CSE 368: Assignment 4

Introduction

In this assignment you will be tested on your understanding of Bayesian Networks by implementing the Enumeration Algorithm, Rejection Sampling, Likelihood Weighting; as well as creating a Bayesian Network. All these functions are found in the inference.py file while helper functions as well as class definitions are in bayesnets.py.

bayesnets.py

Before we talk about what functions you have to implement, we will look into the classes and their helpful functions in this file.

- Class ProbDist()
    - The use of this class is solely for creating discrete probability distributions. The constructor as well as the normalize() method are what you will be using for the scope of this assignment.

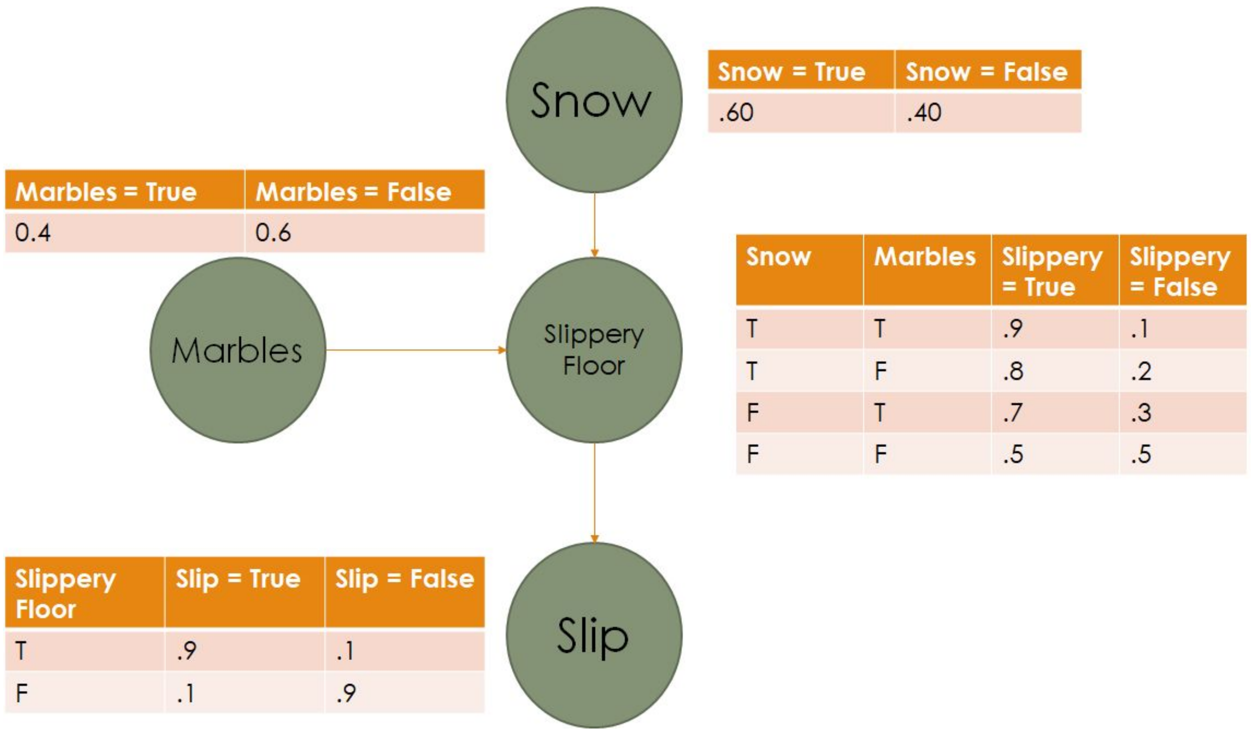
    - When using the constructor, you can specify a string that will be the name of your random variable, and an optional dictionary as the second argument which specifies the probability of values
        - Example: ProbDist('Sleepy', {'T':0.5, 'F':0.5})
        - Example 2: ProbDist('Sleepy')
        - Example 3: ProbDist('Sleepy', {'Low':0.2, 'Medium':0.3, 'High':0.5})

    - The ***normalize()*** method is useful for when you need the probabilities of your values to sum to 1.
        - Example: P=ProbDist('Sleepy', {'Low':0.1, 'Medium':0.2, 'High':0.2}, will return 0.2, 0.4, and 0.4 for low, medium, and high respectively.

- Class BayesNet()
    - This class is used to construct Bayesian Networks by passing in a list containing tuples that are variables in your network. [(node_1), (node_2), …(node_n)]

    - More specifically, each tuple has 4 arguments: a random variable name, domain, parents, and values. So, BayesNet([('Snow', '', '', (0.6, 0.4))]) creates a Bayesian Network with one variable called ‘Snow’, it has no parent, and it has a 60% chance of being true and a 40% chance of being false. Note that if nothing is specified in the second argument, then by default your network has only true values. Say we wanted snow to range from low to high, we would do the following: BayesNet([('Snow', 'low moderate high', '', {‘low’: 0.1, ‘moderate’:0.3, ‘high’:0.6})])

- As another example, to construct the network below:



Snow = True	Snow = False
.60	.40

Marbles = True	Marbles = False
0.4	0.6

Snow	Marbles	Slippery = True	Slippery = False
T	T	.9	.1
T	F	.8	.2
F	T	.7	.3
F	F	.5	.5

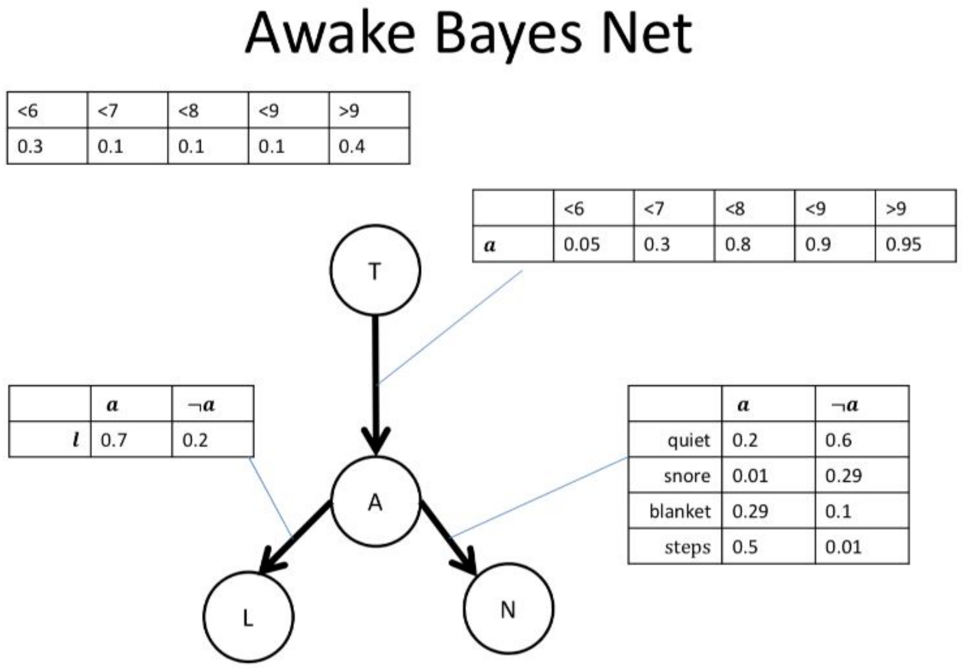
Slippery Floor	Slip = True	Slip = False
T	.9	.1
F	.1	.9

```
slipNet = BayesNet([('Snow', '', '', (0.6, 0.4)),
            ('Marbles', '', '', (0.4, 0.6)),
            ('Slippery_Floor', '','Snow Marbles', {(True, True):(0.9,
0.1), (True, False):(0.8, 0.2),(False,True):(0.7,
0.3),(False,False):(0.5,0.5)}),
            ('Slip', '', 'Slippery_Floor',
{True:(0.9,0.1),False:(0.1,0.9)})])
```

- ***variable_values(self, var)***
  - This method returns the domain of a variable var in the network.
  - Example: slipNet.variable_values(‘Snow’) returns (True, False)
- ***variable_node(self, var)***

        - This method returns the **node** of a variable var in the network. Note that this returns an instance of type BayesNode (which has a method that will be useful for enumeration)
    - ***nodes***
        - This is a list containing all the nodes in the network (they are of type BayesNodes)
    - ***Variables***
        - This is a list containing all the variables in the network (they are strings)
- <u>Class BayesNode</u>
    - This class represents a node in the Bayesian Network. There is only one function you will utilize here.
    - ***p(self, value, evidence)***
        - This function returns the conditional probability P(X=value | parents=parent_values), where parent_values are the values of parents in evidence.
        - Example: slipNet.variable_node(‘Snow’).p(True, {}) returns 0.6. Note that the second argument is an empty dictionary because ‘Snow’ has no parents
        - Example 2: slipNet.variable_node('Slippery_Floor').p(True, {'Snow':True, 'Marbles':False}) will return 0.8
        - **Note:** when you don’t include all the parents of the node in the dictionary, a KeyError will be thrown.

Inference.py



<6	<7	<8	<9	>9
0.3	0.1	0.1	0.1	0.4

	<6	<7	<8	<9	>9
a	0.05	0.3	0.8	0.9	0.95

	a	$\neg a$
l	0.7	0.2

	a	$\neg a$
quiet	0.2	0.6
snore	0.01	0.29
blanket	0.29	0.1
steps	0.5	0.01

- **(20 points) Awake Net:** Given the diagram above, construct it using the BayesNet constructor. The variable should already be initiated, called awakeNet (should be on line 112, if it's not just use CTRL + F to find it) and you will store your solution here. If you get lost, look at the example network constructed above.

- **(30 points) Enumeration Algorithm:** Implement the enumeration algorithm to compute the distribution of a random variable in the network. The function you will be editing is *enumeration_infer(self, X, e=None)*, where X is a string which represents the name of a node in the network, and e the evidence present (which is defaulted to None).
    - You may need to create another function for summing up over all the hidden variables, given the evidence.

- **(30 points) Rejection Sampling:** Implement Rejection Sampling to estimate the distribution of a random variable using N samples and evidence e(If there is no evidence, the default 'self.evidence' is used). The function you will be editing is *rejection_sampling(self, X: str, N: int, e: dict = None)*; in order to get a random sample from the network, call ***self.sample()*** which will return a dictionary where the keys are all the variables in your network and the values are random values for each of the variables (in their respective domains, and excluding evidence variables).

- **(20 points) Likelihood Weighting:** This function actually has two sub functions: *weighted_sample(self,e :dict = None)* computes a weighted sample where the weight is the likelihood an event occurs according to the evidence, and *likelihood_weighting(self, X: str, N: int, e: dict = None)* estimates the probability distribution of a variable X given the evidence e and N samples.