Project 3: Uncertain Inference

Xuejian Ye

November 26, 2018

Table of Contents

1. Project Summary

This project aims to construct the program with four different algorithms to do probabilistic inference

a. Language: Python 3

b. Individual work

Notes: *90% of my project is completed by myself, however, I cannot complete the part of reading dog-problem.xml well, so the result of dog-problem.xml inference part is coming from my teammate (Yanhao Ding), which is completed by Java.*

c. How to run project:

Run the code in terminal and it will show the result.

2. Overall Structure and details

2.1 Get information about variables and probability from XML

1) Use a library called xml.dom.minidom.parse as the parser for XML file.

2) Get variables, the values of every variable, their parents and prior or conditional probability table from XML files.

Notes: My program read aima-alarm.xml and aima-wet-grass.xml successfully but will have some problems to read dog-problem.xml due to the data format, which is sensitive to program. Because its CPT (conditional probability table) has no information about their parents' value, only the number without indicating what their parents' values are under the probability.

2.2 Construct Bayes network

There are two classes for constructing bayes network: Bayes_node and Bayse_DAG

1) Class Bayes_node: Stores relevant information about every variable, which including its name, values, parents and prior or conditional probability.

2) Class Bayes_DAG: Constructs Bayes network and has two major attributes: variables and nodes, recording every variables and nodes in the network.

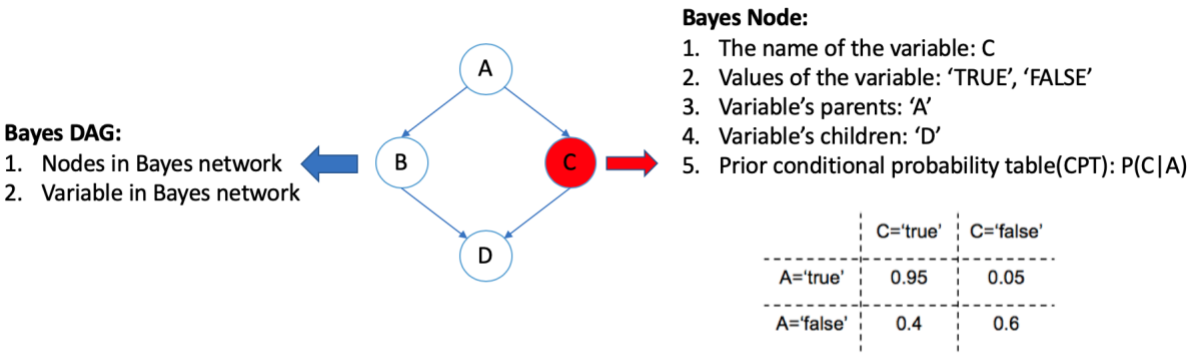


Figure 1. Structures of two classes

In this part, in order to sort nodes in topological order, which means parents should be in front of children. I try to use a brute algorithm: use while loop to put variables in the list if all variable's parents have been in that list, until all variables settle down.

2.3 Exact Inference

2.3.1 Basic Introduction

Two classes are created to create Bayes Net. As for exact inference, all variables are listed and calculated to generate the conditional probability of variable X. Input query and return distribution of variable and then enumerate all variable with specific value (True and False)

2.3.2 Results Analysis

The query results are shown as following. The query variable and evidence variable are chosen to show the results

1) ***Aima-alarm.xml***

aima-alarm.xml B J true M:
results: {'true': 0.2841718353643929, 'false': 0.7158281646356071}
aima-alarm.xml B J true M false
results: {'true': 0.005129858133401301, 'false': 0.9948701418665987}
aima-alarm.xml B J false M false
results: {'true': 9.018439375484347e-05, 'false': 0.9999098156062451}
aima-alarm.xml E J true M true
results: {'true': 0.17606683840507922, 'false': 0.8239331615949208}
aima-alarm.xml E J true M true B true

results: {'true': 0.0020212156643445722, 'false': 0.9979787843356555}

aima-alarm.xml E J true

results: {'true': 0.011394968773811182, 'false': 0.9886050312261888}

2) *Aima-wet-grass.xml*

aima-wet-grass.xml R S true

results: {'true': 0.3, 'false': 0.7}

aima-wet-grass.xml R S false

results: {'true': 0.5857142857142857, 'false': 0.4142857142857143}

aima-wet-grass.xml C S false

results: {'true': 0.6428571428571429, 'false': 0.3571428571428571}

aima-wet-grass.xml C S true

results: {'true': 0.16666666666666669, 'false': 0.8333333333333334}

aima-wet-grass.xml C S true W true

results: {'true': 0.17475728155339806, 'false': 0.825242718446602}

3) *Dog-problme.xml*

java ExactInference dog-problem.xml light-on dog-out true

results: {'true': 0.23776811, 'false': 0.76223189}

java ExactInference dog-problem.xml light-on dog-out false

results: {'true': 0.06353220, 'false': 0.93646780}

java ExactInference dog-problem.xml light-on dog-out true bowel-problem false

results: {'true': 0.98358491, 'false': 0.01641509}

java ExactInference dog-problem.xml light-on dog-out true bowel-problem true

results: {'true': 0.98358491, 'false': 0.01641509}

2.4 Approximate Inference

2.4.1 Rejection sampling

The implementation of Rejection Sampling uses the method shown in Figure 14.1 in AIMA. We need to generate value of variables sequentially according to CPT, then count samples which are consistent with evidence and calculate the percentage of different values (True or False)

2.4.2 Likelihood weighting

The implementation of Rejection Sampling uses the method shown in Figure 14.4 in AIMA. Only generate sample consistent with evidence and weight samples with conditional probability and calculate the percentage of different values (True and False) of variable based on weighted samples.

2.4.3 Gibbs sampling

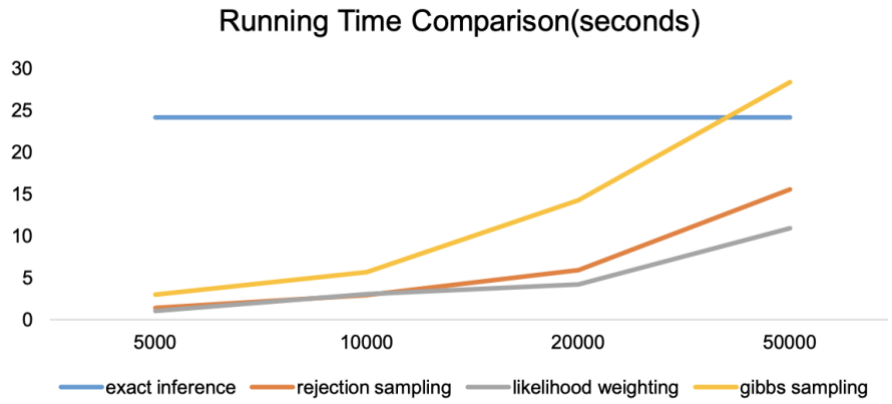
The implementation of Rejection Sampling uses the method shown in Figure 14.4 in AIMA. Generate values of the variables from Marcov blanket and then calculate and return the distribution of variables.

2.4.4 Results Analysis

For approximate inference, rejection sampling, likelihood weighting and gibbs sampling are all implemented. Actually, these algorithms are straightforward and pseudo-codes give lots of help.

1) **Time efficiency benchmark analysis:**

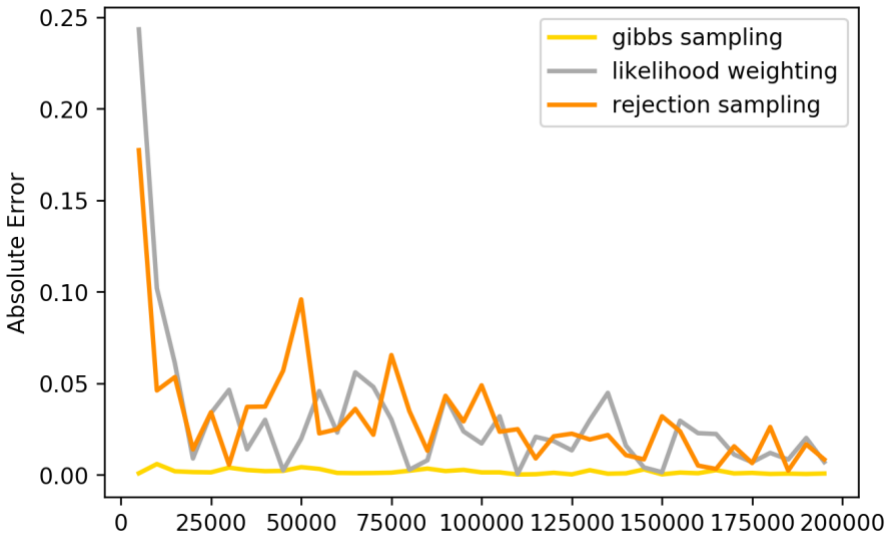
Actually, in order to get the result from exact inference algorithm in acceptable running time, I have to construct a really long evidence with more variables (because $O(2^n)$ time complexity of this algorithm. In order to test time efficiency of 4 algorithms, I used "alarm" problem in the test.



The exact inference has no sampling time; Thus, it is a line. It is shown that under certain sampling times, approximate inference has fewer running time since. Their time complexity is more related to the sampling times(n), while that of exact inference is related to the number of hidden variables(m^n). For three approximate inference algorithms, likelihood weighting has best performance in running time while Gibbs sampling needs most time for it has more computation in sampling process.

2) Accuracy benchmark analysis

Compared with exact inference algorithm, approximate inference sacrifices accuracy to improve the time efficiency. Therefore, accuracy is an important item to evaluate the performance of different approximate inference algorithms. In this part, we will use probability error to test accuracy with the change of sample size.



Example: aima-alarm.xml {number} B J true M true

Because the exact inference will give the right answer (0.284171835), thus we compare this answer to other three algorithms, compare them with the error margin. In this figure, we can see that the error decrease with the increase of sample size. When size is small, both rejection sample and likelihood weighting have bad performance, however, Gibbs sampling get the best results and it is really close to the right answer, when sample size is very large, all of them seem has very good performance.

2.4.5 Conclusion:

In my opinion, my choice depend on the sample size and efficiency, likelihood weighting works best considering its running time and result accuracy. If the sampling size limited, I will choose Gibbs sampling. Moreover, if there are only a few evidence variables, rejection sampling is also a choice since it is easy to write the code.