CS 520: A Simple Bayesian Network Example

16:198:520

Consider a simple Bayesian Network in the form of a chain:

$$X_1 \to X_2 \to X_3 \to \ldots \to X_N. \tag{1}$$

This kind of model frequently describe something changing over time, based on what it was previously. To complete our 'probabilistic knowledge base', we need to specify a base rate for X_1, and conditional probability tables for every link afterwards. It is convenient to define the following terms:

$$B_1 = \begin{bmatrix} \mathbb{P}(X_1) \\ \mathbb{P}(\neg X_1) \end{bmatrix} \tag{2}$$

and for $i = 2, 3, \ldots, N$,

$$C_{i|i-1} = \begin{bmatrix} \mathbb{P}(X_i|X_{i-1}) & \mathbb{P}(X_i|\neg X_{i-1}) \\ \mathbb{P}(\neg X_i|X_{i-1}) & \mathbb{P}(\neg X_i|\neg X_{i-1}) \end{bmatrix}. \tag{3}$$

Specifying values for each of these probabilities in the base vector for X_1 and the transition probabilities from $X_{i-1} \to X_i$ completes the specification of this Bayesian Network. Giving each of the above values is not strictly necessary as they are not independent of each other (i.e., $\mathbb{P}(\neg X_1) = 1 - \mathbb{P}(X_1)$), but this matrix formulation will prove useful in a moment.

1 Querying The Network: The Usual Approach

Suppose we want to know, for a collection of values $\{x_1, x_2, \ldots, x_N\}$ what the probability of these values occurring. This is a full joint assignment of all variables, and we can answer this query fairly immediately, utilizing the conditional relationships between the variables:

$$\begin{aligned} &\mathbb{P}(X_1 : x_1, X_2 : x_2, \ldots, X_N : x_N) \\ &= \mathbb{P}(X_1 : x_1)\,\mathbb{P}(X_2 : x_2|X_1 : x_1)\,\mathbb{P}(X_3 : x_3|X_1 : x_1, X_2 : x_2)\ldots\mathbb{P}(X_N : x_N|X_1 : x_1, \ldots, X_{N-1} : x_{n-1}) \\ &= \mathbb{P}(X_1 : x_1)\,\mathbb{P}(X_2 : x_2|X_1 : x_1)\,\mathbb{P}(X_3 : x_3|X_2 : x_2)\ldots\mathbb{P}(X_N : x_N|X_{N-1} : x_{n-1}) \\ &= \mathbb{P}(X_1 : x_1)\prod_{k=2}^{N}\mathbb{P}(X_k : x_k|X_{k-1} : x_{k-1}). \end{aligned} \tag{4}$$

At this point, every factor can be evaluated by referencing the corresponding factor in the correct probability table.

The computation becomes much more involved when dealing with only *partial* queries. Suppose we are only interested in the probability that $X_N = x_N$. In the usual way, we express this by introducing specifications for all the remaining variables, reducing the partial query to a collection of full queries. But to do this, we have to look at *every other possible value all other variables might have.*

$$\begin{aligned} \mathbb{P}(X_N : x_n) &= \sum_{x_1}\sum_{x_2}\ldots\sum_{x_{N-1}}\mathbb{P}(X_1 : x_1, X_2 : x_2, \ldots, X_N : x_N) \\ &= \sum_{x_1}\sum_{x_2}\ldots\sum_{x_{N-1}}\mathbb{P}(X_1 : x_1)\prod_{k=2}^{N}\mathbb{P}(X_k : x_k|X_{k-1} : x_{k-1}). \end{aligned} \tag{5}$$

Each full query can be computed, using the previous expression - but we are still left with the problem of summing over every term in the summation - representing the 2^{N-1} possible values the other $N-1$ variables might have. This particular instance is something of an extreme case - in general, we may want to query more variables at a given

time and thus have fewer free variables to sum over, but the problem of exponential explosion remains. This can frequently be addressed by systematically iterating over all relevant value assignments (in the manner of traversing a value assignment tree, as discussed in class), but remains a serious computational burden over all.

Conditional queries complicate the issue further. For instance, suppose that given an observation of X_N, you wanted to query the knowledge base about X_1, i.e., compute $\mathbb{P}(X_1 : x_1 | X_N : x_N)$. In this case, the first thing to do would be to expand the conditional query into partial queries, and then treat those similarly, summing over the possible unspecified values for each:

$$\begin{aligned}\mathbb{P}(X_1 : x_1 | X_N : x_N) &= \frac{\mathbb{P}(X_1 : x_1, X_N : x_N)}{\mathbb{P}(X_N : x_N)} \\ &= \frac{\sum_{x_2} \dots \sum_{x_{N-1}} \mathbb{P}(X_1 : x_1, X_2 : x_2, \dots, X_N : x_N)}{\sum_{x_1} \sum_{x_2} \dots \sum_{x_{N-1}} \mathbb{P}(X_1 : x_1, X_2 : x_2, \dots, X_N : x_N)}.\end{aligned} \tag{6}$$

This reduces the original query to something computationally approachable but frequently intractable for incredibly large networks of variables. However, if the relationships between the variables is particularly simple, the simplicity can be exploited to streamline to computation.

2 Querying The Network: A More Efficient Approach

Consider the problem of computing the unconditional distribution of X_2, i.e., $\mathbb{P}(X_2)$ and $\mathbb{P}(\neg X_2)$. The most direct way would be to marginalize and condition based on X_1, in which case you get something like the following:

$$\begin{aligned}\mathbb{P}(X_2) &= \mathbb{P}(X_2 \wedge X_1) + \mathbb{P}(X_2 \wedge \neg X_1) = \mathbb{P}(X_1)\mathbb{P}(X_2|X_1) + \mathbb{P}(\neg X_1)\mathbb{P}(X_2|\neg X_1) \\ \mathbb{P}(\neg X_2) &= \mathbb{P}(\neg X_2 \wedge X_1) + \mathbb{P}(\neg X_2 \wedge \neg X_1) = \mathbb{P}(X_1)\mathbb{P}(\neg X_2|X_1) + \mathbb{P}(\neg X_1)\mathbb{P}(\neg X_2|\neg X_1).\end{aligned} \tag{7}$$

But this same relation can be more efficiently expressed in terms we already defined, namely:

$$\begin{bmatrix}\mathbb{P}(X_2) \\ \mathbb{P}(\neg X_2)\end{bmatrix} = \begin{bmatrix}\mathbb{P}(X_2|X_1) & \mathbb{P}(X_2|\neg X_1) \\ \mathbb{P}(\neg X_2|X_1) & \mathbb{P}(\neg X_2|\neg X_1)\end{bmatrix} \begin{bmatrix}\mathbb{P}(X_1) \\ \mathbb{P}(\neg X_1) \\ ,\end{bmatrix} \tag{8}$$

or, giving the vector a name,

$$B_2 = \begin{bmatrix}\mathbb{P}(X_2) \\ \mathbb{P}(\neg X_2)\end{bmatrix} = M_{2|1} B_1. \tag{9}$$

Expanding on this idea, it is easy to verify that for any $i > 1$, we have that

$$B_i = \begin{bmatrix}\mathbb{P}(X_i) \\ \mathbb{P}(\neg X_i)\end{bmatrix} = M_{i|i-1} B_{i-1}. \tag{10}$$

Re-expressing everything in these linear algebraic terms then allows for very efficient computation of the base rates for any given variable, in terms of products of these transition matrices: For any $i > 1$,

$$B_i = M_{i|i-1} M_{i-1|i-2} \dots M_{2|1} B_1. \tag{11}$$

We can then query the probability of X_i having a given value x_i as the appropriate term in the vector B_i, or $B_i[X_i : x_i]$. This represents a tremendous computational savings over all - a polynomial number of computations to multiply all these matrices, compared to an exponential number of terms to be summed in the previous approach

This approach can be extended further. Similarly, we have the following relation, capturing the relationship between a variable X_{i-1} and a variable X_{i+1}, that

$$M_{i+1|i-1} = \begin{bmatrix} \mathbb{P}(X_{i+1}|X_i) & \mathbb{P}(X_{i+1}|\neg X_i) \\ \mathbb{P}(\neg X_{i+1}|X_i) & \mathbb{P}(\neg X_{i+1}|\neg X_i) \end{bmatrix} = M_{i+1|i}M_{i|i-1}. \tag{12}$$

Extending this similarly, we have that for any $j < i$,

$$M_{i|j} = M_{i|i-1}M_{i-1|i-2}\dots M_{j+1|j}. \tag{13}$$

Using these generalized transition matrices, we can then query between arbitrary variables:

$$\mathbb{P}(X_1 : x_1, X_N : x_N) = \mathbb{P}(X_1 : x_1)\,\mathbb{P}(X_N : x_N|X_1 : x_1) = B_1[X_1 : x_1]M_{N|1}[X_N : x_n, X_1 : x_1]. \tag{14}$$

If we then want to perform the previous conditional query, we have

$$\mathbb{P}(X_1 : x_1|X_N : x_N) = \frac{\mathbb{P}(X_1 : x_1, X_N : x_N)}{\mathbb{P}(X_N : x_N)} = \frac{B_1[X_1 : x_1]M_{N|1}[X_N : x_n, X_1 : x_1]}{B_N[X_N : x_N]}. \tag{15}$$

And we can apply this to the notion of general partial queries as well. For instance, we have for indices $i_1 < i_2 < \dots < i_k$,

$$\begin{aligned} \mathbb{P}(X_{i_1} : x_{i_1}, X_{i_2} : x_{i_2}, \dots, X_{i_k} : x_{i_k}) &= \mathbb{P}(X_{i_1} : x_{i_1})\,\mathbb{P}(X_{i_2} : x_{i_2}|X_{i_1} : x_{i_1})\dots\mathbb{P}\left(X_{i_k} : x_{i_k}|X_{i_{k-1}} : x_{i_{k-1}}\right) \\ &= B_{i_1}[X_{i_1} : x_{i_1}]\prod_{j=2}^{k} M_{i_j|i_{j-1}}[X_{i_j} : x_{i_j}, X_{i_{j-1}} : x_{i_{j-1}}] \end{aligned} \tag{16}$$

Taking the above approach to partial queries, we can perform general conditional queries efficiently as well. At every step, we simply reference the appropriate term of the relevant base rate vector or transition matrix, multiplying everything as we go. And the important observation to make here - all these transition matrices can be efficiently computed from the original data. They might even be pre-computed before any queries are performed (or memorized along the way).

2.1 Why does this work?

To begin with, it is worth asking - why does this approach work? What is computing

$$B_N = M_{N|N-1}M_{N-1|N-2}\dots M_{2|1}B_1 \tag{17}$$

capturing that

$$\mathbb{P}(X_N : x_n) = \sum_{x_1}\sum_{x_2}\dots\sum_{x_{N-1}} \mathbb{P}(X_1 : x_1)\prod_{k=2}^{N}\mathbb{P}(X_k : x_k|X_{k-1} : x_{k-1}) \tag{18}$$

is not?

The key observation is that there a lot of *redundancy* in the above summation. We have to visit or examine the factor $\mathbb{P}(X_N : x_N|X_{N-1} : x_{N-1})$ not only for each of the two values of x_{N-1}, but also *for every possible value of the remaining $N-2$ variables*. The summation approach, with it's top down view of variable assignment, then probability computation, does not exploit the fact that this factor really isn't changing (much) from set of assignments to set of assignments. This is exactly what the matrix multiplication approach does, however, essentially computing probabilities 'from the bottom up' looking at all the possible values of X_{N-1}, then seeing how that impacts all the possible values of X_{N-2}, and iterating this along the chain. This is an incredible efficiency boost over all.

2.2 Generalizing and Extending

It would be nice if this kind of trick were available to simplify querying Bayesian networks generally. Unfortunately, recall that in full generality probabilistic inference is at least as hard as logical inference, hence (subject to some claims about **P** vs **NP**), *in general* querying hard networks is going to be a hard problem. This specific network admits a nice simplification because it is so simple.

However, we can generalize and extend this approach somewhat: Note, for instance, that this approach immediately generalizes to random variables that take on multiple discrete values (not just **true** and **false**) - we simply need to expand the size of our transition matrices and base rate vectors. We can additionally extend this though taking into account more complicated dependencies between the variables.

Suppose, for instance, that X conditionally depended both on Y and Z. In this case, we could express the conditional probability table $(X|Y,Z)$ as not a two-dimensional matrix but a *three-dimensional tensor*, capturing all the ways that X might depend on the specific values of Y and Z. Expressing all conditional probability tables as tensors, we can then multiply them as before (taking some care in how we multiply tensors) to produce new tensors, capturing the distribution and dependency of other variables. As the number of dimensions of these tensors grows (i.e., we allow more complicated dependencies between variables in our network) multiplying these factors becomes computationally more intensive until we approach something on the same order of complexity as evaluating the full summation we started with.