EECS 16ML Fa/Sp 20xx **Machine Learning in Practice** **Gwyn, Durvasula, Hurst, and Arora** Note **x**

1 Bayesian networks

Probabilistic models and the Bayesian interpretation of statistics have played and continue to play a key role in machine learning and its applications. Probabilistically interpreting a statistical model – or more directly, building a model based on ideas in probability – tends to make elements of machine learning more intuitive and more interpretable. In this note, we turn our attention to a single type of probabilistic model: the **Bayesian network** (alternatively called a **Bayesian belief network** or a **Bayes' net**). Before we can discuss Bayes' nets, however, we review the underlying mathematics.

1.1 Bayesian probability

Underlying any probabilistic model is the Bayesian interpretation of statistics, in which we view the outcome of an event as being drawn from a probability distribution. More verbosely, the "fundamental figure" in which we are interested is our confidence that an event will have a certain outcome. This comes in contrast to the frequentist point of view, in which the fundamental figure is the frequency with which a certain outcome occurs over many observations of a certain event. The distinction may be unclear, and indeed the two philosophies may be viewed as two sides of the same coin (statistical inference). Since the nuances of statistical methodology are neither our prime focus in this note nor a crucial prerequisite, we conclude our discussion of Bayesianism and frequentism here.

Note: Sections 1.1.2, 1.1.3, and 1.1.4 are *not* necessary to understand discrete Bayesian networks (Bayesian networks containing discrete random variables) and can be freely skipped for the sake of this lesson. However, the applications of Bayesian networks to continuous random variables rest heavily on the concepts outlined in those sections, so motivated students are encouraged to read them.

1.1.1 Probability distributions

We include this interlude on introductory probability for completeness and to broaden the potential audience of these notes. For students who have taken or are concurrently taken CS 70 (or who are familiar with the language of probability), this may largely be review and hence can be freely skipped.

The central object of probability theory is the **experiment**. An experiment is a process of which we can conduct arbitrarily many **trials**, and each trial returns one out of a set of possible **outcomes** (the terms "experiment" and "trial" are often used interchangeably, but we use them separately to highlight the connection with real-world experimentation). Here are some examples of experiments:

- *Coin toss:* we can flip a coin as many times as we want; each trial (one toss of the coin) gives us one of the outcomes in the set
$$\Omega = \{\text{Heads}, \text{Tails}\}$$
- *Dice roll:* we can roll a standard die as many times as we want; each trial (a roll) gives us one of the outcomes in the set
$$\Omega = \{1, 2, 3, 4, 5, 6\}$$
- *Javelin throw:* we can observe an Olympian (preferably one who throws javelins) throw a javelin and record the displacement of the javelin, in meters. In this case, our set of outcomes could be
$$\Omega = \{x \in \mathbb{R} \mid x \geq 0\}$$

Bearing in mind that even an Olympian is unlikely to, for example, send a javelin into orbit, we might adjust our set of outcomes to be

$$\Omega = \{x \in \mathbb{R} \mid 0 \leq x \leq 200\}$$

You may have inferred at this point that we tend to denote the set of outcomes as Ω.

In any of the above examples, we can speak of the probability of something occurring. In the dice roll case, we'd say that the die has a probability of $\frac{1}{6}$ of turning up 5, a $\frac{1}{6}$ probability of turning up 2, and so on. In the coin toss case, the coin has a $\frac{1}{2}$ probability of turning up heads and a $\frac{1}{2}$ probability of turning up tails. In the javelin throw case, however, we get a bit stuck. What is the probability that the javelin travels *exactly* 67.2 meters – not 67.1, not 67.22? Indeed, this probability would be very small, as we find it very unlikely that this would occur, and indeed, we rate the probability of this event as 0. On the other hand, if asked how likely it is that the javelin travels *at least* 67.2 meters, then we might be more apt to give a response. For a talented male javelin thrower, this probability might be something like 0.9. For a female, this might be more like 0.2.

When we speak of probabilities, we can't always just talk about *outcomes*, as evidenced by the javelin example above. Rather, we have to speak of **events**, which are *sets* of outcomes to which we can assign a probability. In this dice roll example, one event might be that the die turns up an odd number, which corresponds to the set of events $\{1, 3, 5\}$. The probability of this occurring is $\frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}$. Notice the idea at play here: we have three *disjoint* events (events that cannot both occur in a singe trial)

$$E_1 = \{1\} \qquad E_2 = \{3\} \qquad E_3 = \{5\}$$

and we have assigned a probability to their union by adding up their individual probabilities:

$$\mathbb{P}[E_1 \sqcup E_2 \sqcup E_3] = \mathbb{P}[E_1] + \mathbb{P}[E_2] + \mathbb{P}[E_3]$$

this relationship holds with any number of disjoint sets, or even a countable number of them.

In the javelin example, we saw that the event $\{67.2\}$ occurs with probability 0, while the event $\{x \mid x \geq 67.2\}$ happens with probability 0.9 or 0.2, depending on the sex of the javelin thrower. A robust discussion of a probabilistic experiment will contain, then, a set of outcomes Ω, a set of events $\mathcal{F}$, and a **probability distribution function** $\mathbb{P} : \mathcal{F} \to [0, 1]$.

Problem. Consider an experiment which consists of flipping *two* fair coins.

(a) List all outcomes of this experiment.

(b) List all events and their probabilities for this experiment.

1.1.2 Probability density functions

We saw in the javelin example of the previous section that it sometimes doesn't do us much good to assign probabilities to individual outcomes. Indeed, when we are dealing with **continuous random variables**, which we'll define as random variables with an uncountable number of possible outcomes (i.e. outcomes that could actually occur, even if only with zero probability). In this note, we deal with random variables that take on real values. The pertinent task when it comes to real-valued random variables is generally to assign a probability to each open interval (a, b). Notice, though, that

$$\mathbb{P}[(-\infty, b)] = \mathbb{P}[(-\infty, a)] + \mathbb{P}[(a, b)]$$

or, rearranging,

$$\mathbb{P}[(a, b)] = \mathbb{P}[(-\infty, b)] - \mathbb{P}[(-\infty, a)]$$

so we really need only specify the function $F(x) = \mathbb{P}[(-\infty, x)]$ in order to understand a real-valued random variable. This function is called the **cumulative density function** of the random variable. This function is continuous in most applications (and it is *always* nondecreasing); we restrict our attention to when it is differentiable. The fundamental theorem of calculus gives us that

$$\mathbb{P}[(a,b)] = F(b) - F(a) = \int_a^b F'(x)\,dx$$

The function $f := F'$ is called the **probability density function** of the random variable. Indeed, the intervals where this function is larger have a greater probability as events, and we can view this as there being a greater "density of probability" within such an interval. Notice some key properties of probability density functions:

- Since F is nondecreasing, we have $f(x) \geq 0$ for all $x \in \mathbb{R}$.
- Since $\mathbb{P}(\mathbb{R}) = 1$, we have

$$1 = \mathbb{P}(\mathbb{R}) = \mathbb{P}((-\infty,\infty)) = \lim_{x\to\infty} F(x) = \lim_{x\to\infty} \int_{-\infty}^{x} f(t)\,dt = \int_{-\infty}^{\infty} f(t)\,dt$$

1.1.3 Common probability density functions

Here we list some important density functions of continuous random variables with which any reader learning about probability should be familiar. In each case, it is left as an exercise to the reader to verify that two properties listed at the end of the previous section:

- *Uniform distribution:* When we have a bounded interval $[a,b]$ (or a union of bounded intervals, but we'll stick to a single interval for now), we can define the probability density function

$$f(x) = \begin{cases} \frac{1}{b-a} & x \in [a,b] \\ 0 & \text{else} \end{cases}$$

  which is called uniform because the density is constant all across the interval with which we are dealing.

- *Exponential distribution:* This density function is often used to model how much time will pass before a certain occurrence takes place. The density function is given by

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{else} \end{cases}$$

  where λ is an arbitrary real parameter.

- *Gaussian distribution:* Arguably the most "important" distribution due to its ubiquity in statistics and machine learning alike, the density function is given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

  where μ and σ are real parameters.

1.1.4 Expectation and Variance

When we have a random variable that takes numerical values, it is often convenient to be able to describe the "average" value of that random variable and how far from this average the random variable tends to vary. With this in mind, we define the **expectation** and the **variance** of a random variable X whose density function is f:

$$\mathbb{E}[X] := \int_{-\infty}^{\infty} x f(x)\,dx \qquad \mathrm{Var}[X] := \int_{-\infty}^{\infty} (x - \mathbb{E}[X])^2 f(x)\,dx$$

We can see that the expectation is like a weighted average of the real numbers with weights based on the density function. We can also observe that if we were to define another random variable Y by $Y =$

Distribution	Parameters	Expectation	Variance
Uniform	a, b	$(a+b)/2$	$(b-a)^2/12$
Exponential	λ	$1/\lambda$	$1/\lambda^2$
Gaussian	μ, σ	μ	σ^2
Dice roll	None	$7/2$	$35/12$

Figure 1: The expectations and variances of several random variables.

Event	Probability without information	Probability given prime
$\{1\}$	$1/6$	0
$\{2\}$	$1/6$	$1/3$
$\{3\}$	$1/6$	$1/3$
$\{4\}$	$1/6$	0
$\{5\}$	$1/6$	$1/3$
$\{6\}$	$1/6$	0
$\{2,4,6\}$	$1/2$	$1/3$
$\{1,3,5\}$	$1/2$	$1/3$
$\{5,6\}$	$1/3$	$1/3$
$\{2,3,5\}$	$1/2$	1

Figure 2: The probability distribution of the outcome of a dice roll f_X, and the probability distribution of the outcome of a dice roll given the event P that the outcome is prime: $f_{X|P}$

$(X - \mathbb{E}[X])^2$, then $\mathrm{Var}[X] = \mathbb{E}[Y]$, so the variance encapsulates the average *squared* distance from a single observation of X to the expectation.

The reader is encouraged to verify that the expectations and variances presented in Figure 1 are indeed correct.

1.1.5 Conditional Probability and Bayes' Law

Often when we are trying to discern something about the behavior of some random quantity, we come into some information that helps us hone our understanding. This is, after all, the entire purpose of data collection when it comes to training probabilistic models. For example, if you are trying to predict the outcome of a dice roll, you will assign a probability of $\frac{1}{6}$ to each of the six possible outcomes. But if I offer you the additional information that the outcome of the dice roll will be prime, then suddenly your assumed probability distribution changes (see Figure 2). The way we represent this mathematically is with the notation $f_{X|E}$. This denotes a "new" probability function which is the density of the outcome of X given that the event E will occur (the outcome will be contained in E). In the above dice roll case, the aforementioned event is the event P that the dice roll turns up a prime.

The concept of conditional probability includes a very important equation that allows us to determine the probability of two events occurring. That is, if E_1 and E_2 are events, we are interested in the probability of $E_1 \cap E_2$. We have that

$$\mathbb{P}[E_1 \cap E_2] = \mathbb{P}[E_1] \cdot \mathbb{P}[E_2 \mid E_1]$$

This should make intuitive sense, as the "fraction of all outcomes" that $E_1 \cap E_2$ makes up can be viewed as some fraction of the outcomes made up by E_1, which in turn makes up some fraction of all outcomes.

Notice that we could just as well have written

$$\mathbb{P}[E_1 \cap E_2] = \mathbb{P}[E_2] \cdot \mathbb{P}[E_1 \mid E_2]$$

Equating these two formulae gives us

$$\mathbb{P}[E_1] \cdot \mathbb{P}[E_2 \mid E_1] = \mathbb{P}[E_2] \cdot \mathbb{P}[E_1 \mid E_2]$$

and dividing by $\mathbb{P}[E_1]$ gives

$$\mathbb{P}[E_2 \mid E_1] = \frac{\mathbb{P}[E_2]\mathbb{P}[E_1 \mid E_2]}{\mathbb{P}[E_1]}$$

and this is known as **Bayes' Law** (or **Bayes' theorem**).

1.1.6 An example

Let's revisit the javelin example once more. We noted that the probability of an Olympian throwing a javelin at least 67.2 meters would be higher for a male javelin thrower than for a female javelin thrower. Say we don't know the sex of the thrower we'll be observing. Then we can view outcomes of this random variable as a tuple (S, D), where S is the sex of the thrower and D is the distance travelled by the javelin. When we specify the distribution of this ordered pair, we are specifying the **joint distribution** of the sex and the distance. Since our probabilities will differ based on whether the thrower is a man or a woman, we will specify to distributions. In particular, we could choose

$$[D \mid S = \text{man}] \sim \text{Normal}(\mu = 76, \sigma = 6) \qquad [D \mid S = \text{woman}] \sim \text{Normal}(\mu = 60, \sigma = 5)$$

and we additionally specify that there is a 75% probability that our thrower is a man. Now suppose we are given the distance that the javelin travelled, and we want to reason a guess at the sex of the thrower. In particular, we are told that the javelin travelled less than 55 meters. Then we can use Bayes' law to calculate

$$\begin{aligned}
\mathbb{P}[S = \text{woman} \mid D < 55] &= \frac{\mathbb{P}[D < 55 \mid S = \text{woman}]\mathbb{P}[S = \text{woman}]}{\mathbb{P}[D < 55]} \\
&= \frac{\mathbb{P}[D < 55 \mid S = \text{woman}]\mathbb{P}[S = \text{woman}]}{\mathbb{P}[D < 55 \wedge S = \text{woman}] + \mathbb{P}[D < 55 \wedge S = \text{man}]} \\
&= \frac{\mathbb{P}[D < 55 \mid S = \text{woman}]\mathbb{P}[S = \text{woman}]}{\mathbb{P}[D < 55 \wedge S = \text{woman}] + \mathbb{P}[D < 55 \wedge S = \text{man}]} \\
&= \frac{\mathbb{P}[D < 55 \mid S = \text{woman}]\mathbb{P}[S = \text{woman}]}{\mathbb{P}[D < 55 \mid S = \text{woman}]\mathbb{P}[S = \text{woman}] + \mathbb{P}[D < 55 \mid S = \text{man}]\mathbb{P}[S = \text{man}]}
\end{aligned}$$

All of the probabilities in the final quantity above are given by the two Gaussians we specified above and the fact that we generally (given no conditions) expect a male thrower with probability 0.75. This preexisting assumption that the thrower has a 75% chance of being male is called a **prior**, and it can be seen as one of the major differences between Bayesian and frequentist probability.

1.2 Directed acyclic graphs

Bayesian networks are among a class of models called **graphical models**. Here we outline a bit of language relating to graphs for those that have not seen it (as with the previous section, those already familiar with this language are free to skip the section).

A **graph** is a collection of **nodes** or **vertices** along with a set of **edges**, which connect the nodes. A graph may be **directed** or **undirected**. In the directed case, we represent edges as ordered pairs (v_1, v_2) of vertices. In the undirected case, we represent edges as unordered pairs $\{v_1, v_2\}$. Undirected graphs are of great importance in many applications, but in this note we concern ourselves with directed graphs only. Graphs are often used to depict relationships between objects that are represented by their vertices. For example, we might have a graph whose vertices are people and whose edges connect people with a common parent, grandparent, or great-grandparent. This is most aptly depicted as an undirected graph.

We could also have a graph whose vertices are classes and whose edges indicate that one class is a prerequisite of another. A directed graph is more suitable or this situation. Note that, if we had such a graph of classes and their dependencies, we wouldn't expect to see something like "A is a prerequisite of A" or "A is a prerequisite of B, which is a prerequisite of C, which is in turn a prerequisite of A." Graphically, these situations are shown in Figure 4. In this scenario and, as it will turn out, with Bayesian networks, our edges

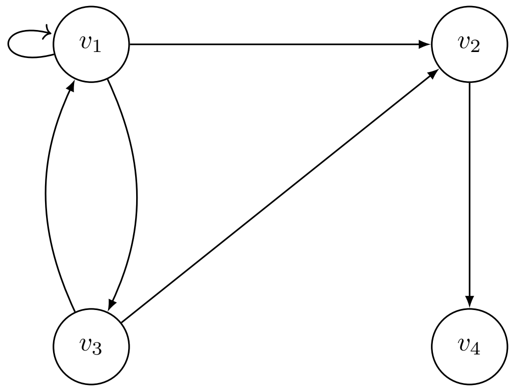

Figure 3: A simple directed graph with vertex set $V = \{v_1, v_2, v_3, v_4\}$ and edge set $\{(v_1, v_1), (v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_1), (v_3, v_2)\}$.

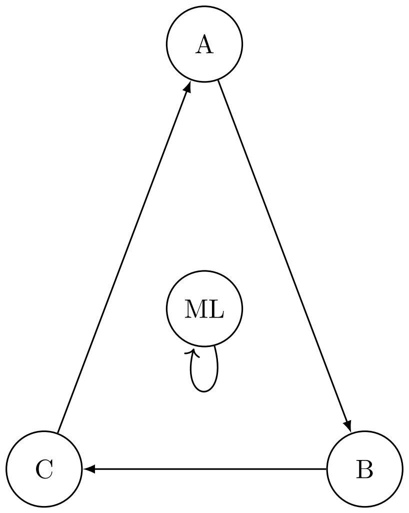

Figure 4: A directed graph with two cycles (ML) and $(\mathrm{A}, \mathrm{B}, \mathrm{C})$.

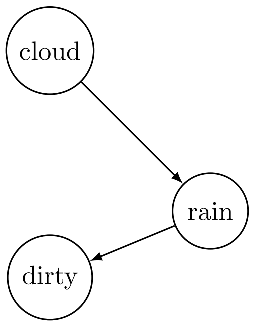

Figure 5: A Bayesian network relating the events of clouds, rain, and my car getting dirty.

represent a relationship that cannot have cycles, meaning that we will be using directed graphs which are **acyclic**. That is, **directed acyclic graphs (DAGs)**.

1.3 Putting it all together

We have now established the preliminaries needed to understand Bayesian networks. As mentioned in the previous section, a Bayesian network is a DAG whose vertices represent observable random quantities and whose edges represent "direct" conditional dependence. To see what we mean by "direct," consider the following example: the presence of clouds in the sky significantly increases the probability that it will rain on a certain day. Rain on a certain day significantly increases the probability that my car gets dirty. In this way, we might say that clouds in the sky increase the probability that my car gets dirty. However, this increase in probability is entirely explained by the fact that the presence of clouds increases the probability of rain. As such, the Bayesian network representation of this dependency would appear as in Figure 5 (note the absence of an arrow from the cloud node to the dirty node).

In the Bayesian network of Figure 5, the random variables represented by the vertices are (1) whether or not there are clouds in the sky, (2) whether or not it rains today, and (3) whether or not my car is dirty today. Notice that each of these variables has as its set of outcomes the set $\{\text{yes}, \text{no}\}$ (that is, for this example, we have no continuous variables at play).

The dependencies depicted in Figure 5 give us some intuitive understanding of the situation, but they fail to tell a complete story. We know that clouds increase the chance of rain, but by how much exactly? This is something that we need to specify in order for our Bayes' net to be complete. We have some options as to how to specify these probabilities:

(bad) Specify the entire joint probability distribution for cloud, rain, and dirty.

(good) Specify the distribution for cloud, the conditional distribution for rain given cloud, and the conditional distribution for dirty given rain.

Why is one of these options bad and the other good? In this scenario, when we have only three variables, we could specify the entire joint distribution of the three variables with $2^3 = 8$ numbers (actually 7, since we know the probabilities must add up to 1). If we instead were to specify the conditional probabilities, we would have

1. Probability of clouds
2. Probability of rain given clouds
3. Probability of rain given no clouds
4. Probability of a dirty car given rain
5. Probability of a dirty car given no rain

So we would need to specify 5 probabilities instead of 7. This doesn't seem like much of a difference.

$$\begin{tikzcd} 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & \cdots \arrow[r] & 28 \arrow[r] & 29 \arrow[r] & 30 \end{tikzcd}$$

Figure 6: A Bayes' net with thirty variables, with each variable dependent only on the "previous" variable.

Variable	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
clouds	0.3	0.7
rain | clouds	0.6	0.4
rain | no clouds	0.05	0.95
dirty car | rain	0.35	0.65
dirty car | no rain	0.1	0.9

Figure 7: The probabilities necessary to fully describe the Bayes' net in Figure 5.

But say instead that we had 30 different variables, each with two outcomes, and suppose our Bayes' net looks as in Figure 6. Then we need only specify

1. The probability of event 1.
2. The probability of event 2 given that event 1 occurs.
3. The probability of event 2 given that event 1 does not occur.

   ⋮

58. The probability of event 30 given that event 29 occurs.
59. The probability of event 30 given that event 29 does not occur.

There are 59 probabilities for us to specify assuming we choose to specify conditional probabilities. On the other hand, if we try to specify the joint distribution directly, then we have $2^{30} - 1 \approx 10$ billion probabilities to specify. Indeed, the structure of a Bayesian network alone can reduce the time it takes for us to make an inference as well as the time it takes for us to learn the parameters of the network. Let's continue with our example of the clouds, the rain, and my potentially dirty car. We already know which probabilities we need to specify. We give those probabilities in Figure 7.

1.3.1 Marginal probabilities with Bayesian networks

Because a Bayesian network gives us the probability distribution of each node conditioned on its parents (which, alongside the independence relations implied by the network structure, is sufficient to determine the joint distribution of all of the variables), we can use a Bayesian network to carry out statistical inference tasks. Let's consider a slightly more complicated version of the previous example: though whether or not it rains has an effect on whether or not my car will be dirty at the end of the day, the dirtiness of my car also depends on whether or not my car was dirty *yesterday*. Additionally, my wheels always get dirty when the street sweeping truck rolls through, but the street sweeping truck often doesn't come when its raining. Our new Bayesian network is shown in Figure 8.

Now we need to specify a bit more information than before in order to complete our Bayes' net. In particular, since the dirtiness of my car now depends on rain, sweep, and yest, we'll need to specify the distribution of my car's dirtiness given all eight possible joint outcomes of those three factors. The probabilities we use are given in Figure 9.

With no additional information, we can calculate the probability that my car will be dirty at the end of the

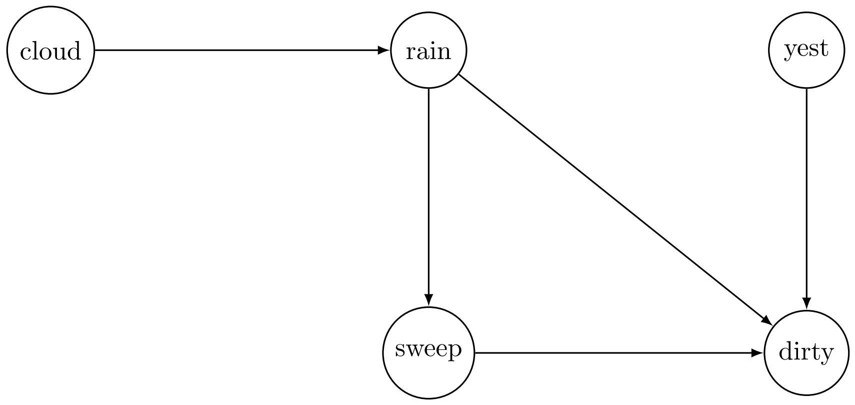

Figure 8: The new Bayes' net for the clouds, the rain, and the dirtiness of my car; building on Figure 5.

cloud		
Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
None	0.3	0.7

rain		
Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
clouds	0.6	0.4
no clouds	0.05	0.95

yest		
Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
None	0.25	0.75

sweep		
Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
rain	0.1	0.9
no clouds	0.3	0.7

dirty		
Condition	$\mathbb{P}$[yes]	$\mathbb{P}$[no]
R, Y, S	0.9	0.01
$R, Y, \neg S$	0.6	0.1
$R, \neg Y, S$	0.6	0.1
$R, \neg Y, \neg S$	0.3	0.7
$\neg R, Y, S$	0.7	0.3
$\neg R, Y, \neg S$	0.5	0.5
$\neg R, \neg Y, S$	0.4	0.6
$\neg R, \neg Y, \neg S$	0.05	0.95

Figure 9: The conditional probabilities defining the relationships between variables in 9.

day:

$$\begin{aligned}
\mathbb{P}[\text{dirty}] &= \sum_{\substack{c,r,y,s,d\in\{0,1\}^5\\ d=1}} \mathbb{P}[\text{cloud} = c, \text{rain} = r, \text{yest} = y, \text{sweep} = s, \text{dirty} = d] \\
&= \sum_{c,r,y,s\in\{0,1\}^4} \mathbb{P}[\text{cloud} = c, \text{rain} = r, \text{yest} = y, \text{sweep} = s, \text{dirty} = 1] \\
&= \sum_{c,r,y,s\in\{0,1\}^4} \mathbb{P}[\text{cloud} = c, \text{rain} = r, \text{yest} = y, \text{sweep} = s] \cdot \mathbb{P}[\text{dirty} = 1 \mid c, r, y, s] \qquad (1) \\
&= \sum_{c,r,y,s\in\{0,1\}^4} \mathbb{P}[\text{cloud} = c, \text{rain} = r, \text{sweep} = s] \cdot \mathbb{P}[\text{yest} = y \mid c, r, s] \cdot \mathbb{P}[\text{dirty} = 1 \mid r, y, s] \qquad (2) \\
&= \sum_{c,r,y,s\in\{0,1\}^4} \mathbb{P}[\text{cloud} = c, \text{rain} = r] \cdot \mathbb{P}[\text{sweep} = s \mid c, r] \cdot \mathbb{P}[\text{yest} = y] \cdot \mathbb{P}[\text{dirty} = 1 \mid r, y, s] \qquad (3) \\
&= \sum_{c,r,y,s\in\{0,1\}^4} \mathbb{P}[\text{cloud} = c] \cdot \mathbb{P}[\text{rain} = r \mid c] \cdot \mathbb{P}[\text{sweep} = s \mid r] \cdot \mathbb{P}[\text{yest} = y] \cdot \mathbb{P}[\text{dirty} = 1 \mid r, y, s] \qquad (4)
\end{aligned}$$

In step (1), we used a simple law of conditional probability. In step (2), we used the same law of conditional probability as well as the fact that, given rain, yest, and sweep, dirty is independent of cloud (that is, cloud has no effect on the probability distribution of dirty once we know the values of rain, yest, and sweep). In step (3), we used that yest is independent of cloud, rain, and sweep as well as the prior law of conditional probability. Finally, in step (4), we used the law of conditional probability once more as well as the fact that, given rain, sweep is independent of cloud.

Now, the astute reader will note that computing a sum over all possible values of the tuple (c, r, y, s) does not scale well at all. The previously mentioned example, in which we needed to specified on the order of 10 billion parameters in order to completely specify a Bayes' net, comes to mind. Indeed, if we try to implement inference with a Bayes' net using the summation above, our code will take obscenely long to run as our Bayes' nets grow. In particular, the time complexity of computing this sum is $O(|S|)$, where S is the set of all values taken by (c, r, y, s). Let's see if we can reduce this with some mathematical manipulations: notice that, for example, $\mathbb{P}[\text{cloud} = c]$ is independent of the values of r, y, s, and d. Applying this idea allows us to write the above sum as an iterated sum:

$$\begin{aligned}
&\sum_{c,r,y,s\in\{0,1\}^4} \mathbb{P}[\text{cloud} = c] \cdot \mathbb{P}[\text{rain} = r \mid c] \cdot \mathbb{P}[\text{sweep} = s \mid r] \cdot \mathbb{P}[\text{yest} = y] \cdot \mathbb{P}[\text{dirty} = 1 \mid r, y, s] \\
&= \sum_{r,y,s\in\{0,1\}^3} \mathbb{P}[\text{sweep} = s \mid r]\mathbb{P}[\text{yest} = y]\mathbb{P}[\text{dirty} = 1 \mid r, y, s] \sum_{c=0}^{1} \mathbb{P}[\text{rain} = r \mid c]\mathbb{P}[\text{cloud} = c]
\end{aligned}$$

What have we accomplished here? Our outer sum is over all values of the tuple (r, y, s). Our inner depends only on r (as it is a sum over all values of c). We can *precompute* all possible values of the inner sum rather than having to recompute the sum for every term. To see this increase in efficiency more clearly, note that in our example above, wherein each variable takes one of only two values, the inner sum will always equal one of two things. In particular, we have

$$\sum_{c=0}^{1} \mathbb{P}[\text{rain} = 1 \mid c]\mathbb{P}[\text{cloud} = c] = 0.035 + 0.18 = 0.215 \qquad \sum_{c=0}^{1} \mathbb{P}[\text{rain} = 0 \mid c]\mathbb{P}[\text{cloud} = c] = 0.665 + 0.12 = 0.785$$

If we were to stick with our original sum over all values of the tuple (c, r, y, s), we would have $2^4 = 16$ total summands. For eight of these summands (when $r = 1$) the value of the inner sum is 0.215, and for the other eight of these summands (when $r = 0$), the value of the inner sum is 0.785. It would be much more efficient if we were to store the values of the inner sum in advance and access them based on the value of r. To do this, we use a *map* (a dictionary or `dict` in Python and a `HashMap` in Java).

Let's analyze the time complexity of our algorithm if we precompute values of the inner sum. For generality, let's say each of the nodes in our Bayesian network has N possible values. The inner sum, in this case, would have N summands (one for each value of c), and since we need to compute the inner sum for each of the N values of r, the precomputation of the inner sum runs in $O(N^2)$ time. When we precompute the inner sum, any further references to its value take constant time. As such, the outer sum is now over all values of (r, y, s). There are N^3 total possible values for this tuple, so the computation of the outer sum runs in $O(N^3)$ time (there is an additional factor in this complexity that depends on the number of factors comprising each summand, but that's not our focus right now). As such, our time complexity with precomputation of the inner sum is $O(N^2 + N^3) = O(N^3)$. This is a noticeable reduction from $O(N^4)$, as even if $N = 2$, we have more-or-less cut the runtime of this process in half.

Even still, the skeptical reader may question how significant our precomputation really is. In the above case, for example, the time complexity still seems like it would scale exponentially with the number of nodes in our Bayes' net. Even though reducing from a big exponential to a smaller exponential is nice, it's not as nice as, say, reducing from exponential time to polynomial time. Let's revisit the example shown in Figure 6. Let's call the variables $V_1, V_2, \ldots, V_{30}$. In that case, we would have

$$\mathbb{P}[V_{30} = a] = \sum_{\substack{(v_1,\ldots,v_{30})\in S \\ v_{30}=a}} \mathbb{P}[V_1 = v_1, V_2 = v_2, \ldots, V_{30} = v_{30}]$$

Using the dependencies implied by the structure of our network (see Figure 6), we can write

$$\begin{aligned}\mathbb{P}[V_{30} = a] &= \sum_{\substack{(v_1,\ldots,v_{30})\in S \\ v_{30}=a}} \mathbb{P}[V_1 = v_1]\mathbb{P}[V_2 = v_2 \mid V_1 = v_1]\mathbb{P}[V_3 = v_3 \mid V_2 = v_2]\cdots\mathbb{P}[V_{30} = v_{30} \mid V_{29} = v_{29}] \\ &= \sum_{v_{29}} \mathbb{P}[V_{30} = v_{30} \mid v_{29}] \sum_{v_{28}} \mathbb{P}[V_{29} = v_{29} \mid v_{28}] \sum \cdots \sum_{v_2} \mathbb{P}[V_3 \mid v_2] \sum_{v_1} \mathbb{P}[V_2 \mid v_1]\end{aligned}$$

Let's suppose that instead of two possible values, we have N possible values for each of the 30 variables. For the same reasons as above, the innermost sum takes $O(N^2)$ time to compute for all values of v_2. Because the value of this innermost sum is dependent only on v_2, we will denote it as $\sigma_2(v_2)$. Then we have

$$\mathbb{P}[V_{30} = a] = \sum_{v_{29}} \mathbb{P}[V_{30} = v_{30} \mid v_{29}] \sum_{v_{28}} \mathbb{P}[V_{29} = v_{29} \mid v_{28}] \sum \cdots \sum_{v_2} \mathbb{P}[V_3 \mid v_2]\sigma(v_2)$$

If we precompute the values of σ_2, then the innermost sum in this equation will also take $O(N^2)$ time to compute (since evaluating σ_2 takes constant time – the time for a map access). We call this new inner sum $\sigma_3(v_3)$, as it depends only and entirely on v_3. Substituting $\sigma_3(v_3)$ into the above equation gives us a new innermost sum, which we will call $\sigma_4(v_4)$, that also takes $O(N^2)$ time to compute given precomputation of σ_3. Applying this reasoning over and over again tells us that we can reduce the number of summations in our expression for $\mathbb{P}[V_{30} = a]$ by one at the expense of an $O(N^2)$ computation. Since there are 29 sums in our original expression, we find that computing $\mathbb{P}[V_{30} = a]$ now takes $O(29N^2)$ time. If we had K nodes instead of 30, we end up with an $O(KN^2)$ computation, which is *much* better than the naïve complexity of $O(N^K)$.

1.3.2 Variable elimination algorithm

We have now seen two examples of how the structure of our Bayesian networks and clever use of precomputation can speed up the time it takes for us to compute marginal probabilities. We now try to describe this method in general. This section is going to have some dense mathematical notation, and it is not necessary to understand every symbol at play in order to get the idea of the algorithm. You will gain practical experience with this algorithm in the notebook assignment. Recall that the first thing we did in both cases was reexpress the joint probability as the product of conditional probabilities. If we have variables $V_1, \ldots, V_n$, then we can consider the conditional probability

$$\mathbb{P}[V_{i_1} = v_{i_1}, \ldots, V_{i_k} = v_{i_k} \mid V_{j_1} = v_{j_1}, \ldots, V_{j_\ell} = v_{j_\ell}] \tag{5}$$

to be a function $\pi(v_{i_1}, \ldots, v_{i_k}, v_{j_1}, v_{j_\ell})$. Then the structure of our Bayesian network will give us an equality of the form

$$\mathbb{P}[V_1 = a_1, V_2 = a_2, \ldots, V_n = a_n] = \prod_{i=1}^{s} \pi_i$$

where π_i is a function of the form (5) for $i = 1, \dots s$. We can then write

$$\mathbb{P}[V_{k_1} = a_{k_1}, \dots, V_{k_r} = a_{k_r}] = \sum_{\substack{v_1,\dots,v_n \in S \\ v_{k_1}=a_{k_1},\dots,v_{k_r}=a_{k_r}}} \mathbb{P}[V_1 = v_1, V_2 = v_2, \dots, V_n = v_n] = \sum_{\substack{v_1,\dots,v_n \in S \\ v_{k_1}=a_{k_1},\dots,v_{k_r}=a_{k_r}}} \prod_{i=1}^{s} \pi_i$$

Now we can try the technique of the previous section where we introduced inner sums that could be precomputed. For any variable V_j, we can there are sum functions π_i for which v_j is a parameter and some for which it is not (let's also assume that V_j doesn't appear on the left-hand side of the above equation, so that V_j can actually vary). Let F_j (F for "function family") be the set of functions π_i in the above product that have V_j as a parameter. Then we have

$$\mathbb{P}[V_{k_1} = a_{k_1}, \dots, V_{k_r} = a_{k_r}] = \sum_{\substack{v_1,\dots,v_n \in S \\ v_{k_1}=a_{k_1},\dots,v_{k_r}=a_{k_r}}} \prod_{i=1}^{s} \pi_i = \sum_{\substack{v_1,\dots,v_n \in S \\ v_{k_1}=a_{k_1},\dots,v_{k_r}=a_{k_r}}} \left(\prod_{\pi_i \in F_j} \pi_i \right) \left(\prod_{\pi_i \notin F_j} \pi_i \right)$$

and now note that the functions $\pi_i \notin F_j$ have nothing to do with V_j, which means we can split up our sum as

$$\mathbb{P}[V_{k_1} = a_{k_1}, \dots, V_{k_r} = a_{k_r}] = \sum_{\substack{v_1,\dots,v_{j-1},v_{j+1},\dots,v_n \\ v_{k_1}=a_{k_1},\dots,v_{k_r}=a_{k_r}}} \prod_{\pi_i \notin F_j} \pi_i \sum_{v_j} \prod_{\pi_i \in F_j} \pi_i$$

Now, the functions $\pi_i \in F_j$ could depend on any number of other variables in our Bayes' net. Let R_j (R for "relative") be the set of variables V_j which are a parameter of at least one function in F_j, and let C_j be the set of all ordered tuples of values taken on by the variables in F_j (so that $|C_j|$ is the "count" of values to precompute). Then the value of the inner sum above is determined by the values of these variables. The number of values that need to be precomputed for this inner sum is then equal to the number of possible values of all variables in F_j. Each term of this inner sum takes $O(|F_j|)$ time to compute (as each term is the product of the functions in F_j, and we have $|C_j|$ terms to precompute. Note that $|C_j|$ varies roughly as $N^{|R_j|}$, where N is the average number of possible values of a variable V_i.

We can denote this inner sum by σ_1 and then proceed similarly. That is, when we replace the inner sum by $\sigma_1(V_i \in R_j)$, we have an expression for the probability that is entirely in terms of $V_1, \dots, V_{j-1}, V_{j+1}, \dots, V_n$. In other words, we have "eliminated a variable" from our summation via precomputation that costs us $O(|C_j| \cdot |F_j|)$ time. We can then select another variable V_k and, by nearly the same methodology, eliminate it. The cost of this additional precomputation *in light of* the precomputation of the values of σ_1 can be denoted by $O(|C_k^{(1)}| \cdot |F_k^{(1)}|$, where the superscript (1) indicates that we exclude any functions π_i that include V_j, the variable that we already removed. We can continue this process until we have removed all variables. Say we remove the variables $V_{j_1}, V_{j_2}, \dots, V_{j_n}$ in that order. Then the time complexity of this algorithm will be

$$O\left(\sum_{i=1}^{n} \left| C_{j_i}^{(i-1)} \right| \cdot \left| F_{j_i}^{(i-1)} \right| \right) \approx O\left(\sum_{i=1}^{n} N^{\left| R_{j_i}^{(i-1)} \right|} \cdot \left| F_{j_i}^{(i-1)} \right| \right)$$

It's hard to speak of what this complexity "is" in general. In the thirty-node case above wherein each variable depended on precisely one other variable, the order $\left| R_{j_i}^{(i-1)} \right|$ was at most 2 for all i (it was actually exactly 2), which gave us a time complexity quadratic in N. However, in networks with more edges, the value of $\left| R_{j_i}^{(i-1)} \right|$ can be quite high. If we have some condition such as

each of the n nodes is connected to at least αn other nodes; $\alpha \in (0, 1]$

then we end up with an exponential-time algorithm in n, the number of nodes. If the details of this algorithm are unclear, no worries. You will be implementing the algorithm (with help) in the notebook.

Cloud	Rain	Sweep	Dirty	yest
T	T	F	T	F
T	F	F	T	T
F	T	T	F	T
T	F	F	T	F
F	F	T	T	F
F	F	F	F	T
F	T	T	F	T
T	T	T	T	T

Figure 10: Small random Data set for given Bayes Net, with complete entries.

1.3.3 Inference and belief updates

Now that we have established a method for somewhat efficiently calculating marginal probabilities using Bayes' nets, we can use Bayes' law to give general conditional probabilities. In particular, we have

$$\mathbb{P}[V_{i_1} = v_{i_1}, \ldots, V_{i_\ell} = v_{i_\ell} \mid V_{j_1} = v_{j_1}, \ldots, V_{j_k} = v_{j_k}] = \frac{\mathbb{P}[V_{i_1} = v_{i_1}, \ldots, V_{i_\ell} = v_{i_\ell}, V_{j_1} = v_{j_1}, \ldots, V_{j_k} = v_{j_k}]}{\mathbb{P}[V_{j_1} = v_{j=1}, \ldots, V_{j_k} = v_{j_k}]}$$

from Bayes' law, and variable elimination allows us to compute the numerator and the denominator.

1.3.4 Parameter Learning for known Bayesian Networks with Complete Data

As seen above, a necessary key for preforming inferences is the conditional probabilities table, which store the true underlining distribution of the data as seen in Figure.9. However, we find that the majority of the these probabilities and true underlining conditional dependencies are unknown to us. In order to solve this problem we must be able to train our network to learn these missing parameters. For simplicity we will begin with training our model given a complete data set, i.e. no data entries contain missing features.

The parameters we wish to train are represented by the entries in our conditional probabilities table, which stores the probability of every possible event for each node within our graph, conditioned on the join between all of its parent nodes. Every node n should contain the number of entries i where $i = (B_j)^p$ and p is equal to the number of parents to that Node and B_j is equal to the number of values the feature of the parent can take.

$$\theta_i^n = \mathbb{P}[N = n|S = s_i]$$

Theta represents a vector of i length containing the parameters we wish to train for each Node N=n, where S represent the vector of every possible condition from the parents of N.

With only training data we are able to approximate these conditional probabilities by solving for the empirical distribution for each event via our data. Given our data set D we can calculate the empirical distribution as the number of entries in D which satisfy an event alpha over the total number of entries.

$$\mathbb{P}[\alpha] = \frac{\#D(\alpha)}{\#D(total)}$$

With in Bayesian networks we know the events which we wish to solve for are conditional probabilities. The only data entries which satisfy these conditions are where N=n and $S = S_i$. Also we are now conditioning on $S = S_i$, so our domain is reduced to the total evens which satisfy $S = S_i$

$$\theta_i^n = \mathbb{P}[N = n|S = s_i] = \frac{\#D(N = n, S = S_i)}{\#D(S = S_i)}$$

For example consider the event, probability of Dirty, given not rain, not sweep, and yest.

$$\theta_i(Dirty) = \mathbb{P}[Dirty|\neg R, \neg S, Y] = \frac{\#D(Dirty, \neg R, \neg S, Y)}{\#D(\neg R, \neg S, Y)} = \frac{1}{2}$$

With large enough data set we can be assured that our empirical distributions will converge to the maximum likelihood estimation(MLE) of the true underling distributions.

However, a large data set is not always available to us and often we will find that not all possible events can be found in within every data set. With our current set up is an event is missing from our data set we assign a zero probability to that event, which is often wrong. To account for this error we can us a technique know as Laplace Smoothing. What Laplace smoothing does for our Bayesian network is assume that every event occurs K times.

$$(Laplace)\theta_i^n = \frac{\#D(N = n, S = S_i) + K}{\#D(S = S_i) + (K * B)}$$

Where B is equal to the domain of Node for this parameter.

1.3.5 Structure learning of Bayesian Networks with Complete Data

So far, we have seen how we can use a Bayes Net to perform inference if we know the structure and the conditional probability table (CPT) entries for the Bayes Net. The CPT entries are either known to us, and if they aren't we now know how to estimate them using MLE and complete data.

On the other hand, in all the scenarios we have considered so far, we have had the structure of the Bayes Net given to us. Often, the structure may be known to us via domain knowledge or any other form of prior information, however what if we do not know the structure of the Bayes Net? Can we use complete data to estimate the structure like we estimated CPT entries? The answer to this question, it turns out, is yes - and this section will briefly cover how we can do so.

Remember, a Bayes Net is a **DAG**. So, using the same idea as in parameter learning, we want to find a DAG $\boldsymbol{G}$, that maximizes the likelihood of the data $\boldsymbol{D}$ we have.

Extending the same ideas of maximum likelihood from parameter estimation, we can formulate our problem as searching over all possible DAG's G, and returning the one that maximizes the likelihood of our complete data. A mathematical expression for our problem then is:

$$\hat{G} = \arg\max_G \ \ell\ell(G; D) = \arg\max_G \ \Pi_{d \in D}\mathbb{P}[d|G]$$

Where Ĝ is the predicted structure that we return.

Let us know shift our focus to the various methods that exist for recovering the structure of a Bayes Net. They are:

1. **Search and Score** - This method follows pretty directly from our discussion of likelihood maximization. Under this method, we look at possible DAG's G, and for each G we assign it a score of $\ell\ell(G; D)$. We then go on to pick the G that had the highest score. Note: The exact math behind how this likelihood/score is calculated is beyond the scope of this course. As you can see, the run time of this algorithm is exponential in the number of variables in our Bayes Net as the number of DAG's possible is exponential with respect to the number of variables.

2. **Approximate learning algorithms** - Clearly, as the number of variables in a Bayes Net starts to increase, the number of possible DAG's (ie. the search space) becomes to big for search and store to be feasible. Thus, we turn to approximate algorithms - algorithms that reduce the search space of Score and Search via methods like:

   (a) Imposing constraints on the structure of the output. (Eg: Has to be a tree, cannot have more than x edges, etc.) A prime example of this is the **Chow Liu Trees Algorithm**. The algorithm always outputs a tree as the structure and its procedure is pretty easy to understand:

      i. Start off with n nodes where n is the number of variables in the Bayes Net. Add every edge (u, v) possible to the graph with a weight equal to the mutual information $(I(u, v))$ between the two variables in question. A higher weight means a higher mutual dependency between the two variables.

      $$I(u, v) = \Sigma_{u \in U} \Sigma_{v \in V} \mathbb{P}[u, v] \log_2 \frac{\mathbb{P}[u, v]}{\mathbb{P}[u]\mathbb{P}[v]}$$

      ii. Using all the edges and their weights, construct a maximum weight spanning tree - ie. choose the edges that form a sparse fully connected structure while trying to use edges to encode high mutual dependencies between variables.

      iii. Choose any root node and assign edge directions such that you have a Directed Tree where each node only has one parent except the root node.

      You should be able to see that the run time of the Chow Liu Tree Algorithm is $O(v^2)$ where v is the number of variables in the Bayes Net.

   (b) Heuristics and A* search

   (c) Dynamic Programming

   (d) Hill Climbing Methods, Simulated Annealing etc.

   As you may expect, with a reduced number of states to search, these approximate algorithms will run much quicker than the search and score algorithm. However, the cost of this efficiency is that since approximate algorithms do not search all states, they may not find the best, or most accurate structure.

3. **Constraint learning algorithms**- This family of algorithms involve calculating correlation or co-occurrence to identify undirected backbones of edges that could exist in $\hat{G}$. We prune these edges systematically until a DAG is reached and then returned. Unfortunately, this is not as intuitive as the previous algorithms as there is no probabilistic model or function over here. These class of algorithms have been described as you should be aware they exist but are not in scope for this class.

In conclusion, we see that often we may not know the structure of a Bayes Net and may have to learn it from complete data. The learning process consists of finding the DAG that maximizes the log likelihood of the data that has been given to us.

We also saw that the number of DAG's is exponential with number of variables, so it may be infeasible to go through each potential DAG. To avoid doing this, we resort to approximate methods such as the Chow Liu Tree Algorithm. With approximate algorithms, there is an inherent trade off between run time of algorithms and how 'good' a structure they can recover.

Thus, we now know how to estimate structure, parameters, and run inference in the context of Bayesian Networks.