Project 3: Uncertain Inference

Team members: Yuxin Liu
Sining Qu

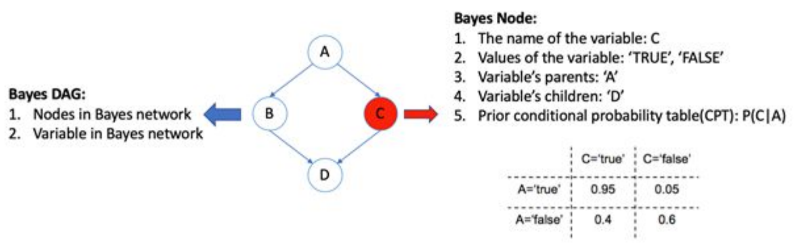
Introduction

In this project, we used Bayesian Networks to implement two kinds of inference algorithms, exact inference (inference by enumeration) and approximate inferences (Rejection sampling and Likelihood weighting), used three examples to test them out and experimented with the number of samples for approximate inferences.

Construct Bayesian network

There are two classes for constructing bayes network: BayesNode and BayesNet

1. Class BayesNode: Stores relevant information about every variable, which includes its name, values, parents and prior or conditional probability.
2. Class BayesNet: Constructs Bayes network and has two major attributes: variables and nodes, recording every variables and nodes in the network.



Bayes DAG:
1. Nodes in Bayes network
2. Variable in Bayes network

Bayes Node:
1. The name of the variable: C
2. Values of the variable: 'TRUE', 'FALSE'
3. Variable's parents: 'A'
4. Variable's children: 'D'
5. Prior conditional probability table(CPT): P(C|A)

	C='true'	C='false'
A='true'	0.95	0.05
A='false'	0.4	0.6

Figure 1. Structures of two classes

We define two classes to construct the network BayesNode and BayesNet. It starts like this:

__init__(self, X, parents, cp):

X: *a variable name*

parents: *a sequence of variable names or a space-separated string.*

cp: *the conditional probability*

Examples:

```
>>> X = BayesNode('B', '', 0.001)

>>> Y = BayesNode('J', 'A', {T: 0.9, F: 0.05})

>>> Z = BayesNode('A', 'B E',

...    {(T, T): 0.95, (T, F): 0.94, (F, T): 0.29, (F, F): 0.001})
```

XML Parser

After some research, we used xml.dom to open the files given. We first located the network by the name ‘NETWORK’ in the file and then nodes by the name ‘DEFINITION’. And we rearranged the nodes so that the ones with no parents come first and parent nodes come before children. Then for each node, we extracted the variable names by ‘FOR’. For conditional probabilities, the condition/parent is extracted by ‘GIVEN’. And then we extracted the conditional probability tables by ‘TABLE’, saved them into a list and only kept the values for when the variable is True. A true/false combo is created if a parent or parents existed to give indications to the conditional probabilities. Then we created a list of tuples containing each node with each tuple consisting of the variable name, the parent names (empty if there’s no parent), and the conditional probabilities with a true/false indication.

For example, for the aima-alarm example, the xml file is parsed into a list as follows,

[('B', '', 0.001), ('E', '', 0.002), ('A', 'B E', {(True, True): 0.95, (True, False): 0.94, (False, True): 0.29, (False, False): 0.001}), ('J', 'A', {True: 0.9, False: 0.05}), ('M', 'A', {True: 0.7, False: 0.01})].

Then we used BaysNet to return the Bayesian Network from the above list, which was discussed in the previous section.

Exact Inference

In this section, we will present the main idea of enumeration algorithm on Bayesian network and some key points in implementing this algorithm.

Enumeration algorithm computes the distribution of a query variable X given evidences e encoded in a Bayesian Network. For each variable of X, we compute products of all conditional probabilities stored in the Bayesian Network. If a variable Y is observed as an evidence, we just multiply the probability P(y|parents(Y)); otherwise, Y is a hidden variable, and we need to sum over all its possible values: P y P(y|parents(y)). The multiplication process is a recursive process until all variables in the network have been touched.

The for loop in ENUMERATION-ASK() loops all possible values of the query variable X, since we hope to compute a probability distribution of X. For each possible value of X, X will be considered as an observed variable and added to evidence E. In the loop, a recursive function ENUMERATE-ALL() will be called. During each recursive process, the conditional probability of one variable in the network is calculated. If the variable is an evidence, e.g., having a value in e, its conditional probability is multiplied directly; otherwise, it is a hidden variable, and a summation will be computed over all its possible values.

Approximate Inference

For approximate inference, we did the rejection sampling and likelihood weighting methods.

Rejection sampling uses a sampling method priorSample to generate events from a Bayesian network first. It randomly samples from P(Xi | parents(Xi)) returning the True/False of each variable according to the conditional probabilities. Rejection sampling checked if sampled events of our query variables from the given Bayesian network is consistent with the evidence we have and returns a normalized probability distribution by counting the consistent events.

Likelihood weighting uses a sampling method weightedSample to generate events from a Bayesian network first. It randomly samples just like priorSample does, but if the sampled variable is part of the evidence, it has a weight which is calculated with P(Xi = xi | parents(Xi)) rather than P(Xi | parents(Xi) for other samples. Likelihood weighting then returns a normalized probability distribution of the query with the weighted counts rather than simply counting.

Experiments

aima-alarm:

command:

```
python exactInference.py examples/aima-alarm.xml B J true M true
python approximateInference.py 10000 examples/aima-alarm.xml B J true M true
```

result:

Enumeration	N	Rejection	Likelihood
{T=0.284, F=0.716}	200	{T=NaN, F=NaN}	{T=0.0, F=1.0}
{T=0.284, F=0.716}	1000	{T=0.667, F=0.333}	{T=0.528, F=0.472}
{T=0.284, F=0.716}	5000	{T=0.230, F=0.769}	{T=0.130, F=0.869}
{T=0.284, F=0.716}	10000	{T=0.277, F=0.722}	{T=0.372, F=0.627}
{T=0.284, F=0.716}	1000000	{T=0.284, F=0.715}	{T=0.298, F=0.701}

When N reaches 1000000, the results from both approximate inference reached within 1% of the exact inference result.

aima-wet-grass:

command:

```
python exactInference.py examples/aima-wet-grass.xml R S true
python approximateInference.py 10000 examples/aima-wet-grass.xml R S true
```

result:

Enumeration	N	Rejection	Likelihood
{T=0.3, F=0.7}	200	{T=0.323, F=0.677}	{T=0.298, F=0.702}
{T=0.3, F=0.7}	1000	{T=0.274, F=0.726}	{T=0.528, F=0.472}

{T=0.3, F=0.7}	5000	{T=0.305, F=0.695}	{T=0.340, F=0.660}
{T=0.3, F=0.7}	10000	{T=0.309, F=0.691}	{T=0.294, F=0.706}

When N reaches 5000, the results from both approximate inference reached within 1% of the exact inference result.

dog-problem:

command:

```
python exactInference.py examples/dog-problem.xml family-out hear-bark true light-on false
python approximateInference.py 10000 examples/dog-problem.xml family-out hear-bark true light-on false
```

result:

Enumeration	N	Rejection	Likelihood
{T=0.858, F=0.142}	200	{T=0.857, F=0.143}	{T=0.832, F=0.168}
{T=0.858, F=0.142}	1000	{T=0.912, F=0.086}	{T=0.848, F=0.152}
{T=0.858, F=0.142}	5000	{T=0.845, F=0.155}	{T=0.868, F=0.132}
{T=0.858, F=0.142}	10000	{T=0.862, F=0.138}	{T=0.858, F=0.142}

When N reaches 10000, the results from both approximate inference reached within 1% of the exact inference, where likelihood weighting reached within 1% when N = 50000 first.

We can see that in general with a larger sample size, such as N = 5000 or 10000, the approximate inference gets results quite similar to the exact inference.

Contribution

Yuxin Liu worked on BayesNet and exact inference part of the project as well as the experiments.
Sining Qu worked on the xml parsing and approximate inference part of the project as well as the readme file.