Probabilistic Artificial Intelligence

Problem Set 3

Oct 26, 2018

1. Variable elimination

In this exercise you will use variable elimination to perform inference on a bayesian network. Consider the network in figure 1 and its corresponding conditional probability tables (CPTs).

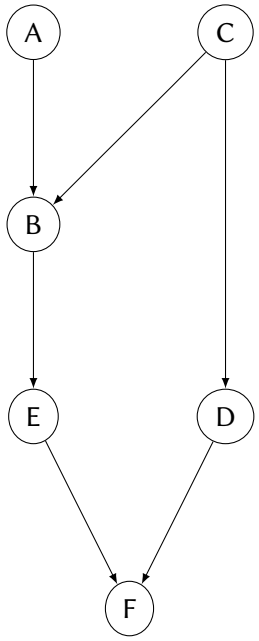

Figure 1: Bayesian network for problem 1.

$$P(A = t) = 0.3 \tag{1}$$
$$P(C = t) = 0.6 \tag{2}$$

Table 1: CPTs for problem 1.

(a) $P(B|A, C)$

A	C	$P(B = t)$
f	f	0.2
f	t	0.8
t	f	0.3
t	t	0.5

(b) $P(D|C)$

C	$P(D = t)$
f	0.9
t	0.75

(c) $P(E|B)$

B	$P(E = t)$
f	0.2
t	0.4

(d) $P(F|D, E)$

D	E	$P(F = t)$
f	f	0.95
f	t	1
t	f	0
t	t	0.25

Assuming a query on A with evidence for B and D, i.e. $P(A|B, D)$, use the variable elimination algorithm to answer the following queries. Make explicit the selected ordering for the variables and compute the probability tables of the intermediate factors.

1. $P(A = t|B = t, D = f)$
2. $P(A = f|B = f, D = f)$
3. $P(A = t|B = t, D = t)$

Consider now the ordering, C, E, F, D, B, A, use again the variable elimination algorithm and write down the intermediate factors, this time without computing their probability tables. Is this ordering better or worse than the one you used before? Why?

2. Belief propagation

In this exercise, you will implement the belief propagation algorithm for performing inference in Bayesian networks. As you have seen in the class lectures, the algorithm is based on converting the Bayesian network to a factor graph and then passing messages between variable and factor nodes of that graph until convergence.

You are provided some skeleton Python code in the `.zip` file accompanying this document. Take the following steps for this exercise.

1. Install the Python dependencies listed in `README.txt`, if your system does not already satisfy them. After that, you should be able to run `demo.py` and produce some plots, albeit wrong ones for now.
2. Implement the missing code in `bprop.py` marked with `TODO`. In particular, you have to fill in parts of the two functions that are responsible for sending messages from variable to factor nodes and vice versa, as well as parts of the function that returns the resulting marginal distribution of a variable node after message passing has terminated.
3. Now, set up the full-fledged earthquake network, whose structure was introduced in Problem Set 2 and is shown again in Figure 2. Here is the story behind this network:

   While Fred is commuting to work, he receives a phone call from his neighbor saying that the burglar alarm in Fred's house is ringing. Upon hearing this, Fred immediately turns around to get back and check his home. A few minutes on his way back, however, he hears on the radio that there was an earthquake near his home earlier that day. Relieved by the news, he turns around again and continues his way to work.

   To build up the conditional probability tables (CPTs) for the network of Figure 2 you may make the following assumptions about the variables involved:

   - All variables in the network are binary.
   - As can be seen from the network structure, burglaries and earthquakes are assumed to be independent. Furthermore, each of them is assumed to occur with probability 0.1%.

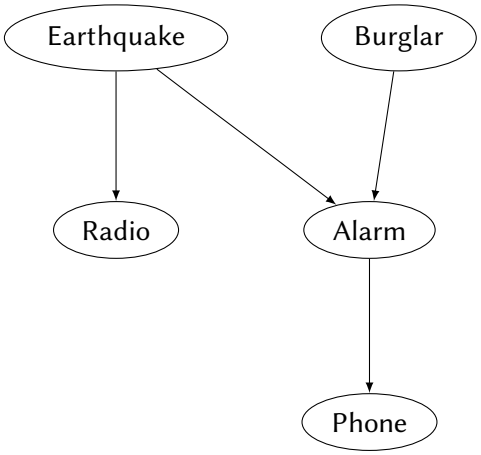

Figure 2: The earthquake network to be implemented.

- The alarm is triggered in the following ways: (1) When a burglar enters the house, the alarm will ring 99% of the time; (2) when an earthquake occurs, there will be a false alarm 1% of the time; (3) the alarm might go off due to other causes (wind, rain, etc.) 0.1% of the time. These three types of causes are assumed to be independent of each other.
- The neighbor is assumed to call only when the alarm is ringing, but only does so 70% of the time when it is actually ringing.
- The radio is assumed to never falsely report an earthquake, but it might fail to report an earthquake that actually happened 50% of the time. (This includes the times that Fred fails to listen to the announcement.)

4. After having set up the network and its CPTs, answer the following questions using your belief propagation implementation:

   (a) Before Fred gets the neighbor's call, what is the probability of a burglary having occurred? What is the probability of an earthquake having occurred?

   (b) How do these probabilities change after Fred receives the neighbor's phonecall?

   (c) How do these probabilities change after Fred listens to the news on the radio?

3. Gibbs sampling

In this exercise, you will implement a Gibbs sampling algorithm for performing approximate inference in Bayesian networks. Although using a factor graph is not necessary for Gibbs sampling, we will use the already available factor graph representation from the previous problem set to conveniently acquire the Markov blanket of each variable. That way, all information required to compute the posterior distribution of a variable v given some values for all other variables, is contained in the CPTs of the neighboring factor nodes $\mathcal{N}(v)$.

More concretely, let $\boldsymbol{x}_{-v}$ be the set of all variables except for v, and $\boldsymbol{s}_{-v}$ be the value of those

variables at the current iteration. Similarly, let $\boldsymbol{x}_{f\setminus v}, \boldsymbol{s}_{f\setminus v}$ be all variables that participate in factor f except for v, and $\boldsymbol{s}_{f\setminus v}$ be the values thereof. Then, to update the value of v you will have to draw from the posterior

$$P(v = d \mid \boldsymbol{x}_{-v} = \boldsymbol{s}_{-v}) = \frac{1}{Z} \prod_{f \in \mathcal{N}(v)} f(v = d, \boldsymbol{x}_{f\setminus v} = \boldsymbol{s}_{f\setminus v}),$$

where Z is a normalization factor. In practice, you will compute the above product (without the $1/Z$ part) for all $d \in \mathrm{dom}(v)$, then normalize to get a proper distribution, and finally draw from that distribution to obtain a new value for v.

You are provided some skeleton Python code in the `.zip` file accompanying this document. Take the following steps for this exercise.

(i) Install the Python dependencies listed in `README.txt`, if your system does not already satisfy them. After that, you should be able to run `demo.py` and produce some plots, albeit wrong ones for now.

(ii) Implement the missing code in `sampling.py` marked with `TODO`. In particular, you have to fill in the part that computes the posterior distribution discussed above, as well as the part that picks a variable and updates the state of the Gibbs sampler.

(iii) If your implementation is correct, you should get (approximately) correct results for the naive Bayes model of the demo file that represents the coin flipping network of exercise 3 in here: `https://las.inf.ethz.ch/courses/pai-f16/sol/hw1-sol.pdf`.

(iv) Now, you can try out your Gibbs sampler on the earthquake network of the previous problem set. Compare your results to those you obtained using belief propagation. There are three parameters you can tune:

- The starting state of the Gibbs sampler. By default, it is created by drawing independent and uniformly random values for each variable.
- The length of the burn-in period, during which the state is updated, but not saved. Therefore, anything sampled during this stage has no effect on the approximate marginals computed afterwards.
- The function used to obtain the approximate marginals that are plotted. By default, this function is a cumulative average, i.e., it computes the approximate marginal distribution of a variable at step i by looking at the average number of occurences of each value of that variable among the samples obtained by the algorithm up to step i. A simple modification would be to only use every k-th sample when computing these averages, since successive samples are heavily correlated.