George Rossney
CSC 242
Project 3 Write-Up
4/15/16

Inference with Bayesian Networks

My Java implementation of Bayesian networks for modeling probability is based on a directed acyclic graph structure. I have created a tree structure to implement a generic graph that can be applied to solving problems. My code uses random variable and conditional probability classes as essential components to solving probability problems represented as Bayesian networks. Each node represents a random variable in the network. A probability problem does not have a clear solution, but it can be used to predict future outcomes. The goal is to be able to design an implementation that accepts the given problem space as input and outputs predictions for possible values for each variable.

Coding Structure

The structure of my project is based off of my BN file, node files, and CPT file, since they all apply to the code for each inference algorithm. My initial plan was to use the the given code as a base in which I could fill in each file with my own code, but I re-wrote each Java file so I could simplify them to understand them. The "Inference" file calls both the parser and "BN" (Bayesian Network) files to parse in network data for evaluation. The other classes are called or contained within the main "BN" file. I struggled with modifying and debugging Prof. Ferguson's parser file, so mine is fairly different. The parser file is important, since it parses the network problems into each algorithm for solving. In my BN file, I have constructed a tree for Bayesian network problems in which nodes are stored in an ArrayList. Each child node is the next sequential node in the DAG. I keep track of the nodes by adding the visited nodes to a new "end" ArrayList. I left Prof. Ferguson's print methods commented out, since I only attempted to use them for testing. My

BNnode file has a similar structure to my BN file, but it builds on it by incorporating random variables into ArrayLists. It then takes those ArrayLists and uses them to construct the conditional probability table (CPT), which should be the printed output of the BNnode.java file.

Exact Inference

In order to solve part one of the Bayesian network problem using exact inference, I have implemented the enumeration algorithm. All possibilities for each variable are evaluated in the space of the given problem. When combining variable predictions, the total value must equal one for a logical probability representation. The problem with enumerating all possibilities for each variable is that it is time consuming and inaccurate. The code in my "EnumerationQuery" file parses in network data and enumerates each node with probability representations. The values are then added to a base of evidence. Methods to relate probability and evidence to the join distribution tree for the inference of the enumeration method are included in my "Exact Inference" java file. The join distribution tree creates a new head node and uses conjunctions to increase tree depth. I used Prof. Ferguson's distrubtion.java file as somewhat of an outline for this algorithm, but it mainly reminded me that the probabilities on a certain random variable must always sum to one.

Approximate Inference

In order to use the approximate inference approach, I have implemented both the rejection sampling and likelihood weighting algorithms from the *AIMA* textbook. Neither algorithm is efficient, but both are much more adequate than using exact inference, with more random results being the trade-off. Both my rejection sampling and likelihood weighting algorithm implementations follow the same ArrayList structure built from parsed network data. Values returned are based on likely outcomes from the range of possible vector outcomes for each random variable. Rejection sampling should ideally generate values based on knowledge from the joint

distribution tree. Only values within the range should be generated. Likelihood weighting should ideally use the same logic as rejection sampling, but remove incorrect cases. Weighting on the probability distribution is achieved by multiplying the current node thru the empty spots and probability table. The likelihood weighting algorithm should be more efficient and accurate than rejection sampling, but it won't be if there are errors in the multiplication and propagation of random variables in the Bayesian network. At small sample sizes, I believe that the performance of both of my approximate inference implementations are very similar.

Parser

The most difficult part of the project for me was handling the given parser file. I wanted to implement Prof. Ferguson's parser since it should output the required network results in the proper format. All of my java files for this project do not contain and errors and compile, but are difficult to test in an IDE due to the parser requirements. To test my project, I used the given command on the command line in terminal. I did achieve different probability results that summed to one for my enumeration exact inference algorithm, but I did not achieve consistent or logical results for both my rejection sampling and likelihood weighting algorithms approximate algorithms.

Testing

To run my project from the command line, the directory has to be my package folder, proj3Rossney. To test on one of the example problems, the format given in the project PDF should be used. The results may not be correct or consistent, but they should demonstrate that I have implemented Bayesian network algorithms that have the capability of generating trees of random variables represented in a conditional probability table.

Sources

I did not have a partner or consult anyone else on any part of this project. I used Professor Ferguson's lecture slides for information on Bayesian networks. I implemented his XMLBIFParser.java file and the structure of his CPT.java file. I also referred to the following outside sources for help:

1. https://dslpitt.org/genie/wiki/Java_Tutorial_1:_Creating_a_Bayesian_Network

   I used this website to try to understand the attributes needed for each node in the network.

2. http://ocw.mit.edu/courses/health-sciences-and-technology/hst-950j-biomedical-computing-fall-2010/lectures-and-readings/MITHST_950JF10_lec14.pdf

   I used these MIT lecture slides to learn more information on each inference algorithm.

3. http://www.cs.cmu.edu/~awm/15781/slides/bayesinf05a.pdf

   I used these CMU lecture slides to try to understand the alarm problem more in depth, but like textbooks and lectures, it just repeats general Bayesian network information.