Gibbs sampling for Bayesian Networks

Davide Sonno
Master's Degree in Artificial Intelligence, University of Bologna
davide.sonno@studio.unibo.it

March 24, 2024

Abstract

This project will discuss the implementation of an algorithm to compute approximate inference for Bayesian Networks: the Gibbs Sampling.

The method for computing approximate inference is an instance of Markov Chain Monte Carlo algorithms, a simulation approach that, using the convergence property of the Markov Chains, computes the desired probability.

We are going to use this method on networks created with the *pgmpy* library and through the class *Gibbs Sampling* we can query some distributions over the model, given some evidence.

Introduction

Domain

In probabilistic reasoning, a Bayesian Network is a structure to represent relationships between variables, in the form of a directed graph. Each node has a *conditional Probability Table (CPT)*, in the case of discrete variables. The rows of those tables are the probabilities of the nodes given the values of its parents, called *conditioning cases*.

We may be interested in computing certain probabilities on this *Bayes net*, in particular when there are some fixed information of the system and we want to know the likelihood of the remaining variables.

There are mathematical ways to compute this, using the so called *Markov Blankets*: the group of variables that directly or indirectly influence each node. If the model involves a lot of variables this might became impossible. [1]

Some techniques have been developed to be able to approximate the desired inference, such as the *Monte Carlo* simulations.

Aim

The purpose of this project is implementing one of those algorithms to compute approximate inference: the GIBBS SAMPLING.

This method will then be used and compared to another method computing the exact inference for a query, to verify its accuracy.

[1] "If it was easy, then we could approximate the desired probability to arbitrary accuracy with a polynomial number of samples. It can be shown that no such polynomial-time approximation scheme can exist.", (Russell and Norvig 2010)

Method

Monte Carlo simulations: To be able to approximate our query, some of the algorithms used are part of the Monte Carlo sampling techniques. The idea is to generate some samples from an easier problem and use them to approximate a more difficult one. A simple example is computing the area of a figure by randomly drawing points and then counting how many ended up inside the figure, with respect to the totality of points.

The most naive approach in producing samples for a complex distribution, is the so called *Rejection Sampling*: generate a bunch of samples and reject those who don't match the evidence. This is proven to converge to the right distribution, but this can be really slow for events with small probabilities.

A more mindful method would be using *Importance Sampling*, for example with *Likelihood Weighting*. This revolves around generating samples from a different distribution, with no particular technical requirements, and weighting them according to the likelihood of evidence.

Markov Chain Monte Carlo: The next class of algorithms emerges by joining the concepts of Monte Carlo simulation and *Markov Chains*. Markov Chains are a random process that generates a sequence of states from an initial state, using the *transition kernel* $k(\boldsymbol{x} \rightarrow \boldsymbol{x'})$.

This transition kernel $\boldsymbol{k}$ expresses the probability of moving from one state to another and only depends on the last state reached. If the chain that we define has some specific properties, such as visiting all the possible states and not getting trapped in loops, it is called an *ergodic* Markov chain.

"A *Markov chain Monte Carlo (MCMC)* method for the simulation of a distribution $\boldsymbol{f}$ is any method producing an ergodic Markov chain $\boldsymbol{X}^{(t)}$ whose stationary distribution is $\boldsymbol{f}$."[2]

Gibbs Sampling: Gibbs Sampling is an instance of those Monte Carlo algorithms developed, in particular, for Bayesian Networks.

[2] Definition 7.1, (Robert and Casella 2004)

The steps of the algorithm used are:

- create the starting state assigning values to all the variables, using evidence values for the evidence
- define the ordering in which all non-evidence variables will be sampled
- for a certain number of iterations:
  - for the variables in the ordering:
    - draw a random sample from $U(0, 1)$
    - using the values of the other variables, draw the next value for the current variable, using the generated number
    - update the current state with new value of the variable
    - update the count for the current state

At the end of the loop we end up with numbers indicating how many time each state has been visited. We can now finally normalize those values and get the approximate inference we were looking for.

Example: Consider a simple network with the nodes $sunny$, $cold$ and $windy$. We are interested in the probability $P(sunny, cold|windy)$. We obtain the following state counts:

COUNTS	sunny	¬sunny
cold	20	45
¬cold	25	10

resulting in this probability distribution:

PROBABILITY	sunny	¬sunny
cold	0.2	0.45
¬cold	0.25	0.1

Convergence of the chain: We can use a Markov chain to sample from a distribution $\pi(x)$ if the *detailed balance* condition holds and if the chain is ergodic.

Detailed balance condition is expressing the equilibrium between *inflow* and *outflow* for every pair of state, so if

$$\pi(\mathrm{x})k(\mathrm{x} \to \mathrm{x}') = \pi(\mathrm{x}')k(\mathrm{x}' \to \mathrm{x}).$$

Let's express a state with x, the single variable with x and all the remaining variables with $\overline{\mathrm{x}}$. A state x can then be written as $(x_i, \overline{x_i})$, so the i-th variable plus all the other.

Now, the transition kernel $k(\mathrm{x} \to \mathrm{x}')$ of the chain created by Gibbs sampling is the following:

1. if exactly one variable changes, say from the value x_i to x'_i, the probability is $k((x_i, \overline{x_i}) \to (x'_i, \overline{x_i})) = P(x'_i|\overline{x_i})$.
2. x and x' differ in one or more variables: this probability is zero since that we only modify at most one variable at the time;
3. the values remain the same, so $k(\mathrm{x} \to \mathrm{x}) = \sum_i k((x_i, \overline{x_i}) \to (x_i, \overline{x_i})) = \sum_i P(x_i|\overline{x_i})$

Suppose $\pi(\mathrm{x}) = P(\mathrm{x}|\mathrm{e})$, for the first case:

$$\begin{aligned}
\pi(\mathrm{x})k(\mathrm{x} \to \mathrm{x}') &= P(\mathrm{x}|\mathrm{e})P(x'_i|\overline{\mathrm{x}}_i, \mathrm{e}) \\
&= P(x_i, \overline{\mathrm{x}}_i|\mathrm{e})P(x'_i|\overline{\mathrm{x}}_i, \mathrm{e}) \\
&= P(x_i|\overline{\mathrm{x}}_i, \mathrm{e})P(\overline{\mathrm{x}}_i|\mathrm{e})P(x'_i|\overline{\mathrm{x}}_i, \mathrm{e}) \\
&= P(x_i|\overline{\mathrm{x}}_i, \mathrm{e})P(x'_i, \overline{\mathrm{x}}_i|\mathrm{e}) \\
&= k(\mathrm{x}' \to \mathrm{x})\pi(\mathrm{x}') \\
&= \pi(\mathrm{x}')k(\mathrm{x}' \to \mathrm{x})
\end{aligned}$$

For case 2. and 3. detailed balance is always satisfied because the transition probabilities are zero (they differ in 2 or more variables).

The ergodicity property is simply satisfied if there are no zeros nor ones in the transition table.

We can notice that the probabilities only depends on the values in the current state and most of the time we use part of the state, rather than all the variables. This is due to the fact that for each variable only it's Markov blanket is needed to infer the probability. The Markov blanket of a node, in a Bayesian network, are all the parents, children and the parents of the node's children. In particular, those blankets can be only computed once and then stored.

Analysis

Experimental setup

To be able to acknowledge if the algorithm is working as intended, we compare its results with methods that compute exact inference, in particular *Variable Elimination* from the pgmpy library. We expect to get very similar values with a big number of iterations. For a number of iterations that goes to infinity we would get the exact distribution, accordingly to the law of large numbers.

Results

Running tests on the same query with a number of iterations increasing from one thousand to one million, the error rate (2-norm, calculated on the difference between the two distributions computed) drops from 0.019 to 0.00086.

The default number of iterations for the algorithm is set to $100\,000$, the user might have to increase this number if there are a lot of variables or if the probabilities are quite low.

Gibbs sampling can sometimes fail whenever the CPT of a variable becomes deterministic, meaning that we may lose the ergodicity of the chain.

Conclusion

Gibbs sampling is a great and easy way to implement approximate inference, even though it can fail and it is generally slow to converge. If we need a more rigorous algorithm, there are some improved versions for the Gibbs algorithm like *block sampling*, drawing multiple variables at the time to restore the chain ergodicity we might lose. Ultimately we can use *Metropolis-Hastings* sampling, of which Gibbs sampling is a particular case, more applicable and flexible.

References

Ankan, A.; Panda, A.; Bali, R.; Shrivastava, R.; and Sen, P. 2023. pgmpy: Probabilistic graphical models using python. `https://pgmpy.org/index.html`.

Heran Lin, Bin Deng, Y. H. Spring 2015. 10-708: Probabilistic graphical models. `https://www.cs.cmu.edu/~epxing/Class/10708-15/notes/10708_scribe_lecture17.pdf`.

Neller, T. W. An introduction to monte carlo techniques in artificial intelligence - part 2. `http://modelai.gettysburg.edu/2017/mc2/index.html`.

Robert, C. P., and Casella, G. 2004. *Monte Carlo Statistical Methods*. Springer Science & Business Media.

Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Pearson.