CMPSC 448: Machine Learning. Take-home Final.
May 7, 12:20pm – May 8, 12:20pm

1. Academic Integrity

- You may not discuss the final with anybody
- You may not post any questions related to the final on the internet. The ONLY exception is the course Piazza page
- You cannot look at anyone else's code. You may not let anyone look at your code.
- You may not import anything except for the fractions library

2. How the test works

- Submit a blank final.py file to the gradescope assignment titled "Download Your Final From Here".
- Autograder for that assignment will print out the function definitions and exact questions that you must answer.
- Copy and paste this into your final.py, then fill in your code.
- Submit your final code to the gradescope assignment titled "Submit Your Final Here"

3. Coding Style

- All code (except import statements and constants) should be inside functions.
- Code must have comments and any constants should be stored in a variable defined near the top of your file.

4. Bayesian Networks

4.1. **Numeric Stability.** In this assignment, to avoid rounding issues, you will need to use Fraction datatypes in python. Here is an example of their usage:

```
from fractions import Fraction as frac
half = frac(1,2)
```

If you choose not to use fraction datatypes, you might get a correct answer marked as incorrect due to rounding issues.

4.2. **Bayesian Network Parameters.** The parameters of a Bayesian network will be passed to your code through a parameter **bn**. It will be a class and its usage is described as follows: suppose the Bayesian network has directed edges $(a, b), (a, c), (b, d), (c, d)$. The parameters of this network are the conditional probabilities $P(a), P(b|a), P(c|a), P(d|b, c)$. The corresponding **bn** variable will be defined something like this:

```
import numpy as np
from fractions import Fraction as frac

class BayesNet1:
   def __init__(self, seed, k=10):
       prng = np.random.RandomState(seed)
       self.prob_a = frac(prng.randint(1, 2**k), 2**k) # P(a=1)
       self.prob_b = {(1,): frac(prng.randint(1, 2**k), 2**k), # P(b=1 | a=1)
                 (0,): frac(prng.randint(1, 2**k), 2**k) # P(b=1 | a=0)
                }
       self.prob_c = {(1,): frac(prng.randint(1, 2**k), 2**k), # P(c=1 | a=1)
                 (0,): frac(prng.randint(1, 2**k), 2**k) # P(c=1 | a=0)
                }
       self.prob_d = {
                 (0, 0): : frac(prng.randint(1, 2**k), 2**k), # P(d=1 | b=0, c=0)
                 (0, 1): : frac(prng.randint(1, 2**k), 2**k), # P(d=1 | b=0, c=1)
                 (1, 0): : frac(prng.randint(1, 2**k), 2**k), # P(d=1 | b=1, c=0)
                 (1, 1): : frac(prng.randint(1, 2**k), 2**k), # P(d=1 | b=1, c=1)
                }

    # the * forces you to name the parameters after the *
    # so bn.a(value=1) is a valid function call
    # but bn.a(1) will return an error that
    # looks like: TypeError: a() takes 1 positional argument but 2 were given
    def a(self, *, value): #returns P(a=value)
        if value == 1:
            return self.prob_a
        else:
            return 1-self.prob_a

    def b(self, *, value, a): #returns P(b=value | a)
        tmp = self.prob_b[(a,)]
        if value == 1:
            return tmp
        else:
            return 1-tmp

    def c(self, *, value, a): #returns P(c=value | a)
        tmp = self.prob_c[(a,)]
        if value == 1:
            return tmp
        else:
            return 1-tmp

    def d(self, *, value, b, c): #returns P(d=value | b, c)
        tmp = self.prob_d[(b,c)]
        if value == 1:
            return tmp
        else:
            return 1-tmp


# example usage
bn = BayesNet1()
```

```
# get parameter p(a=1)
bn.a(value=1) # must call with arg names, bn.a(1) is incorrect
bn.a(value=0) # get parameter p(a=0)

bn.d(value=0, b=1, c=0) # get parameter P(d=0 | b=1, c=0)

bd.a(value=1, d=2) #throws error because P(a|d) is not a parameter
```

If a parameter is not needed for a particular problem, using it may throw an exception. For example, for the Bayesian network in the code above, $P(a, b)$ can be computed without using the parameter $P(d \mid b, c)$. That is, after you write $P(a, b)$ in terms of the network parameters and simplify, you will notice that $P(d|b, c)$ is not used at all. Thus our implementation of **bn** might not define the function **bn.d(value, b, c)**. This is used to test that you simplified the expression correctly.

4.3. **Types of Questions.** There will be two types of questions:

(1) Probability calculations: given network parameters, compute probabilities such as $P(A = 1 \mid B = 0, C = 1)$.

(2) D-separation. The questions might ask you if A is conditionally independent of D given E. You will write a function that returns **result**, **pathverdict** where **result** is the answer (are they conditionally independent?) and **pathverdict** looks like:

```
[
  (('a', 'b', 'd'), False)
  (('a', 'e', 'd'), False)
]
```

each element in the list is a tuple. The first part of the tuple describes the path (e.g., a,e,d is an undirected path from a to d) and the second part of the tuple tells us if that path is blocked or not. Make sure **pathverdict** contains <u>all</u> of the appropriate undirected paths. A path <u>cannot</u> repeat nodes (so a e a e d is not a path). You have to hard-code the appropriate paths, and results. In other words, for d-separations questions, your functions should look like:

```
def question0():
   parthverdict = [
      (('a', 'b', 'd'), False)
      (('a', 'e', 'd'), False)
    ]
   result = False
   return result, pathverdict
```

In this case, this answer indicates that the first path is not blocked and the second path is not blocked.

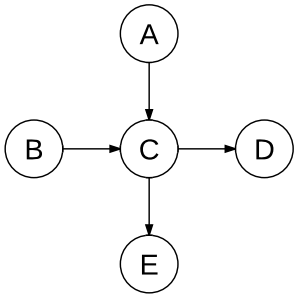

FIGURE 1. Bayesian Network 1: All nodes are upper-case

Question 1 (5 pts)**.** *This is a d-separation question using the Bayesian Network in Figure 1. In final.py, fill in the function question1 as instructed so that it returns result, pathverdict.*

Question 2 (5 pts)**.** *This is a probability question using the Bayesian Network in Figure 1. In final.py, fill in the function question2 as instructed. Make sure it returns a frac (not float).*

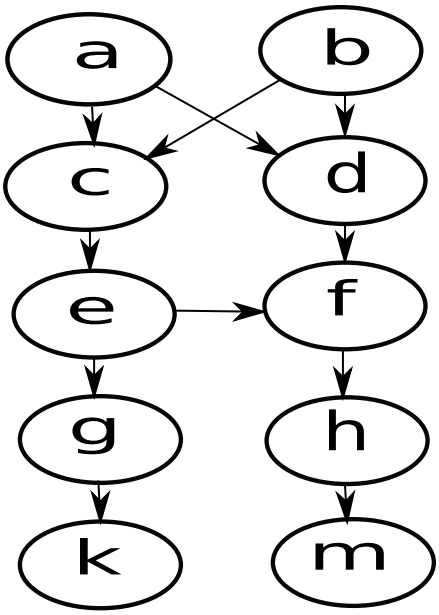

FIGURE 2. Bayesian Network 2: All nodes are lower-case

Question 3 (10 pts)**.** *This is a d-separation question using the Bayesian Network in Figure 2. In final.py, fill in the function question3 as instructed so that it returns result, pathverdict.*

Question 4 (10 pts)**.** *This is a d-separation question using the Bayesian Network in Figure 2. In final.py, fill in the function question4 as instructed so that it returns result, pathverdict.*

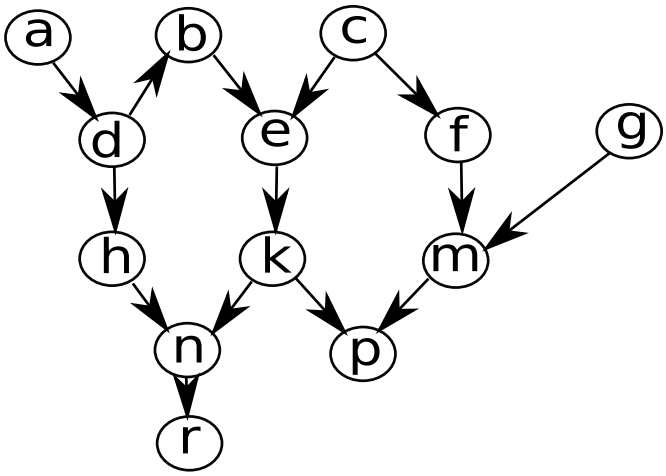

FIGURE 3. Bayesian Network 3, All nodes are lower-case

Question 5 (10 pts)**.** *This is a d-separation question using the Bayesian Network in Figure 3. In final.py, fill in the function question5 as instructed so that it returns result, pathverdict.*

Question 6 (10 pts)**.** *This is a d-separation question using the Bayesian Network in Figure 3. In final.py, fill in the function question6 as instructed so that it returns result, pathverdict.*

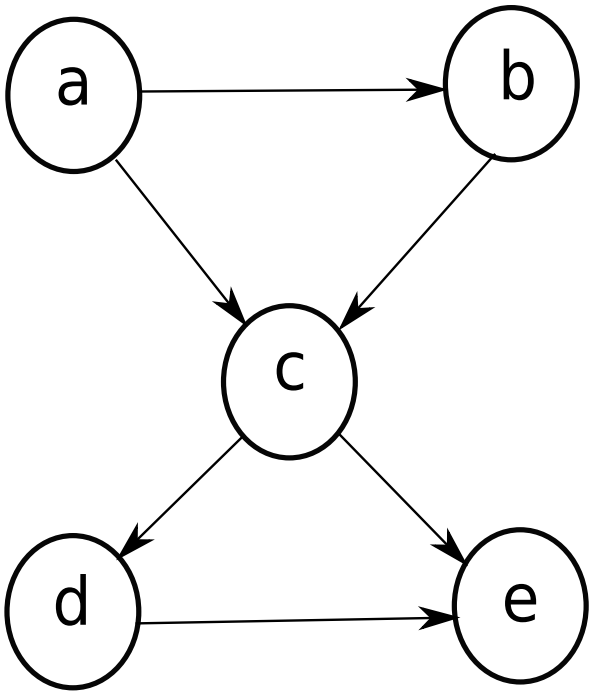

FIGURE 4. Bayesian Network 4, All nodes are lower-case

Question 7 (10 pts)**.** *This is a probability question using the Bayesian Network in Figure 4. In final.py, fill in the function question7 as instructed. Make sure it returns a frac (not float) and does not use any unnecessary network parameters.*

Question 8 (10 pts)**.** *This is a probability question using the Bayesian Network in Figure 4. In final.py, fill in the function question8 as instructed. Make sure it returns a frac (not float) and does not use any unnecessary network parameters.*

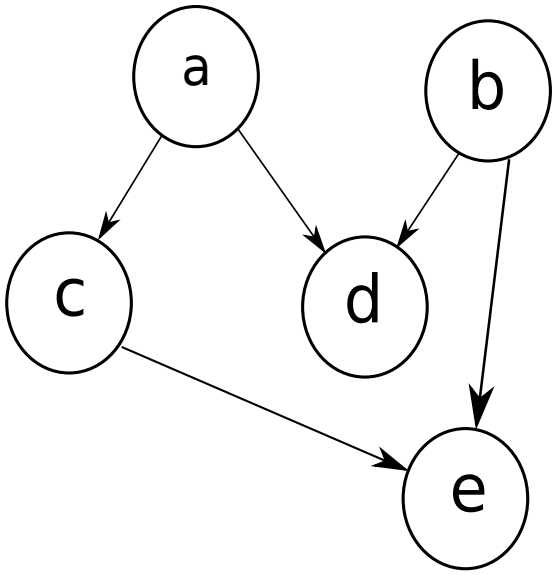

FIGURE 5. Bayesian Network 5, All nodes are lower-case

Question 9 (10 pts)**.** *This is a probability question using the Bayesian Network in Figure 5. In final.py, fill in the function question9 as instructed. Make sure it returns a frac (not float) and does not use any unnecessary network parameters.*

Question 10 (10 pts)**.** *This is a probability question using the Bayesian Network in Figure 5. In final.py, fill in the function question10 as instructed. Make sure it returns a frac (not float) and does not use any unnecessary network parameters.*