}, c = {c_value}, s = {s_value}) =␣
 ↪{p_dcs_new[d_value, c_value, s_value]}")
```

```
p(d = 0, c = 0, s = 0) = 0.06101052631578948
p(d = 0, c = 0, s = 1) = 0.13218947368421055
p(d = 0, c = 1, s = 0) = 0.03952592592592593
p(d = 0, c = 1, s = 1) = 0.2272740740740741
p(d = 1, c = 0, s = 0) = 0.059165217391304356
p(d = 1, c = 0, s = 1) = 0.1676347826086957
p(d = 1, c = 1, s = 0) = 0.04041290322580646
p(d = 1, c = 1, s = 1) = 0.2727870967741936
```

For better readability, we can also convert the results to percentages and round to three digits:

[16]:
```
d_max, c_max, s_max =  p_dcs_new.shape

for d_value in range(d_max):
    for c_value in range(c_max):
        for s_value in range(s_max):
            print(f"p_new(d = {d_value}, c = {c_value}, s = {s_value}) =␣
 ↪{p_dcs_new[d_value, c_value, s_value]:2.1%}")
```

```
p_new(d = 0, c = 0, s = 0) = 6.1%
p_new(d = 0, c = 0, s = 1) = 13.2%
p_new(d = 0, c = 1, s = 0) = 4.0%
p_new(d = 0, c = 1, s = 1) = 22.7%
p_new(d = 1, c = 0, s = 0) = 5.9%
```

```
p_new(d = 1, c = 0, s = 1) = 16.8%
p_new(d = 1, c = 1, s = 0) = 4.0%
p_new(d = 1, c = 1, s = 1) = 27.3%
```

We can also look at the difference to before:

[17]:
```
d_max, c_max, s_max =  p_dcs_new.shape

for d_value in range(d_max):
    for c_value in range(c_max):
        for s_value in range(s_max):
            print(f"p(d = {d_value}, c = {c_value}, s = {s_value}) - p_new(d =␣
 ↪{d_value}, c = {c_value}, s = {s_value}) = {(p_dcs[d_value, c_value, s_value]␣
 ↪- p_dcs_new[d_value, c_value, s_value]):2.1%}")
```

```
p(d = 0, c = 0, s = 0) - p_new(d = 0, c = 0, s = 0) = -0.1%
p(d = 0, c = 0, s = 1) - p_new(d = 0, c = 0, s = 1) = -0.2%
p(d = 0, c = 1, s = 0) - p_new(d = 0, c = 1, s = 0) = 0.0%
p(d = 0, c = 1, s = 1) - p_new(d = 0, c = 1, s = 1) = 0.3%
p(d = 1, c = 0, s = 0) - p_new(d = 1, c = 0, s = 0) = 0.1%
p(d = 1, c = 0, s = 1) - p_new(d = 1, c = 0, s = 1) = 0.2%
p(d = 1, c = 1, s = 0) - p_new(d = 1, c = 1, s = 0) = -0.0%
p(d = 1, c = 1, s = 1) - p_new(d = 1, c = 1, s = 1) = -0.3%
```

Here we can see, that overall the change has been pretty small. The difference is, that the probability of the chief operating during daytime has been decreased, as has the probability of someone else operating at night. Inversely, the probability of the chief operaing at night has been slightly increased as has someone else operating during daytime.