Artificial Intelligence Homework 4
Due date: 3/29 by 11:59pm

In this assignment, you will be implementing 3 approximate inference algorithms for a Bayesian network: direct sampling, rejection sampling, and likelihood weighting.

This is a programming assignment and you may work with a partner if you choose. Click here to download the starter code. You will notice that the code is arranged in packages:

- **tui:** This package contains a textual user interface (tui) that allows the user to execute different probabilistic queries.
- **bn:** This package contains a Node class and a Bayesian network class.
- **util:** This package contains various data structures.

You will notice there are also 2 files with extension ".bn". These files encode the Bayesian networks.

Specifying a Query

When you run the Inference Engine, it will ask you to type in a query. All queries should obey the following conventions:

- Use standard probability notation -- e.g. "p(alarm)" or "p(alarm|burglary)"
- Use the "!" symbol to represent false. For example, "earthquake" represents {Earthquake=true} whereas "!earthquake" represents {Earthquake=false}
- Only evidence variables can be false -- by default, it is assumed that you want the *distribution* over any query variables. For example, the query "p(cloudy | !wet_grass)" is interpreted as "Compute the *distribution* over the variable cloudy given we have observed that the grass is not wet"

Sample queries include:

- p(alarm | !earthquake, burglary)
- p(wet_grass)
- p(wet_grass, cloudy)
- p(earthquake | !alarm)
- p(john_calls | burglary, earthquake, !alarm, mary_calls)

The Textual User Interface Package

The **tui** package contains three classes: InferenceEngine, Query, and Reader. The InferenceEngine is the main class that you should run. It reads in a Bayesian network from file and then continuously prompts the user to enter a query.

The Query class contains all of the pieces of information specified by a query -- i.e., it stores the query variables, the evidence variables, and the value of the evidence variables. You should read through this class carefully since you will need to interact with this class when implementing the 3 approximate inference algorithms. I have purposefully made the instance variables public so that you can directly access them.

Finally, the Reader class is responsible for reading in a Bayesian network from file. It contains only a single public method that takes in a filename and returns a list of nodes. You should not have to interact with this class.

Bayesian Network

A Bayesian network is a directed acyclic graph where each node corresponds to a random variable. The `Node` class represents a single node in a Bayesian network and as such should contain:

1. A name (e.g. "earthquake")
2. A sampled value (either true or false)
3. A set of parent nodes
4. A conditional probability table

The methods for the `Node` class have already been provided although you are required to fill them in.

The `BayesianNetwork` class should contain a list of the nodes in the Bayesian network in topological order. You do not need to sort the nodes into topological order -- the nodes are listed in topological order in the .bn file and the `Reader` class maintains that topological ordering.

The methods for the `BayesianNetwork` class have already been provided although you are required to fill them in. It is in the `BayesianNetwork` class that you will implement the 3 approximate inference algorithms. (Note: for rejection sampling, if you reject all samples, it is okay to return a weighted set where the weight for each event is simply 0)

Representing Events and Distributions

Here is some terminology that we will use to simplify matters:

- **configuration** -- A configuration is one particular assignment of values to n boolean random variables. For example, {Burglary=F, Earthquake=T} is a configuration of 2 boolean random variables.

- **all possible configurations** -- Given n boolean random variables, there are 2^n possible configurations. For example, if n=2 then the set of all possible configurations is given by {TT, TF, FT, FF}

The data structures in the **util** package provide ways of working with configurations. This package contains three classes: AssignmentIterator, BitVector, and WeightedSet. You are responsible for implementing the WeightedSet class although you should read through the other two classes to familiarize yourself with how they work.

The BitVector class

A BitVector contains an array of n boolean values. This is how we represent a configuration. For example, the configuration from above {Burglary=F, Earthquake=T} is represented by a BitVector with size 2 where the first position in the array is set to false, and the second position in the array is set to true.

The BitVector class has methods to get and set each position in the array. It also contains a public static variable TRUE that you should use for variables in the Bayesian network that do not have a parent.

AssignmentIterator

AssignmentIterator is an iterator (i.e. it implements the Iterator interface). This class iterates over all possible configurations. Thus, if you create a new AssignmentIterator with size n=3, each call to next() will return:
111
110
101
100
011
010
001
000
Any place in your code where you need to enumerate all possible configurations, you should use this class.

WeightedSet

Finally, the WeightedSet class is a general-purpose class that maps configurations to numerical values. The meaning of the numerical value depends upon the usage. For example, the numerical value could be a tally (for direct sampling or rejection sampling), or it could be a weight (for likelihood weighting), or it could be a probability (for a conditional probability table).

Your WeightedSet class should contain a mapping from configurations (i.e. BitVectors) to numerical values (Doubles). The constructor should populate the mapping with all possible configurations (and assign each configuration some default numerical value). You will see that the WeightedSet class has methods for adding to the map and getting from the map. It also has methods for modifying (e.g. incrementing or normalizing) the numerical values.

Running Your Code

The main() method is in the InferenceEngine class. This class takes a Bayesian network file (with extension .bn) via the command line. I have provided 2 such Bayesian network files for you however you are welcome to create your own Bayesian network files. A Bayesian network file has the following format:

- The first line is an integer specifying the number of nodes in the Bayesian network
- The remaining lines list the CPTs of the nodes in the Bayesian network in topological order. The first line gives the name of the variable and the name of any parents (separated by a pipe "|")
- The following lines list the rows of the CPT -- i.e. the following lines list p(X = true | parents(X)) for all possible configurations of the parents. Thus, if the node has 2 parents, there should be 4 lines specifiying the CPT.

Grading and Submission

Your assignment will be graded based upon correctness (i.e. I will run each of the 3 inference algorithms on various queries), efficiency (i.e. the time/space complexity of your implementation), as well as its adherence to the Java style guide.

To submit, rename your directory hw4_FirstName_LastName. Then zip your directory and upload it to Moodle. Please remember to rename your directory *before* you zip it.

Last modified: Tue Jul 15 13:34:17 PDT 2014