General Inference

A Bayesian Network gives us a reasonable way to specify the joint probability of a network of many random variables. Before we celebrate, realize that we still don't know how to use such a network to answer probability questions. There are many techniques for doing so. I am going to introduce you to one of the great ideas in probability for computer science: we can use sampling to solve inference questions on Bayesian networks. Sampling is frequently used in practice because it is relatively easy to understand and easy to implement.

Rejection Sampling

As a warmup consider what it would take to sample an assignment to each of the random variables in our Bayes net. Such a sample is often called a "joint sample" or a "particle" (as in a particle of sand). To sample a particle, simply sample a value for each random variable one at a time based on the value of the random variable's parents. This means that if X_i is a parent of X_j, you will have to sample a value for X_i before you sample a value for X_j.

Let's work through an example of sampling a "particle" for the Simple Disease Model in the Bayes Net section:

1. Sample from $P(\text{Uni} = 1)$: Bern(0.8). Sampled value for Uni is 1.
2. Sample from $P(\text{Influenza} = 1 | \text{Uni} = 1)$: Bern$(0.2)$. Sampled value for Influenza is 0.
3. Sample from $P(\text{Fever} = 1 | \text{Influenza} = 0)$: Bern$(0.05)$. Sampled value for Fever is 0.
4. Sample from $P(\text{Tired} = 1 | \text{Uni} = 1, \text{Influenza} = 0)$: Bern$(0.8)$. Sampled value for Tired is 0.

Thus the sampled particle is: [Uni = 1, Influenza = 0, Fever = 0, Tired = 0]. If we were to run the process again we would get a new particle (with likelihood determined by the joint probability).

Now our strategy is simple: we are going to generate N samples where N is in the hundreds of thousands (if not millions). Then we can compute probability queries by counting. Let $N(\mathbf{X} = \mathbf{k})$ be notation for the number of particles where random variables $\mathbf{X}$ take on values $\mathbf{k}$. Recall that the bold notation $\mathbf{X}$ means that $\mathbf{X}$ is a vector with one or more elements. By the "frequentist" definition of probability:

$$P(\mathbf{X} = \mathbf{k}) = \frac{N(\mathbf{X} = \mathbf{k})}{N}$$

Counting for the win! But what about conditional probabilities? Well using the definition of conditional probabilities, we can see it's still some pretty straightforward counting:

$$P(\mathbf{X} = \mathbf{a} | \mathbf{Y} = \mathbf{b}) = \frac{P(\mathbf{X} = \mathbf{a}, \mathbf{Y} = \mathbf{b})}{P(\mathbf{Y} = \mathbf{b})} = \frac{\frac{N(\mathbf{X}=\mathbf{a}, \mathbf{Y}=\mathbf{b})}{N}}{\frac{N(\mathbf{Y}=\mathbf{b})}{N}} = \frac{N(\mathbf{X} = \mathbf{a}, \mathbf{Y} = \mathbf{b})}{N(\mathbf{Y} = \mathbf{b})}$$

Let's take a moment to recognize that this is straight-up fantastic. General inference based on analytic probability (math without samples) is hard even given a Bayesian network (if you don't believe me, try to calculate the probability of flu conditioning on one demographic and one symptom in the Full Disease Model). However if we generate enough samples we can calculate any conditional probability question by reducing our samples to the ones that are consistent with the condition ($\vec{Y} = \vec{b}$) and then counting how many of those are also consistent with the query ($\vec{X} = \vec{a}$). Here is the algorithm in pseudocode:

```
N = 10000
# "query" is the assignment to variables we want probabilities for
# condition" is the assignments to variables we will condition on
def get_any_probability(query, condition):
  particles = generate_many_joint_samples(N)
  cond_particles = reject_non_consistent_samples(particles, condition)
  K = count_consistent_samples(cond_particles, query)
  return K / len(cond_particles)
```

This algorithm is sometimes called "Rejection Sampling" because it works by generating many particles from the joint distribution and rejecting the ones that are not consistent with the set of assignments we are conditioning on. Of course this algorithm is an approximation, though with enough samples it often works out to be a very good approximation. However, in cases where the event we're conditioning on is rare enough that it doesn't occur after millions of samples are generated, our algorithm will not work. The last line of our code will result in a divide by 0 error. See the next section for solutions!

General Inference when Conditioning on Rare Events

Joint Sampling is a powerful technique that takes advantage of computational power. But it doesn't always work. In fact it doesn't work any time that the probability of the event we are conditioning is rare enough that we are unlikely to ever produce samples that exactly match the event. The simplest example is with continuous random variables. Consider the Simple Disease Model. Let's change Fever from being a binary variable to being a continuous variable. To do so the only thing we need to do is re-specify the likelihood of fever given assignments to its parents (influenza). Let's say that the likelihoods come from the normal PDF:

$$\text{if Influenza} = 0\text{, then Fever} \sim N(\mu = 98.3, \sigma = 0.7)$$
$$\therefore f(\text{Fever} = x) = \frac{1}{\sqrt{2\pi \cdot 0.7}} e^{-\frac{(x-98.3)^2}{2 \cdot 0.7}}$$

$$\text{if Influenza} = 1\text{, then Fever} \sim N(\mu = 100.0, \sigma = 1.8)$$
$$\therefore f(\text{Fever} = x) = \frac{1}{\sqrt{2\pi \cdot 1.8}} e^{-\frac{(x-100.0)^2}{2 \cdot 1.8}}$$

Drawing samples (aka particles) is still straightforward. We apply the same process until we get to the step where we sample a value for the Fever random variable (in the example from the previous section that was step 3). If we had sampled a 0 for influenza we draw a value for fever from the normal for healthy adults (which has $\mu = 98.3$). If we had sampled a 1 for influenza we draw a value for fever from the normal for adults with the flu (which has $\mu = 100.0$). The problem comes in the "rejection" stage of joint sampling.

When we sample values for fever we get numbers with infinite precision (eg 100.819238 etc). If we condition on someone having a fever equal to 101 we would reject every single particle. Why? No particle will have exactly a fever of 101.

There are several ways to deal with this problem. One especially easy solution is to be less strict when rejecting particles. We could round all fevers to whole numbers.

There is an algorithm called "Likelihood Weighting" which sometimes helps, but which we don't cover in CS109. Instead, in class we talked about a new algorithm called Markov Chain Monte Carlo (MCMC) that allowed us to sample from the "posterior" probability: the distribution of random variables after (post) us fixing variables in the conditioned event. The version of MCMC we talked about is called Gibbs Sampling. While I don't require that students in CS109 know how to implement Gibbs Sampling, I wanted everyone to know that it exists and that it isn't beyond your capabilities. If you need to use it, you can learn it given the knowledge you have now.

MCMC does require more math than Joint Sampling. For every random variable you will need to specify how to calculate the likelihood of assignments given the variable's: parents, children and parents of its children (a set of variables cozily called a "blanket"). Want to learn more? Take CS221 or CS228!

Thoughts

While there are slightly-more-powerful "general inference algorithms" that you will get to learn in the future, it is worth recognizing that at this point we have reached an important milestone in CS109. You can take very complicated probability models (encoded as Bayesian networks) and can answer general inference queries on them. To get there we worked through the concrete example of predicting disease. While the WebMD website is great for home users, similar probability models are being used in thousands of hospitals around the world. As you are reading this general inference is being used to

improve health care (and sometimes even save lives) for real human beings. That's some probability for computer scientists that is worth learning. What if we don't have an expert? Could we learn those probabilities from data? Jump to part 5 to answer that question.