Uncertain Inference: Bayesian Networks

By Alec. R. Rajeev

CSC 242: Artificial Intelligence

Prof. Ferguson

I. Introduction

The purpose of this project was to build a Bayesian network that can make exact inferences and approximate inferences. The algorithm used for exact inferences was inference by enumeration. The algorithm used for approximate inferences was rejection sampling. The project was done in python 2.7.10 and uses python's built in xml parser to parse the xmlbif files.

II. Design

The program defines various classes similar to the design that Prof. Ferguson sent out in his java code. The first class is the Random Variable class. It contains the name of variable and its domain. The domain is its own Domain class. The domain class contains the list of possible outcomes. This is a simple python because it's possible that the outcomes in the xml file are not simple true or false rather than a more efficient numpy array. The next class is the CPT class. This is a conditional probability table class and is among the most important classes in the program. It contains the random variable that the table is for. It also contains a list of given variables if there are any. This is none (python's null) if there are not any given variables for that particular CPT. The table has one row for every possible combination of values of the given variables. If there are 2 given variables, each with 3 possible values, then

there are 9 rows. A second table called a helper table is built as well. This helper table gives that values of given variables at each row. This is done using python's itertools and the itertools' product. It builds all the possible combinations with the outcomes of the given variables. For this to work properly, it is necessary that the order of the outcomes in the xml file corresponds to the order in the conditional probability table in the xml file. So if the random variable G lists true as it's first outcome, its conditional probability table must list g as true as the first value. Now that the rows of the conditional probability table have been explained, the columns will be addressed. The number of columns corresponds to the number of possible outcomes in the variable for which the conditional probability table is for. In instances where a variable's conditional probability table has no givens, then there is just a single row. The parser is written in python using python's built in xml parser. As the program parses the xml file, it first builds the random variables and domains, then the conditional probability tables. It puts each of these into separate hash tables in order for quick accessing of them later on in the program.

Once the random variables and conditional probability tables have been written imported into the program, the network must be built. The program does this by creating a Node object for every random variable. The node contains a random variable and its conditional probability table. It then puts all the nodes into a hash table for quick accessing. Then it connects all the nodes that need to be connected. It does this by cycling through the probability tables for each node. The givens for each probability table will be the parents for that node. Each node that has children will have a list of other nodes as an attribute. If a node is a root node, then it will have this attribute as none. Then the program sorts these nodes into a topological sort. It does this by using depth first search topological sort algorithm that keeps track of nodes temporarily and permanently being marked.

III. Exact Inference

Now that the network has been built and topologically sorted, the first algorithm can be implemented. The inference by enumeration algorithm was used. First it needs the query variable for which the posterior distribution will be calculated. This is done by grabbing the name of the query variable from the command line arguments and then matching it to the correct random variable object. Then the evidence is needed. This is again taken from the command line arguments. It puts all of the evidence into a python list. Each element of the list is a second python list. This inner python list will have as its first element the name of the evidence variable. The second element of this inner list will be the value of this variable from the evidence. This python list of elements will be used to on numerous occasions further in the program. Then the program cycles through every possible outcome of the query variable and finds the unnormalized probability. It does this by adding that outcome to the list of evidence variables and calculating the probability. Once the calculation is completed, it removes that piece of evidence and adds on a different outcome as evidence. The probability is calculated using enumeration. The function takes in the list of variables, evidence, and network. If the list of variables is empty, then it simply returns 1.0. This is the base case and will eventually happen because at each level of depth for the recursive algorithm the list of variables decreases by one. The function then takes the first element of the list of variables. It puts the rest of the variables into a list called rest. It checks if there is a piece of evidence for that first variable. If there is evidence then it finds the conditional probability for that value of that variable given the evidence, and multiplies that by the enumeration algorithm again except with the rest of the variables as the list of variables. This is important because for the algorithm to terminate it must decrease the number of variables at every level of recursion. If there is not evidence for that first variable then it takes the sum of the conditional

probabilities for every possible outcome of that variable. It does this by taking a deep copy of the evidence. Then it adds onto the evidence an outcome of that first variable. Then it sums up all these conditional probabilities and multiplies them by the recursively called enumeration function like before. All of this is done until all of the variables have been gone through.

Now the particular business of calculating the conditional probability is of some interest. The way this structure is formatted is important. The parameters of the conditional probability table that come into the function are the evidence variables. The program first needs to find the row of the table that matches with the evidence. If the conditional probability table has no givens, then it is just one row and it is simple. If there are multiple rows because there are different combinations for the outcomes of the given variables, then it is more complicated. The program used a second helper table to match the values for the given variables with the correct row. Once the correct row is of the conditional probability table is found, then the column has to be found. This is done by matching the evidence with the domain list of variable for which the conditional probability table represents.

Now that the posterior distribution is found it has to be normalized. This is done by doing 1 divided by the sum of the values.

III. Approximate Inference

The approximate inference algorithm used was rejection sampling. This was done by first creating 1000 (or whichever number was specified) number of samples. These samples were calculated by the randomly assigned conditional probabilities. Each sample was called an Atomic Event. Then each event had a list of Sample Nodes that correspond to each random variable. These sample nodes were used to keep track of variables assigned values for the conditional probabilities. Once all 1000 events have been created then some of them

were rejected. They were rejected if they did not satisfy the evidence. Once all the samples that were not rejected were found, then the program counted the number of occurrences of each possible outcome for the query variable. Then these counts were divided by the total count to normalize the posterior distribution. This is an approximation.