PGM Programming Assignment: Simple BN Knowledge Engineering

1 Overview

Welcome to the course! The goal of this first assignment is for you to gain familiarity with Bayesian networks and to understand how we might compute probability queries in these networks. As such, this assignment has two parts. In the first part of the assignment, you will use the SAMIAM package to design a small Bayesian network for evaluating credit-worthiness. Then, in the second part of the assignment, you will replicate some of the functionality in SAMIAM for answering probability queries in the network using the factor operations discussed in the lectures. You will test your implementation on the network you designed in the first part of the assignment.

2 Engineering a network for credit-worthiness

In the first part of the assignment, you will use the SAMIAM software package to design a Bayesian network for the purpose of predicting credit-worthiness.

2.1 Using SAMIAM

SAMIAM was developed by the Automated Reasoning Group at UCLA to provide a graphical interface for manipulating probabilistic networks on Windows, Linux, or Mac OS. It has extensive functionality for learning and inference in probabilistic networks; however, we will use it only to manipulate a network and view marginals. Please download the program at:

`http://reasoning.cs.ucla.edu/samiam/index.php`

There are separate files for each operating system (Linux, Windows, and Mac), so just use whichever is appropriate for the machine you are on. SAMIAM relies on Java and needs both the Java Runtime Environment (JRE) and the Java Development Kit (JDK). If you do not already have these installed on your machine (or you receive errors that no java executable is found, or the program does not start), you should download them from:

JRE: `http://java.com/en/download/manual.jsp`
JDK: `http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html#javasejdk`

The network we will use is an example network for evaluating credit-worthiness. Load this into the program by going to File, Open, and then opening `Credit_net.net`.

When constructing your network, you will be using **Edit Mode** to add edges to your network. To engage **Edit Mode**, go to the **Mode** menu and simply select **Edit Mode**; if the option is greyed out, SAMIAM is already in this mode. To add an edge, go to **Edit**, **Add Edge**, click on whichever node you want to be the parent, and then click on the node you want to be the child. To change the node's properties, double click on it. **You should only change the CPDs for**

the nodes; all other properties should be left the same, as changing any of the other properties will cause problems with the automatic grading system. You can edit the CPD for the selected node by clicking on the **Probabilities** tab, double clicking on any of the values, and simply inputting your own value. Remember that you should edit the CPD for the child node whenever you add an edge.

In the latter part of the assignment, you may wish to use **Query Mode** to monitor the marginals of nodes in this network. Go to the **Mode** menu and select **Query Mode**. On the left you should have a list of nodes. Clicking on one will reveal the values that node can take on. Clicking one of these values will assign it to have that observed value. Clicking again will free the node to be unobserved. To view a node's marginal, right click and select **Monitor**. You can also display all marginals by going to the **Query** menu, then **Show monitors**, and selecting **Show All**.

2.2 Constructing the network

Your friend at the bank, hearing of your newfound expertise in probabilistic graphical models, asks you to help him develop a predictor for whether a person will make timely payments on his/her debt obligations, like credit card bills and loan payments. In short, your friend wants you to develop a predictor for credit-worthiness. He tells you that the bank is able to observe a customer's **income**, the amount of **assets** the person has, the person's **ratio of debts to income**, the person's **payment history**, as well as the person's **age**. He also thinks that the **credit-worthiness** of a person is ultimately dependent on how **reliable** a person is, the person's **future income**, as well as the person's **ratio of debts to income**. As such, he has created a skeleton Bayesian network containing the 8 relevant variables he has mentioned, and defined the possible values they can take in **Credit_net.net**. However he has trouble defining the connections between these variables and their CPDs, so he has asked you to help him.

He hopes that you can help him encode into the network the following observations he has made from his experience in evaluating people's credit-worthiness:

1. The better a person's payment history, the more likely the person is to be reliable.
2. The older a person is, the more likely the person is to be reliable.
3. Older people are more likely to have an excellent payment history.
4. People who have a high ratio of debts to income are likely to be in financial hardship and hence less likely to have a good payment history.
5. The higher a person's income, the more likely it is for the person to have many assets.
6. The more assets a person has and the higher the person's income, the more likely the person is to have a promising future income.
7. All other things being equal, reliable people are more likely to be credit-worthy than unreliable people. Likewise, people who have promising future incomes, or who have low ratios of debts to income, are more likely to be credit-worthy than people who do not.

Add the appropriate edges and define the CPDs so that your network captures the behavior that your friend expects. Your network will be evaluated *solely on whether it produces marginals that are consistent with the desired behavior and not on the actual values of the CPDs in the network.* As an example, here is the condition that your network should satisfy for it to be considered

consistent with observation 1: if we let R denote the random variable for the **reliability** variable, and let H denote the random variable for **payment history**, then your network should satisfy

$$\mathrm{P}(R = \text{Reliable}|H = \text{Excellent}) > \mathrm{P}(R = \text{Reliable}|H = \text{Acceptable}) > \mathrm{P}(R = \text{Reliable}|H = \text{Unacceptable}).$$

You may wish to use **Query Mode** to check your network before calling **submit**[1] in MATLAB/Octave to submit your network for grading. Ensure that the inference algorithm used by SAMIAM is "**hugin**" in order to obtain correct marginals; you may change this option using the left-most drop-down textbox on the toolbar. You may submit your network as many times as you wish, but only your most recent submission will be graded. In addition, you will not receive feedback until after the deadline (and grace period) has passed. This part of the assignment is worth **40 points**.

3 Computing probability queries in a Bayesian network

Inspired by what SAMIAM can do, you decide to replicate some of its functionality in answering probability queries. In this part of the assignment, you will write code to compute (conditional) probabilities by first computing the full joint distribution over the variables in a network, and then marginalizing out irrelevant variables.

3.1 Basic factor operations

Recall that we use factors to represent the CPDs in the Bayesian network. As such, the core functionality, which you will now implement, are the factor product, factor marginalization and factor reduction operations.

- **FactorProduct.m [10 points]** - This function should compute the product of two factors.
- **FactorMarginalization.m [10 points]** - This function should sum over the given variables in a given factor and return the resulting factor.
- **ObserveEvidence.m [10 points]** - This function should modify a set of factors given the observed values of some of the variables, so that assignments not consistent with the observed values are set to zero (in effect, reducing them). These factors do **not** need to be re-normalized.

We will be using MATLAB/Octave structures to implement the factor datatype. A tutorial on the factor datatype and the associated functions that we have provided to manipulate factors is provided in **FactorTutorial.m**. In addition, **FactorTutorial.m** also contains sample factors and expected outputs to help you debug your code. We recommend that you test your code using these inputs as well as on other inputs before submitting your code by calling **submit**.

3.2 Computing the joint distribution

You can now complete the function **ComputeJointDistribution.m**, which computes the joint distribution over a Bayesian network using the factor product operation that you implemented

[1] The **submit** function will not work if your computer is unable to establish a connection to the class servers in MATLAB/Octave. In such cases, we have provided an alternate submission mechanism via the course website. We have provided a function, **submitWeb** that will package the output of your code into a file that you can then submit to the course servers using the "Web Submission" button on the Programming Assignments page on the course website. If **submit** works for you, we recommend using it.

and the chain rule for Bayesian networks. As a reminder, the chain rule states that for a Bayesian network G over variables $X_1, \ldots, X_n$ and a distribution $P(X_1, \ldots, X_n)$ that factorizes over G,

$$P(X_1, \ldots, X_n) = \prod_i P(X_i | \text{Parents}_G(X_i)).$$

- **ComputeJointDistribution.m [10 points]** - This function should return a factor representing the joint distribution given a set of factors that define a Bayesian network. You may assume that you will only be given factors defining valid CPDs, so no input validation is required.

We have provided a simple test case for this function in **FactorTutorial.m** that you may wish to check your implementation with. Call the **submit** function to submit your code.

3.3 Computing marginals

Having computed the joint distribution, we can compute marginal probabilities over sets of variables in the network by marginalizing out the irrelevant variables from the joint distribution. However, this procedure is missing a key step when we have observed evidence – if we had evidence, we would need to first reduce the factor representing the joint distribution by the evidence before marginalizing out the irrelevant variables. In addition, in this case, the factor we obtain is now un-normalized (because **ObserveEvidence.m** does not re-normalize factors), so be sure to normalize the result after calling **ObserveEvidence**. Complete **ComputeMarginal.m** with your implementation.

- **ComputeMarginal.m [20 points]** - This function should return the marginals over input variables (the input variables are those that remain in the marginal), given a set of factors that define a Bayesian network, and, optionally, evidence.

We have provided a simple test case for this function in **FactorTutorial.m** that you may wish to use for checking your implementation. Then, submit your code using the provided **submit** function.

3.4 Performing queries on the credit-worthiness network

Congratulations! You have now implemented a rudimentary inference engine for Bayesian networks. You can use your implementation to experiment with the network for credit-worthiness that you constructed in the earlier part of the assignment. We have provided a function **ConvertNetwork.m** that reads in files in the .net format of SAMIAM into a vector (struct array) of factors. You may wish to check the correctness of your implementation by computing various marginals in the network and seeing if the results match those in SAMIAM; make sure that the inference algorithm used by SAMIAM is "**hugin**" so that it returns exact marginals.