Project 3 Report

Yanhao Ding

November 25, 2018

Contents

The project is developed using Java. The attached README file provides the instructions for running the project. This report outlines the procedure to develop the project and decribes the results of the program.

1 Representation of Bayesian Networks

As described in the project instructions, a Bayesian network is a directed acyclic graph (DAG) of random variables, with a probability distribution stored at each node. So the basic data structure we should use for this project is graph. In addtion to the classical graph representation, we could add more fields of the graph class to represent the possibility table. After read into the .xml files the information shold be constructed into a graph which also contains the possibility table. I will describe this process in the following text.

1.1 Process the XMLBIF representation

Before the construction of Bayesian network, the first job we should do is extract information from the input file. In this project, only basic java io package io java.io.File, java.io.BufferedReader, java.io.FileReader were imported and the file ReadXMLFile.java was developed to read .xml file and get the information necessary for Bayesian network setup. The implementation is straightforward once we read throught the .xml file. I won't describle the detail about this part. The important part is we should store the information and transform then into the graph representation we need after the first read step.

1.2 Building block for Bayesian network

The class **GraphElement** was developed to store the building block for Bayesian network during the read process. The field of **GraphElement** includes String name, List¡String¿ parents, List¡Double¿ probabilities. These three fields are the elements we need when we bulild the Bayesian network. After read the each session of the .xml file, the name of the node, the parents of the node and the correspoding conditional probability table (CPT) was stored in class **GraphElement** for later use. That is the direct information we extract from the input files. Actually this intermidiate process and class is not necessray. We could build the bayesian network directly during reading the input files. However, at the coding process I found keeping more information help me to make things clear. In class **GraphElement**, several method addParents, getName, getParents, getPro, addProbabilities was implemented to interact with the object.

1.3 Object Bayesian network

```
Name : B
Parents :
Children : A
probabilities : 0.001  0.999
*******************************
Name : E
Parents :
Children : A
probabilities : 0.002  0.998
*******************************
Name : A
Parents : B  E
Children : J  M
probabilities : 0.95  0.05  0.94  0.06  0.29  0.71  0.001  0.999
*******************************
Name : J
Parents : A
Children :
probabilities : 0.9  0.1  0.05  0.95
*******************************
Name : M
Parents : A
Children :
probabilities : 0.7  0.3  0.01  0.99
*******************************
```

Figure 1: Representation of Bayesian network

The class **GraphElement** was developed to represent the object, the entire Bayesian network, which includes nodes, connections between nodes and the corresponding conditional probability table(CPT). The field of the class contains the number of vertices(nodes) and edges(connections) of the Bayesian network. The name of all the nodes are stored in a array. For each node, its parents are also stored in a list. And all the list is reprenseted by the array. In the same manner, the children of each node is also represented. as a list. The CPT for certain node is represented by a list. Hashmap was used to map the name of the vertices(nodes) to a interger which represent the number of vertices. If we need to index the vertices from the vertices name array we need this index. In the similar way, we also need a inverse map which maps the indexs of the name array to the name of node. The complete representation of implementation of Bayesian network is shown

as Fig 1. From the above figure, we could see the key points(name, parents, children, CPT) are all included.

1.4 Assumptions

One thing need to mention here is that during the implementation, we made several assumptions or simplifications. The random variables(nodes or vertices) has a name and a domain: the set of its possible values. Technique speaking we should maintain a list to store its domain. However, in out examples, the domain are only two cases, true or false. So in the later implementation, I only consider these two cases. If the variable domian contains more choices, we need maintain the list for each of the random varibale. Secondly, the parents or children could be empty, we showed this case in Fig 1.

2 Exact Inference

2.1 Pre-Processing

After setting up the Bayesian network, we try to develop ExactInference.java first which contains main method of Exact Inference. Inside the main method, we should read from command line input where the file name, query variable and evidences are provided. the input file name is needed for the ReadXMLFile. The query varibale is stored in a lsit X. The evidence variables are also maintained using a list. One important point is that we also need to maintain a hashmap to map from the name of evidence variable to its domain, in out examples, true or false. In the last, the bayesian network was set up and both query variable and evidence variables are passing to exact inference algorithm.

2.2 EnumerationAsk

The "Inference by enumeration" algorithm described in AIMA Section 14.4 was implemented in this part. In the class **EnumerationAsk**, we maintain the private field BayesianNetwork bn and the number of nodes in the network. A public method numerationAsk was implemented to answer the query questions. We should maintain a list to store all the variables in the network. The sequence of this list is important. The sequence of to visit each vertice(node) should be topological order. This is very important, otherwise, we will encouter error in the code. Usually in java nullpointer encountered. Another point I need to method is the process to find correct probability in the conditional probability table(CPT). The CPT is stored in a list in my inplementation. We should find the probability according the sign of parents take and tranform this parents sign into the index and use index to find the probability in the conditional probability table(CPT).

2.3 Results Analysis

The query results are shown as following. The query variable and evidence variable are chosen to shown the results. The comparasion with approximate inference will be shown later. The query and results output from terminal are also documented in separate .txt files. The comparison with exact case is shown at the approximate inference part.

2.3.1 aima-alarm.xml

java ExactInference aima-alarm.xml B J true M true, results : (0.28417184, 0.71582816)
time : 0.00050173
java ExactInference aima-alarm.xml B J true M false, results : (0.00512986, 0.99487014)
time : 0.00042217
java ExactInference aima-alarm.xml B J false M false, results : (0.00009018, 0.99990982)
time : 0.00051026
java ExactInference aima-alarm.xml E J true M true, results : (0.17606684, 0.82393316)
time : 0.00057060
java ExactInference aima-alarm.xml E J true M true B true, results : (0.00202122, 0.99797878)
time : 0.00039169
java ExactInference aima-alarm.xml E J true, results : (0.01628373 0.98371627)

time : 0.00061511

2.3.2 aima-wet-grass.xml

java ExactInference aima-wet-grass.xml R S true, results : (0.30000000, 0.70000000)
time : 0.00043222
java ExactInference aima-wet-grass.xml R S false, results : (0.58571429, 0.41428571)
time : 0.00045020
java ExactInference aima-wet-grass.xml C S false, results : (0.64285714, 0.35714286)
time : 0.00043971
java ExactInference aima-wet-grass.xml C S true, results : (0.16666667 0.83333333)
time : 0.00049770
java ExactInference aima-wet-grass.xml C S true W true, results : (0.17475728, 0.82524272)
time : 0.00040572

2.3.3 dog-problem.xml

java ExactInference dog-problem.xml light-on dog-out true, results : (0.23776811, 0.76223189)
time : 0.00069528
java ExactInference dog-problem.xml light-on dog-out false, results : (0.06353220, 0.93646780)
time : 0.00069356
java ExactInference dog-problem.xml dog-out light-on true bowel-problem false, results : (0.70754717 0.29245283)
time : 0.00058713
java ExactInference dog-problem.xml dog-out light-on true bowel-problem true, results : (0.98358491 0.01641509)
time : 0.00049897

2.3.4 performance of the algorithm

The time complexity of this problem is $O(2^n)$ in worst case, n is the size of the problem. I implement the time clock in the code. However since out test problem is very small, the differences is not that big. But still the time complexity grows as $O(2^n)$ in worst case. For all the above problems, the computation time is approximately less than 0.0001 second. The time consume is quite samll since the problem is very small.

3 Approximate Inference

3.1 Pre-Processing

The Bayesian network implementation can be re-used in this part. Class **ApproxInferencer** is developed in this part. The main method is also implemented in this class. The field of this class contains number of nodes, interactions, Bayesian network bn, all the variables of network vars and query variable evidence variables. Three method was implemented is this class RejectionSampling, LikelihoodWeighting and GibbsAsk. The only difference of input from the previous exact case is that we need read interactions from the command line input.

3.2 Rejection Sampling

The implementation of Rejection Sampling uses the method shown in Figure 14.4 in AIMA book. The sampling algorithm that generates events from Bayesian network is from Figure 14.3. Each variable is sampled according to the conditional distribution given the values already sampled for the variable's parents. The input parameter number of samples are **accepted samples**, do not include the sample we rejected. The sample size we discard might be mush larger than we accepted.

3.3 Likelihood Weighting

The Likelihood Weighting algorithm uses the method shown in Figure 14.4 in AIMA book. In WEIGHTED-SAMPLE, each nonevidence variable is sampled according to the conditional distribution given the values already sampled for the variable's parents, while a weight is accumulated based on the likelihood for each evidence variable.

3.4 Gibbs sampling

The Gibbs sampling algorithm uses the method shown in Figure 14.4 in AIMA book. My inplementation cycles through the variables, but choosing variables at random also works.

3.5 Results Analysis

3.5.1 Absolute probability error versus sample size

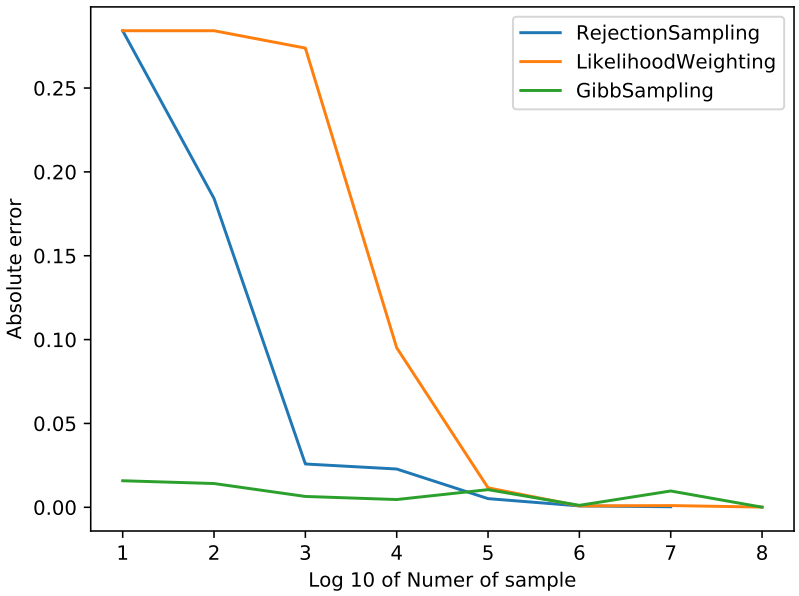

Figure 2: Error vs sample number

The absolute eror of sampled distribution of true case of example aima-alarm.xml B J true M true is shown as Fig 2. The absolute error is the probability of approximate inference substract probability from exact inference which is 0.28417184 in this case. The X axis shows the log_{10} of generated samples, while Y axis shows the absolute error of probability of B equals true case. We could see that the error decrease with the increase of sample size. At the beginning, both rejectiion sample and likelihood Weighting gives pretty large error. The error will reduce to within 1 percent when the sample size is over 10^6. Comparing with both rejection sample and likelihood Weighting, the error of Gibbs sampling is not that big at the beginning.

3.5.2 running time versus sample size

Next, we compare the computation time of each method. Gibbs Fig 3 shows us the running time versus the sample size. The plot is log-log plot, we could easily see that the computation time for rejection sample is much larger(order of magnitude higher) than likelihood weighting and gibbs sampling. The reason is that rejecting sample discard large number of samples. In all three cases, the computation time is increase with the increase of sample size.

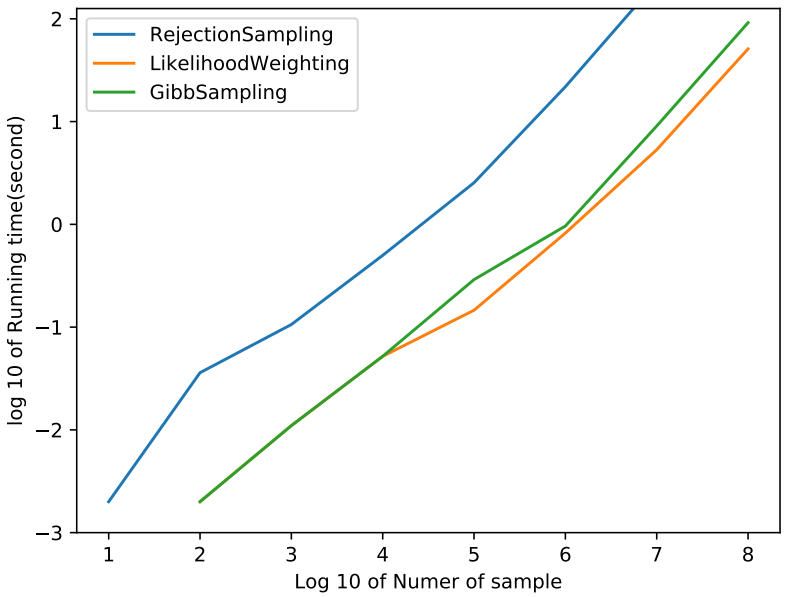

Figure 3: running time vs sample number

3.5.3 Compare with exact inference case

The exact inference time complexity is $O(2^n)$. For large n problem, the exact inference will soon become intractable due to the exponential increase time complexity. The approximate inference could solve this problem since the accuracy of approximate inference is depend on how many sample we generate. In above small problems, the approximate method might not seems better than exact inference, For these problem, the computation time is approximate less than 0.001 second, However, when we deal with larger problem, the approximate method shows its advantage.