CSC 242 Uncertain Inference Writeup

PRIKSHET SHARMA

CSC Concepts: • **Artificial Intelligence, Uncertainty, Probability**

KEYWORDS

Bayesian Network, Inference, Sampling

0 COMPILATION INSTRUCTIONS

Go to the directory CSC242-project-3-ferguson/src and run the following command:

javac bn/inference/Exact.java

javac bn/inference/Rejection.java

javac bn/inference/LikelihoodWeighting.java

To run the algorithms use:

Java bn.inference.Exact [any\_random\_number] bn/examples/[dataset] [Random Variable] [Evidence Variables]

Java bn.inference.Rejection [number\_of\_samples] bn/examples/[dataset] [Random Variable] [Evidence Variables]

Java bn.inference.LikelihoodWeighting [number\_of\_samples] bn/examples/[dataset] [Random Variable] [Evidence Variables]

1 INTRODUCTION

“A Bayesian network is a directed acyclic graph whose vertices are the random variables {X} U E U Y, where X is the query variable, E are the evidence variables, e are the observed values for the evidence variables and Y are the hidden variables.

The inference problem for Bayesian Networks is to calculate P (X | e), that is, the conditional distribution of the query variable given the evidence (observed values of the evidence variables). In other words, compute the probability of each possible value of the query variable, given the evidence.

In general, we have that:

And for a Bayesian network you can factor that full joint distribution into a product of the conditional probabilities stored at the nodes of the network:

Therefore:

Or in words: “a query can be answered from a Bayesian Network by computing sums of products of conditional probabilities from the network” (AIMA, page 523). These equations are the basis for the various inference algorithms for Bayesian networks. ”[[1]](#footnote-2)

Four Bayesian inference algorithms have been implemented in my project: inference by enumeration, rejection sampling, likelihood weighting and Gibbs sampling.

2 EXPERIMENTAL AND COMPUTATIONAL DETAILS

2.1 Exact Inference

Exact inference is an NP-hard problem, with a time complexity of O(2n).[[2]](#footnote-3) For exact inference, two methods are implemented, enumerationAsk and enumerateAll. The enumerateAsk algorithm evaluates the trees in a Bayesian Network using depth-first recursion.[[3]](#footnote-4)

2.2 Rejection Sampling

“Rejection sampling is a general method for producing samples from a hard-to-sample distribution given an easy-to-sample distribution… First it generates samples from the prior distribution specified by the network. Then, it rejects all those that do not match the evidence. Finally, the estimate P(X=x|e) is obtained by counting how often X = x occurs in the remaining samples.”[[4]](#footnote-5) Random Samples were generated using the following algorithm:

**for**(RandomVariable Xi : bn.getVariableListTopologicallySorted()) {  
 **double** p = Math.*random*();  
 **double** cumulativeProb = 0.0;  
 **for**(Object o : Xi.getDomain()) {  
 x.set(Xi, o);  
 cumulativeProb += bn.getProb(Xi, x);  
 **if**(p <= cumulativeProb) {  
  
 **break**;  
 }  
 }  
}

2.3 Likelihood weighting

“Likelihood weighting avoids the inefficiency of rejection sampling by generating only events that are consistent with the evidence e… [It’s implementation] fixes the values for the evidence variables E and samples only the nonevidence variables… Before tallying the counts in the distribution for the query variable, each event is weighted by the *likelihood* that the event accords to the evidence, as measured by the product of the conditional probabilities for each evidence variable, given its parents. Intuitively, events in which the actual evidence appears unlikely should be given less weight.”[[5]](#footnote-6) Like rejection sampling, Likelihood weighing also uses the following formula to generate random samples:

**double** p = Math.*random*();  
**double** cumulativeProb = 0.0;  
**for**(Object o : Xi.getDomain()) {  
 x.set(Xi, o);  
 cumulativeProb += bn.getProb(Xi, x);  
 **if**(p <= cumulativeProb) {  
 **break**;  
 }  
}

2.4 Gibbs Sampling

“The Gibbs sampling algorithm for Bayesian networks starts with an arbitrary state (with the evidence variables ﬁxed at their observed values) and generates a next state by randomly sampling a value for one of the nonevidence variables Xi. The sampling for Xi is done conditioned on the current values of the variables in the Markov blanket of Xi … The algorithm therefore wanders randomly around the state space—the space of possible complete assignments—ﬂipping one variable at a time, but keeping the evidence variables ﬁxed.”[[6]](#footnote-7)

3 ANALYSIS AND COMPARISON OF IMPLEMENTATIONS

The algorithms were run on different datasets, and their run time was recorded using System.nanoTime(). The query that was used to test the computation time is the following:

java bn.inference.[class\_name] [num\_of-samples] bn/examples/aima-alarm.xml B J true M true.

One of the sample outputs is shown in the following figure:

C:\Users\Prikshet\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture.png

Fig. 1. Sample output for the example query given in the Project description when run with Rejection Sampling Algorithm

3.1 RAW DATA ANALYSIS

|  |  |  |  |
| --- | --- | --- | --- |
| Sample Size (Not applicable to exact inference) | Exact inference | Rejection Sampling | Likelihood Weighting |
| 1,000 | 0.2841 | NaN (0) | 0.3581 |
| 10,000 | 0.2841 | 0.3012 | 0.3133 |
| 100,000 | 0.2841 | 0.2882 | 0.2953 |

Table 1. Shows the probabilities of aima-alarm.xml dataset with different sample sizes. Note: Since we do not sample in Exact inference, all the probabilities are identical. The query used is the one given in project description.

3.2 Comparison of Computational Times

|  |  |  |  |
| --- | --- | --- | --- |
| Sample Size (Not applicable to Exact inference) | Exact inference (ns) | Rejection Sampling (ns) | Likelihood Weighting (ns) |
| 1,000 | 3,852,000 | 112,396,000 | 109,295,000 |
| 10,000 | 3,852,000 | 387,117,000 | 405,116,000 |
| 100,000 | 3,852,000 | 2,019,385,000 | 2,112,969,000 |

Table 3. Shows the computation times of various algorithms in nanoseconds.

3.3 Graphs (Processed Data)

Graph 1. Difference between Exact true value vs. Approximate true value compared across different sample sizes.

Graph 2. Computation time in nanoseconds compared to number of samples taken

3.4 Analysis of Processed Data

Graph 1 tells us that as the sample size increases, the average difference between the exact inference and the approximate inference decreases. This is expected because our approximate inference methods are *consistent*, i.e., as the sample size increases, our calculated probability tends toward the real probability.

Surprisingly, Graph 2 shows that the approximate inference is *worse* than exact inference, even though theoretically it should be faster. However, for full joint distributions with a small number of Random Variables, this is expected. But if we increase the number of Random Variables in our full joint distribution, exact inference becomes intractable.

4 CONCLUSIONS

Approximate inference performs poorly when used with used with full joint distributions with a small number of Random Variables. On the other hand, exact inference becomes intractable as the number of Random Variables increases in our full joint distribution.

REFERENCES

|  |  |
| --- | --- |
| [1] | RUSSELL, STUART NORVIG PETER. *ARTIFICIAL INTELLIGENCE: a Modern Approach*. PEARSON, 2018. |
| [2] | FERGUSON, GEORGE. CSC242 Lecture Slides. |

1. Project 3 description [↑](#footnote-ref-2)
2. Lecture slides [↑](#footnote-ref-3)
3. AIMA Pg.525 [↑](#footnote-ref-4)
4. Ibid. Pg.532 [↑](#footnote-ref-5)
5. Ibid. Pg. 533 [↑](#footnote-ref-6)
6. Ibid Pg. 536 [↑](#footnote-ref-7)