6CCS3AIN & 7CCSMAIN, 2018, Tutorial 03 (Version 1.2)

Several of these questions make use of the Bayesian network in Figure 1.

1. Write down a Bayesian network that captures the knowledge that (a) Smoking causes cancer; and (b) Smoking causes bad breath. Given the information that:

$$P(cancer|smoking) = 0.6$$
$$P(badBreath|smoking) = 0.95$$

   and $P(smoking) = 0.2$ use a Naive Bayes model to determine $P(smoking, cancer, badBreath)$.

   Assume that all three variables are binary: $Smoking$ has values $smoking$ and $\neg smoking$, $Cancer$ has values $cancer$ and $\neg cancer$, and $BadBreath$ has values $badBreath$ and $\neg badBreath$.

2. Write down a Bayesian network that captures the knowledge that (a) a Late Start can cause a student to Fail their Project, and (b) Ignoring Advice can cause a student to Fail their Project. Use the binary variables $LateStart$, with values l and $\neg l$, $IgnoreAdvice$, with values i and $\neg i$, and $FailProject$, with values f and $\neg f$.

   You know that $LateStart$ and $IgnoreAdvice$ are non-interacting cuases of $FailProject$, so use a Noisy-Or model to build the conditional probability table relating the three variables from:

$$P(f|l) = 0.7$$
$$P(f|i) = 0.8$$

3. How many numbers do we need to specify all the necessary probability values for the network in Figure 1?

   How many would we need if there were no conditional independencies (if we didn't have the network)?

4. Compute the joint probability $P(m, \neg t, h, s, \neg c)$

5. Use the enumeration algorithm to compute the probability of $P(m|h, s)$.

6. Use prior sampling to compute the joint probability over m, t, h, s and c.

   Use the sequence of random numbers in Table 1. Give the results of the first 5 samples only.

   If you need more random numbers than exist in Table 1, start that sequence again from the beginning.

7. Use rejection sampling to compute $P(m|h, s)$

   Use the sequence of random numbers in Table 1, starting at the beginning of the sequence. This time you should show the results of the first 5 samples that aren't rejected.

   If you need more random numbers than exist in Table 1, start that sequence again from the beginning.

8. Use importance sampling to compute $P(m|h, s)$.

   Use the sequence of random numbers in Table 1, starting at the beginning of the sequence, and again report the results of the first five samples.

   If you need more random numbers than exist in Table 1, start that sequence again from the beginning.

9. Compute $P(m|h, s)$ using Gibbs sampling.

   Use the sequence of random numbers in Table 1, and give the results of the first 5 samples only.

   If you need more random numbers than exist in Table 1, start that sequence again from the beginning.

10. For the optional computational part of the tutorial, download the file `wetGrass.py` from KEATS.

    This provides code, using `pomegranate`[1], that implements the "Wet Grass" example from Lecture 3 (slide 46 etc.). You can run the example using:

[1]For details on `pomegranate` see Tutorial 2.

python wetGrass.py

That should give you this as output:

```
Evidence is:  Grass is wet.

Here's the prediction:
Cloudy, Sprinkler Off, Rain, Wet grass.
And by the numbers:
Cloudy:  (('c', 0.6010250059300832), ('l', 0.3989749940699168))
Sprinkler:  (('f', 0.5976922222293125), ('n', 0.40230777777068755))
Rain:  (('r', 0.7772459441092383), ('n', 0.2227540558907617))
```

The first part of `wetGrass.py` sets up the Bayesian network. This is a straightforward extension of what we had for `underwear.py`.

The second part then runs a query on the model. The variable `scenario`:

scenario = [[None, None, None, 'w']]

specifies the observed values (the evidence), if any, of the variables in the model — the order of the variable is the order specified in the `model.add_states()` command. Here we are setting the variable `WetGrass` to value `w`, denoting that the grass is wet, and not saying anything about the values of `Cloudy`, `Sprinkler` and `Rain`. This:

```
Evidence is:  Grass is wet.
```

reflects the input back — code does this in a somewhat general way[2].

The code uses two methods to do inference given the evidence. First `model.predict()` identifies the most likely (highest probability) value of each variable in the model:

```
Here's the prediction:
Cloudy, Sprinkler Off, Rain, Wet grass.
```

and then `model.predict_proba()` is used to generate the probabilities of all the values of the non-evidence variables:

```
And by the numbers:
Cloudy:  (('c', 0.6010250059300832), ('l', 0.3989749940699168))
Sprinkler:  (('f', 0.5976922222293125), ('n', 0.40230777777068755))
Rain:  (('r', 0.7772459441092383), ('n', 0.2227540558907617))
```

It turns out that `model.predict_proba()` handles evidence variables differently to non-evidence variables, and so the code doesn't output any information about evidence variables (we already know the value taken by evidence variables after all).

The final part of the code does some minimal formatting on what is generated by `model.predict()` and `model.predict_proba()` to produce the output above.

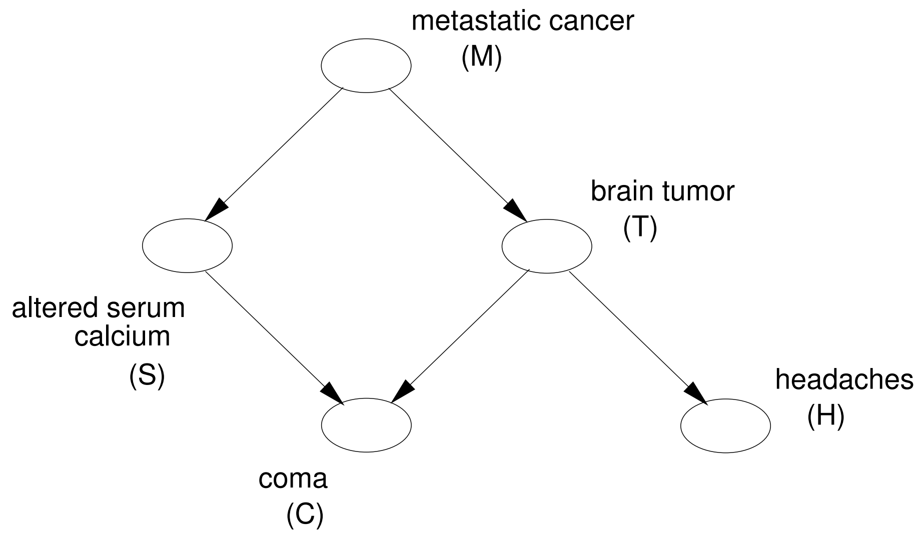
(a) Use `wetGrass.py` to calculate the probability of the variables in the model if:
   i. The grass is wet and the sprinkler is on.
   ii. The sprinkler is on and the sky is cloudy.
   iii. It is raining.

(b) Use `pomegranate` to build the model in Figure 1. Use this model to calculate $P(m|h, s)$.

[2]The documentation for `pomegranate` is not great, but from what I can tell, there is no function that gives access to the states in a model once the model has been built, so we can't retrieve the variables, and their names and values in a general way. (We could build our own structures to keep track of this, of course). Hence the way that they are hard-coded.

0.14	0.57	0.01	0.43	0.59	0.50	0.12	0.54	0.97	0.51	0.49	0.67	0.96	0.55	0.89	0.21	0.34

Table 1: Some random numbers between 0 and 1.



$P(M)$
0.1

M	$P(S \mid M)$
T	0.8
F	0.2

M	$P(T \mid M)$
T	0.7
F	0.1

S	T	$P(C \mid S, T)$
T	T	0.95
T	F	0.85
F	T	0.85
F	F	0.01

T	$P(H \mid T)$
T	0.9
F	0.7

Figure 1: The example Bayesian network.