

20 LEARNING PROBABILISTIC MODELS

In which we view learning as a form of uncertain reasoning from observations.

Chapter 13 pointed out the prevalence of uncertainty in real environments. Agents can handle uncertainty by using the methods of probability and decision theory, but first they must learn their probabilistic theories of the world from experience. This chapter explains how they can do that, by formulating the learning task itself as a process of probabilistic inference (Section 20.1). We will see that a Bayesian view of learning is extremely powerful, providing general solutions to the problems of noise, overfitting, and optimal prediction. It also takes into account the fact that a less-than-omniscient agent can never be certain about which theory of the world is correct, yet must still make decisions by using some theory of the world.

We describe methods for learning probability models—primarily Bayesian networks—in Sections 20.2 and 20.3. Some of the material in this chapter is fairly mathematical, although the general lessons can be understood without plunging into the details. It may benefit the reader to review Chapters 13 and 14 and peek at Appendix A.

20.1 STATISTICAL LEARNING

The key concepts in this chapter, just as in Chapter 18, are **data** and **hypotheses**. Here, the data are **evidence**—that is, instantiations of some or all of the random variables describing the domain. The hypotheses in this chapter are probabilistic theories of how the domain works, including logical theories as a special case.

Consider a simple example. Our favorite Surprise candy comes in two flavors: cherry (yum) and lime (ugh). The manufacturer has a peculiar sense of humor and wraps each piece of candy in the same opaque wrapper, regardless of flavor. The candy is sold in very large bags, of which there are known to be five kinds—again, indistinguishable from the outside:

- h_1 : 100% cherry,
- h_2 : 75% cherry + 25% lime,
- h_3 : 50% cherry + 50% lime,
- h_4 : 25% cherry + 75% lime,
- h_5 : 100% lime .

Given a new bag of candy, the random variable H (for *hypothesis*) denotes the type of the bag, with possible values h_1 through h_5 . H is not directly observable, of course. As the pieces of candy are opened and inspected, data are revealed— D_1, D_2, \dots, D_N , where each D_i is a random variable with possible values *cherry* and *lime*. The basic task faced by the agent is to predict the flavor of the next piece of candy.¹ Despite its apparent triviality, this scenario serves to introduce many of the major issues. The agent really does need to infer a theory of its world, albeit a very simple one.

BAYESIAN LEARNING

Bayesian learning simply calculates the probability of each hypothesis, given the data, and makes predictions on that basis. That is, the predictions are made by using *all* the hypotheses, weighted by their probabilities, rather than by using just a single “best” hypothesis. In this way, learning is reduced to probabilistic inference. Let \mathbf{D} represent all the data, with observed value \mathbf{d} ; then the probability of each hypothesis is obtained by Bayes’ rule:

$$P(h_i | \mathbf{d}) = \alpha P(\mathbf{d} | h_i) P(h_i) . \quad (20.1)$$

Now, suppose we want to make a prediction about an unknown quantity X . Then we have

$$\mathbf{P}(X | \mathbf{d}) = \sum_i \mathbf{P}(X | \mathbf{d}, h_i) \mathbf{P}(h_i | \mathbf{d}) = \sum_i \mathbf{P}(X | h_i) P(h_i | \mathbf{d}) , \quad (20.2)$$

where we have assumed that each hypothesis determines a probability distribution over X . This equation shows that predictions are weighted averages over the predictions of the individual hypotheses. The hypotheses themselves are essentially “intermediaries” between the raw data and the predictions. The key quantities in the Bayesian approach are the **hypothesis prior**, $P(h_i)$, and the **likelihood** of the data under each hypothesis, $P(\mathbf{d} | h_i)$.

HYPOTHESIS PRIOR

LIKELIHOOD

For our candy example, we will assume for the time being that the prior distribution over h_1, \dots, h_5 is given by $\langle 0.1, 0.2, 0.4, 0.2, 0.1 \rangle$, as advertised by the manufacturer. The likelihood of the data is calculated under the assumption that the observations are **i.i.d.** (see page 708), so that

$$P(\mathbf{d} | h_i) = \prod_j P(d_j | h_i) . \quad (20.3)$$

For example, suppose the bag is really an all-lime bag (h_5) and the first 10 candies are all lime; then $P(\mathbf{d} | h_3)$ is 0.5^{10} , because half the candies in an h_3 bag are lime.² Figure 20.1(a) shows how the posterior probabilities of the five hypotheses change as the sequence of 10 lime candies is observed. Notice that the probabilities start out at their prior values, so h_3 is initially the most likely choice and remains so after 1 lime candy is unwrapped. After 2 lime candies are unwrapped, h_4 is most likely; after 3 or more, h_5 (the dreaded all-lime bag) is the most likely. After 10 in a row, we are fairly certain of our fate. Figure 20.1(b) shows the predicted probability that the next candy is lime, based on Equation (20.2). As we would expect, it increases monotonically toward 1.

¹ Statistically sophisticated readers will recognize this scenario as a variant of the **urn-and-ball** setup. We find urns and balls less compelling than candy; furthermore, candy lends itself to other tasks, such as deciding whether to trade the bag with a friend—see Exercise 20.2.

² We stated earlier that the bags of candy are very large; otherwise, the i.i.d. assumption fails to hold. Technically, it is more correct (but less hygienic) to rewrap each candy after inspection and return it to the bag.

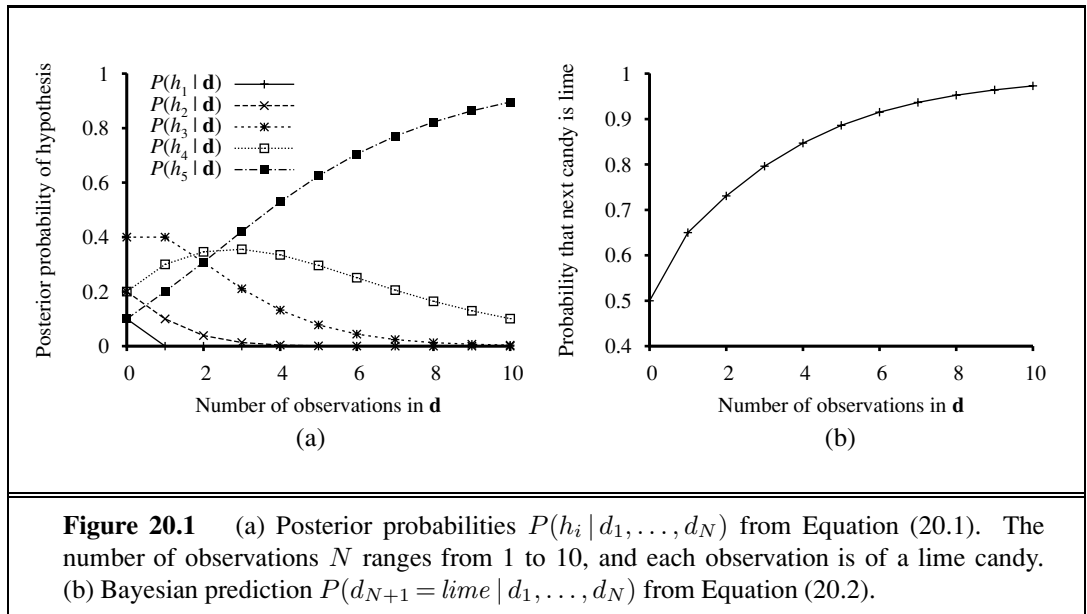


Figure 20.1 (a) Posterior probabilities $P(h_i | d_1, \dots, d_N)$ from Equation (20.1). The number of observations N ranges from 1 to 10, and each observation is of a lime candy. (b) Bayesian prediction $P(d_{N+1} = \text{lime} | d_1, \dots, d_N)$ from Equation (20.2).



The example shows that *the Bayesian prediction eventually agrees with the true hypothesis*. This is characteristic of Bayesian learning. For any fixed prior that does not rule out the true hypothesis, the posterior probability of any false hypothesis will, under certain technical conditions, eventually vanish. This happens simply because the probability of generating “uncharacteristic” data indefinitely is vanishingly small. (This point is analogous to one made in the discussion of PAC learning in Chapter 18.) More important, the Bayesian prediction is *optimal*, whether the data set be small or large. Given the hypothesis prior, any other prediction is expected to be correct less often.

The optimality of Bayesian learning comes at a price, of course. For real learning problems, the hypothesis space is usually very large or infinite, as we saw in Chapter 18. In some cases, the summation in Equation (20.2) (or integration, in the continuous case) can be carried out tractably, but in most cases we must resort to approximate or simplified methods.

A very common approximation—one that is usually adopted in science—is to make predictions based on a single *most probable* hypothesis—that is, an h_i that maximizes $P(h_i | \mathbf{d})$. This is often called a **maximum a posteriori** or MAP (pronounced “em-ay-pee”) hypothesis. Predictions made according to an MAP hypothesis h_{MAP} are approximately Bayesian to the extent that $\mathbf{P}(X | \mathbf{d}) \approx \mathbf{P}(X | h_{\text{MAP}})$. In our candy example, $h_{\text{MAP}} = h_5$ after three lime candies in a row, so the MAP learner then predicts that the fourth candy is lime with probability 1.0—a much more dangerous prediction than the Bayesian prediction of 0.8 shown in Figure 20.1(b). As more data arrive, the MAP and Bayesian predictions become closer, because the competitors to the MAP hypothesis become less and less probable.

Although our example doesn’t show it, finding MAP hypotheses is often much easier than Bayesian learning, because it requires solving an optimization problem instead of a large summation (or integration) problem. We will see examples of this later in the chapter.

In both Bayesian learning and MAP learning, the hypothesis prior $P(h_i)$ plays an important role. We saw in Chapter 18 that **overfitting** can occur when the hypothesis space is too expressive, so that it contains many hypotheses that fit the data set well. Rather than placing an arbitrary limit on the hypotheses to be considered, Bayesian and MAP learning methods use the prior to *penalize complexity*. Typically, more complex hypotheses have a lower prior probability—in part because there are usually many more complex hypotheses than simple hypotheses. On the other hand, more complex hypotheses have a greater capacity to fit the data. (In the extreme case, a lookup table can reproduce the data exactly with probability 1.) Hence, the hypothesis prior embodies a tradeoff between the complexity of a hypothesis and its degree of fit to the data.

We can see the effect of this tradeoff most clearly in the logical case, where H contains only *deterministic* hypotheses. In that case, $P(\mathbf{d} | h_i)$ is 1 if h_i is consistent and 0 otherwise. Looking at Equation (20.1), we see that h_{MAP} will then be the *simplest logical theory that is consistent with the data*. Therefore, maximum *a posteriori* learning provides a natural embodiment of Ockham's razor.

Another insight into the tradeoff between complexity and degree of fit is obtained by taking the logarithm of Equation (20.1). Choosing h_{MAP} to maximize $P(\mathbf{d} | h_i)P(h_i)$ is equivalent to minimizing

$$-\log_2 P(\mathbf{d} | h_i) - \log_2 P(h_i) .$$

Using the connection between information encoding and probability that we introduced in Chapter 18.3.4, we see that the $-\log_2 P(h_i)$ term equals the number of bits required to specify the hypothesis h_i . Furthermore, $-\log_2 P(\mathbf{d} | h_i)$ is the additional number of bits required to specify the data, given the hypothesis. (To see this, consider that no bits are required if the hypothesis predicts the data exactly—as with h_5 and the string of lime candies—and $\log_2 1 = 0$.) Hence, MAP learning is choosing the hypothesis that provides maximum *compression* of the data. The same task is addressed more directly by the **minimum description length**, or MDL, learning method. Whereas MAP learning expresses simplicity by assigning higher probabilities to simpler hypotheses, MDL expresses it directly by counting the bits in a binary encoding of the hypotheses and data.

A final simplification is provided by assuming a **uniform** prior over the space of hypotheses. In that case, MAP learning reduces to choosing an h_i that maximizes $P(\mathbf{d} | h_i)$. This is called a **maximum-likelihood** (ML) hypothesis, h_{ML} . Maximum-likelihood learning is very common in statistics, a discipline in which many researchers distrust the subjective nature of hypothesis priors. It is a reasonable approach when there is no reason to prefer one hypothesis over another *a priori*—for example, when all hypotheses are equally complex. It provides a good approximation to Bayesian and MAP learning when the data set is large, because the data swamps the prior distribution over hypotheses, but it has problems (as we shall see) with small data sets.



20.2 LEARNING WITH COMPLETE DATA

DENSITY ESTIMATION

The general task of learning a probability model, given data that are assumed to be generated from that model, is called **density estimation**. (The term applied originally to probability density functions for continuous variables, but is used now for discrete distributions too.)

COMPLETE DATA

PARAMETER
LEARNING

This section covers the simplest case, where we have **complete data**. Data are complete when each data point contains values for every variable in the probability model being learned. We focus on **parameter learning**—finding the numerical parameters for a probability model whose structure is fixed. For example, we might be interested in learning the conditional probabilities in a Bayesian network with a given structure. We will also look briefly at the problem of learning structure and at nonparametric density estimation.

20.2.1 Maximum-likelihood parameter learning: Discrete models

Suppose we buy a bag of lime and cherry candy from a new manufacturer whose lime–cherry proportions are completely unknown; the fraction could be anywhere between 0 and 1. In that case, we have a continuum of hypotheses. The **parameter** in this case, which we call θ , is the proportion of cherry candies, and the hypothesis is h_θ . (The proportion of limes is just $1 - \theta$.) If we assume that all proportions are equally likely *a priori*, then a maximum-likelihood approach is reasonable. If we model the situation with a Bayesian network, we need just one random variable, *Flavor* (the flavor of a randomly chosen candy from the bag). It has values *cherry* and *lime*, where the probability of *cherry* is θ (see Figure 20.2(a)). Now suppose we unwrap N candies, of which c are cherries and $\ell = N - c$ are limes. According to Equation (20.3), the likelihood of this particular data set is

$$P(\mathbf{d} | h_\theta) = \prod_{j=1}^N P(d_j | h_\theta) = \theta^c \cdot (1 - \theta)^\ell.$$

LOG LIKELIHOOD

The maximum-likelihood hypothesis is given by the value of θ that maximizes this expression. The same value is obtained by maximizing the **log likelihood**,

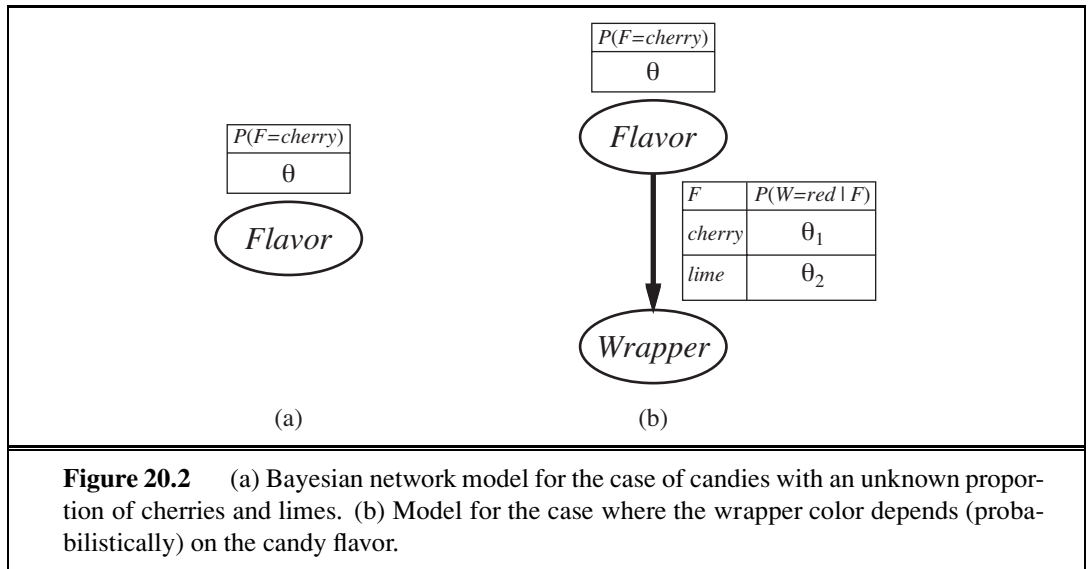
$$L(\mathbf{d} | h_\theta) = \log P(\mathbf{d} | h_\theta) = \sum_{j=1}^N \log P(d_j | h_\theta) = c \log \theta + \ell \log(1 - \theta).$$

(By taking logarithms, we reduce the product to a sum over the data, which is usually easier to maximize.) To find the maximum-likelihood value of θ , we differentiate L with respect to θ and set the resulting expression to zero:

$$\frac{dL(\mathbf{d} | h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1 - \theta} = 0 \quad \Rightarrow \quad \theta = \frac{c}{c + \ell} = \frac{c}{N}.$$

In English, then, the maximum-likelihood hypothesis h_{ML} asserts that the actual proportion of cherries in the bag is equal to the observed proportion in the candies unwrapped so far!

It appears that we have done a lot of work to discover the obvious. In fact, though, we have laid out one standard method for maximum-likelihood parameter learning, a method with broad applicability:



1. Write down an expression for the likelihood of the data as a function of the parameter(s).
2. Write down the derivative of the log likelihood with respect to each parameter.
3. Find the parameter values such that the derivatives are zero.

The trickiest step is usually the last. In our example, it was trivial, but we will see that in many cases we need to resort to iterative solution algorithms or other numerical optimization techniques, as described in Chapter 4. The example also illustrates a significant problem with maximum-likelihood learning in general: *when the data set is small enough that some events have not yet been observed—for instance, no cherry candies—the maximum-likelihood hypothesis assigns zero probability to those events*. Various tricks are used to avoid this problem, such as initializing the counts for each event to 1 instead of 0.

Let us look at another example. Suppose this new candy manufacturer wants to give a little hint to the consumer and uses candy wrappers colored red and green. The *Wrapper* for each candy is selected *probabilistically*, according to some unknown conditional distribution, depending on the flavor. The corresponding probability model is shown in Figure 20.2(b). Notice that it has three parameters: θ , θ_1 , and θ_2 . With these parameters, the likelihood of seeing, say, a cherry candy in a green wrapper can be obtained from the standard semantics for Bayesian networks (page 513):

$$\begin{aligned}
 &P(\text{Flavor} = \text{cherry}, \text{Wrapper} = \text{green} \mid h_{\theta, \theta_1, \theta_2}) \\
 &= P(\text{Flavor} = \text{cherry} \mid h_{\theta, \theta_1, \theta_2}) P(\text{Wrapper} = \text{green} \mid \text{Flavor} = \text{cherry}, h_{\theta, \theta_1, \theta_2}) \\
 &= \theta \cdot (1 - \theta_1) .
 \end{aligned}$$

Now we unwrap N candies, of which c are cherries and ℓ are limes. The wrapper counts are as follows: r_c of the cherries have red wrappers and g_c have green, while r_ℓ of the limes have red and g_ℓ have green. The likelihood of the data is given by

$$P(\mathbf{d} \mid h_{\theta, \theta_1, \theta_2}) = \theta^c (1 - \theta)^\ell \cdot \theta_1^{r_c} (1 - \theta_1)^{g_c} \cdot \theta_2^{r_\ell} (1 - \theta_2)^{g_\ell} .$$

This looks pretty horrible, but taking logarithms helps:

$$L = [c \log \theta + \ell \log(1 - \theta)] + [r_c \log \theta_1 + g_c \log(1 - \theta_1)] + [r_\ell \log \theta_2 + g_\ell \log(1 - \theta_2)] .$$

The benefit of taking logs is clear: the log likelihood is the sum of three terms, each of which contains a single parameter. When we take derivatives with respect to each parameter and set them to zero, we get three independent equations, each containing just one parameter:

$$\begin{aligned} \frac{\partial L}{\partial \theta} &= \frac{c}{\theta} - \frac{\ell}{1-\theta} = 0 & \Rightarrow & \theta = \frac{c}{c+\ell} \\ \frac{\partial L}{\partial \theta_1} &= \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 & \Rightarrow & \theta_1 = \frac{r_c}{r_c+g_c} \\ \frac{\partial L}{\partial \theta_2} &= \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1-\theta_2} = 0 & \Rightarrow & \theta_2 = \frac{r_\ell}{r_\ell+g_\ell} . \end{aligned}$$

The solution for θ is the same as before. The solution for θ_1 , the probability that a cherry candy has a red wrapper, is the observed fraction of cherry candies with red wrappers, and similarly for θ_2 .

These results are very comforting, and it is easy to see that they can be extended to any Bayesian network whose conditional probabilities are represented as tables. The most important point is that, *with complete data, the maximum-likelihood parameter learning problem for a Bayesian network decomposes into separate learning problems, one for each parameter.* (See Exercise 20.6 for the nontabulated case, where each parameter affects several conditional probabilities.) The second point is that the parameter values for a variable, given its parents, are just the observed frequencies of the variable values for each setting of the parent values. As before, we must be careful to avoid zeroes when the data set is small.



20.2.2 Naive Bayes models

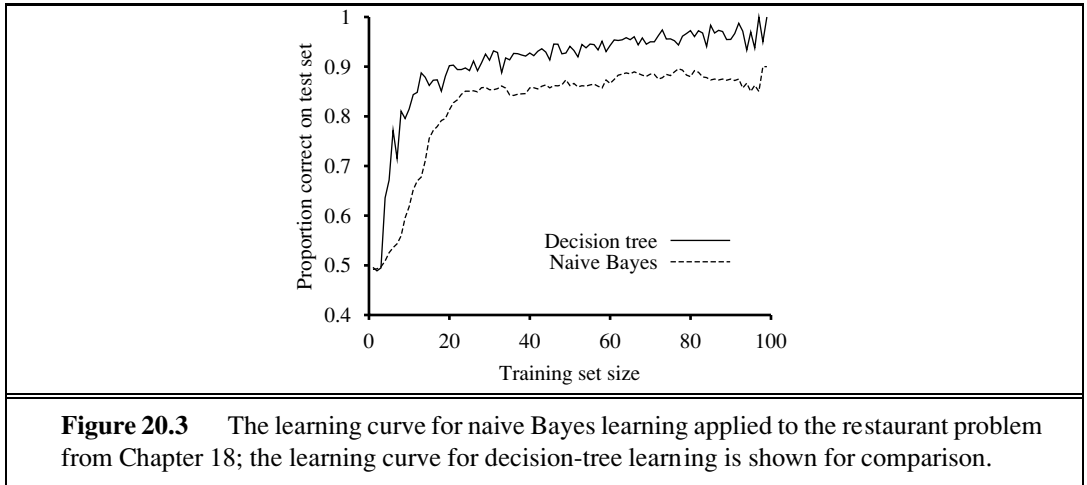
Probably the most common Bayesian network model used in machine learning is the **naive Bayes** model first introduced on page 499. In this model, the “class” variable C (which is to be predicted) is the root and the “attribute” variables X_i are the leaves. The model is “naive” because it assumes that the attributes are conditionally independent of each other, given the class. (The model in Figure 20.2(b) is a naive Bayes model with class *Flavor* and just one attribute, *Wrapper*.) Assuming Boolean variables, the parameters are

$$\theta = P(C = \text{true}), \theta_{i1} = P(X_i = \text{true} \mid C = \text{true}), \theta_{i2} = P(X_i = \text{true} \mid C = \text{false}).$$

The maximum-likelihood parameter values are found in exactly the same way as for Figure 20.2(b). Once the model has been trained in this way, it can be used to classify new examples for which the class variable C is unobserved. With observed attribute values x_1, \dots, x_n , the probability of each class is given by

$$\mathbf{P}(C \mid x_1, \dots, x_n) = \alpha \mathbf{P}(C) \prod_i \mathbf{P}(x_i \mid C) .$$

A deterministic prediction can be obtained by choosing the most likely class. Figure 20.3 shows the learning curve for this method when it is applied to the restaurant problem from Chapter 18. The method learns fairly well but not as well as decision-tree learning; this is presumably because the true hypothesis—which is a decision tree—is not representable exactly using a naive Bayes model. Naive Bayes learning turns out to do surprisingly well in a wide range of applications; the boosted version (Exercise 20.4) is one of the most effective



general-purpose learning algorithms. Naive Bayes learning scales well to very large problems: with n Boolean attributes, there are just $2n + 1$ parameters, and *no search is required to find h_{ML} , the maximum-likelihood naive Bayes hypothesis*. Finally, naive Bayes learning systems have no difficulty with noisy or missing data and can give probabilistic predictions when appropriate.

20.2.3 Maximum-likelihood parameter learning: Continuous models

Continuous probability models such as the **linear Gaussian** model were introduced in Section 14.3. Because continuous variables are ubiquitous in real-world applications, it is important to know how to learn the parameters of continuous models from data. The principles for maximum-likelihood learning are identical in the continuous and discrete cases.

Let us begin with a very simple case: learning the parameters of a Gaussian density function on a single variable. That is, the data are generated as follows:

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

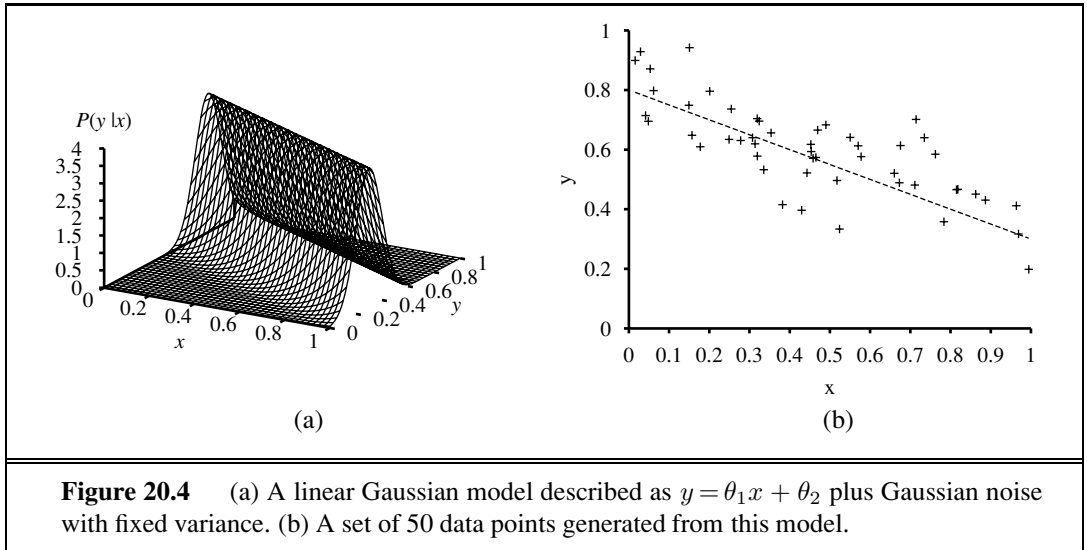
The parameters of this model are the mean μ and the standard deviation σ . (Notice that the normalizing “constant” depends on σ , so we cannot ignore it.) Let the observed values be x_1, \dots, x_N . Then the log likelihood is

$$L = \sum_{j=1}^N \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} = N(-\log \sqrt{2\pi} - \log \sigma) - \sum_{j=1}^N \frac{(x_j - \mu)^2}{2\sigma^2}.$$

Setting the derivatives to zero as usual, we obtain

$$\begin{aligned} \frac{\partial L}{\partial \mu} &= -\frac{1}{\sigma^2} \sum_{j=1}^N (x_j - \mu) = 0 & \Rightarrow \mu &= \frac{\sum_j x_j}{N} \\ \frac{\partial L}{\partial \sigma} &= -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^N (x_j - \mu)^2 = 0 & \Rightarrow \sigma &= \sqrt{\frac{\sum_j (x_j - \mu)^2}{N}}. \end{aligned} \quad (20.4)$$

That is, the maximum-likelihood value of the mean is the sample average and the maximum-likelihood value of the standard deviation is the square root of the sample variance. Again, these are comforting results that confirm “commonsense” practice.



Now consider a linear Gaussian model with one continuous parent X and a continuous child Y . As explained on page 520, Y has a Gaussian distribution whose mean depends linearly on the value of X and whose standard deviation is fixed. To learn the conditional distribution $P(Y | X)$, we can maximize the conditional likelihood

$$P(y | x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y - (\theta_1 x + \theta_2))^2}{2\sigma^2}}. \quad (20.5)$$

Here, the parameters are θ_1 , θ_2 , and σ . The data are a collection of (x_j, y_j) pairs, as illustrated in Figure 20.4. Using the usual methods (Exercise 20.5), we can find the maximum-likelihood values of the parameters. The point here is different. If we consider just the parameters θ_1 and θ_2 that define the linear relationship between x and y , it becomes clear that maximizing the log likelihood with respect to these parameters is the same as *minimizing* the numerator $(y - (\theta_1 x + \theta_2))^2$ in the exponent of Equation (20.5). This is the L_2 loss, the squared error between the actual value y and the prediction $\theta_1 x + \theta_2$. This is the quantity minimized by the standard **linear regression** procedure described in Section 18.6. Now we can understand why: minimizing the sum of squared errors gives the maximum-likelihood straight-line model, *provided that the data are generated with Gaussian noise of fixed variance*.

20.2.4 Bayesian parameter learning

Maximum-likelihood learning gives rise to some very simple procedures, but it has some serious deficiencies with small data sets. For example, after seeing one cherry candy, the maximum-likelihood hypothesis is that the bag is 100% cherry (i.e., $\theta = 1.0$). Unless one's hypothesis prior is that bags must be either all cherry or all lime, this is not a reasonable conclusion. It is more likely that the bag is a mixture of lime and cherry. The Bayesian approach to parameter learning starts by defining a prior probability distribution over the possible hypotheses. We call this the **hypothesis prior**. Then, as data arrives, the posterior probability distribution is updated.

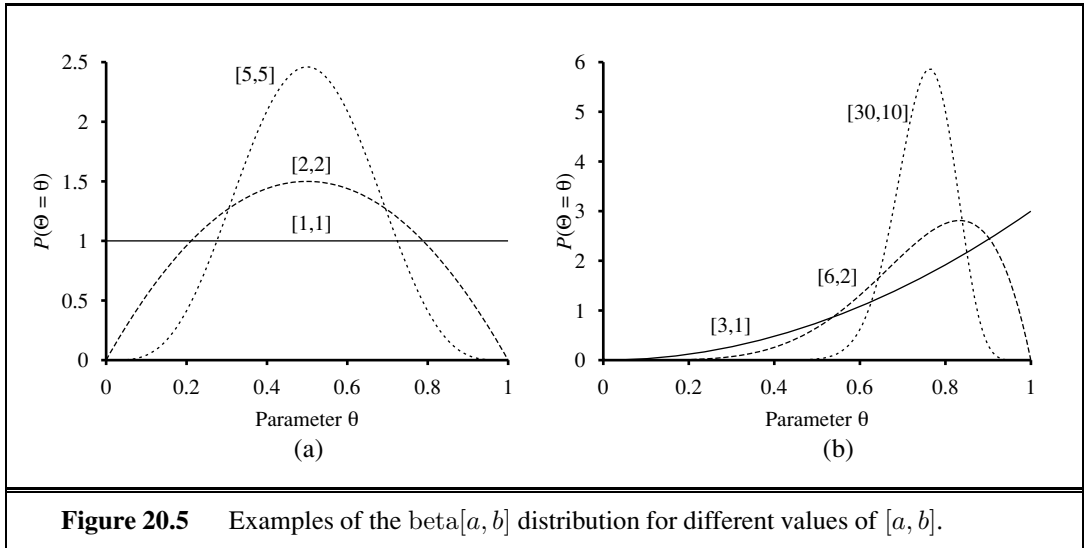


Figure 20.5 Examples of the $\text{beta}[a, b]$ distribution for different values of $[a, b]$.

The candy example in Figure 20.2(a) has one parameter, θ : the probability that a randomly selected piece of candy is cherry-flavored. In the Bayesian view, θ is the (unknown) value of a random variable Θ that defines the hypothesis space; the hypothesis prior is just the prior distribution $\mathbf{P}(\Theta)$. Thus, $P(\Theta = \theta)$ is the prior probability that the bag has a fraction θ of cherry candies.

If the parameter θ can be any value between 0 and 1, then $\mathbf{P}(\Theta)$ must be a continuous distribution that is nonzero only between 0 and 1 and that integrates to 1. The uniform density $P(\theta) = \text{Uniform}[0, 1](\theta)$ is one candidate. (See Chapter 13.) It turns out that the uniform density is a member of the family of **beta distributions**. Each beta distribution is defined by two **hyperparameters**³ a and b such that

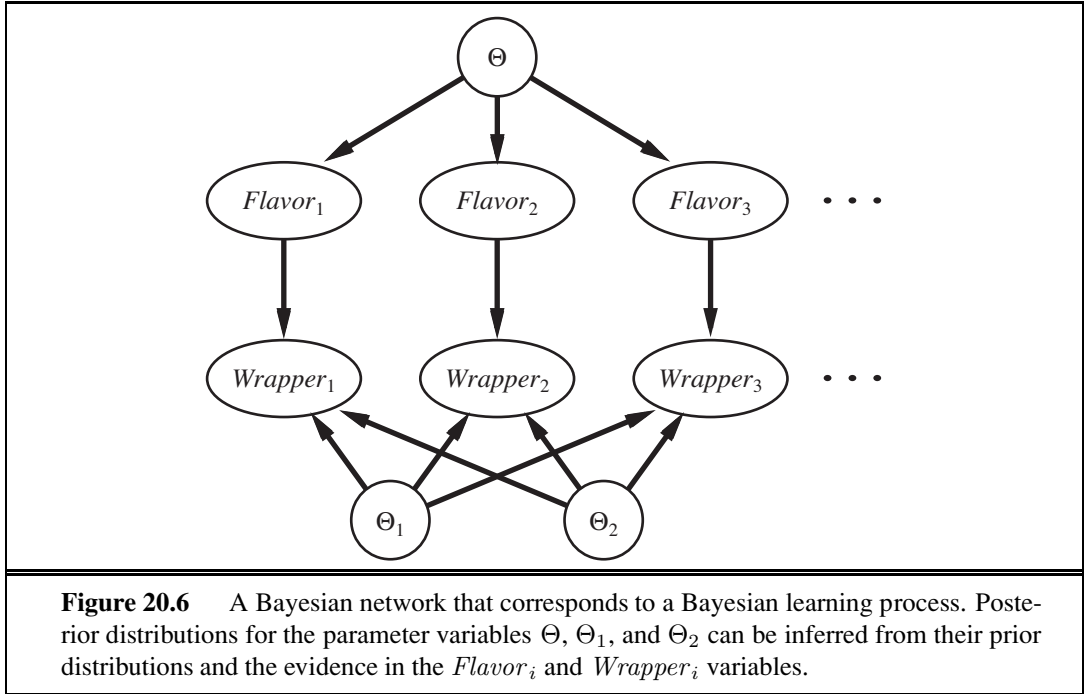
$$\text{beta}[a, b](\theta) = \alpha \theta^{a-1} (1 - \theta)^{b-1}, \quad (20.6)$$

for θ in the range $[0, 1]$. The normalization constant α , which makes the distribution integrate to 1, depends on a and b . (See Exercise 20.7.) Figure 20.5 shows what the distribution looks like for various values of a and b . The mean value of the distribution is $a/(a + b)$, so larger values of a suggest a belief that Θ is closer to 1 than to 0. Larger values of $a + b$ make the distribution more peaked, suggesting greater certainty about the value of Θ . Thus, the beta family provides a useful range of possibilities for the hypothesis prior.

Besides its flexibility, the beta family has another wonderful property: if Θ has a prior $\text{beta}[a, b]$, then, after a data point is observed, the posterior distribution for Θ is also a beta distribution. In other words, beta is closed under update. The beta family is called the **conjugate prior** for the family of distributions for a Boolean variable.⁴ Let's see how this works. Suppose we observe a cherry candy; then we have

³ They are called hyperparameters because they parameterize a distribution over θ , which is itself a parameter.

⁴ Other conjugate priors include the **Dirichlet** family for the parameters of a discrete multivalued distribution and the **Normal-Wishart** family for the parameters of a Gaussian distribution. See Bernardo and Smith (1994).



$$\begin{aligned}
 P(\theta \mid D_1 = \text{cherry}) &= \alpha P(D_1 = \text{cherry} \mid \theta) P(\theta) \\
 &= \alpha' \theta \cdot \text{beta}[a, b](\theta) = \alpha' \theta \cdot \theta^{a-1} (1 - \theta)^{b-1} \\
 &= \alpha' \theta^a (1 - \theta)^{b-1} = \text{beta}[a + 1, b](\theta) .
 \end{aligned}$$

Thus, after seeing a cherry candy, we simply increment the a parameter to get the posterior; similarly, after seeing a lime candy, we increment the b parameter. Thus, we can view the a and b hyperparameters as **virtual counts**, in the sense that a prior $\text{beta}[a, b]$ behaves exactly as if we had started out with a uniform prior $\text{beta}[1, 1]$ and seen $a - 1$ actual cherry candies and $b - 1$ actual lime candies.

By examining a sequence of beta distributions for increasing values of a and b , keeping the proportions fixed, we can see vividly how the posterior distribution over the parameter Θ changes as data arrive. For example, suppose the actual bag of candy is 75% cherry. Figure 20.5(b) shows the sequence $\text{beta}[3, 1]$, $\text{beta}[6, 2]$, $\text{beta}[30, 10]$. Clearly, the distribution is converging to a narrow peak around the true value of Θ . For large data sets, then, Bayesian learning (at least in this case) converges to the same answer as maximum-likelihood learning.

Now let us consider a more complicated case. The network in Figure 20.2(b) has three parameters, θ , θ_1 , and θ_2 , where θ_1 is the probability of a red wrapper on a cherry candy and θ_2 is the probability of a red wrapper on a lime candy. The Bayesian hypothesis prior must cover all three parameters—that is, we need to specify $\mathbf{P}(\Theta, \Theta_1, \Theta_2)$. Usually, we assume **parameter independence**:

$$\mathbf{P}(\Theta, \Theta_1, \Theta_2) = \mathbf{P}(\Theta) \mathbf{P}(\Theta_1) \mathbf{P}(\Theta_2) .$$

VIRTUAL COUNTS

PARAMETER
INDEPENDENCE

With this assumption, each parameter can have its own beta distribution that is updated separately as data arrive. Figure 20.6 shows how we can incorporate the hypothesis prior and any data into one Bayesian network. The nodes Θ , Θ_1 , Θ_2 have no parents. But each time we make an observation of a wrapper and corresponding flavor of a piece of candy, we add a node $Flavor_i$, which is dependent on the flavor parameter Θ :

$$P(Flavor_i = \text{cherry} \mid \Theta = \theta) = \theta .$$

We also add a node $Wrapper_i$, which is dependent on Θ_1 and Θ_2 :

$$P(Wrapper_i = \text{red} \mid Flavor_i = \text{cherry}, \Theta_1 = \theta_1) = \theta_1$$

$$P(Wrapper_i = \text{red} \mid Flavor_i = \text{lime}, \Theta_2 = \theta_2) = \theta_2 .$$

Now, the entire Bayesian learning process can be formulated as an *inference* problem. We add new evidence nodes, then query the unknown nodes (in this case, Θ , Θ_1 , Θ_2). This formulation of learning and prediction makes it clear that Bayesian learning requires no extra “principles of learning.” Furthermore, *there is, in essence, just one learning algorithm*—the inference algorithm for Bayesian networks. Of course, the nature of these networks is somewhat different from those of Chapter 14 because of the potentially huge number of evidence variables representing the training set and the prevalence of continuous-valued parameter variables.



20.2.5 Learning Bayes net structures

So far, we have assumed that the structure of the Bayes net is given and we are just trying to learn the parameters. The structure of the network represents basic causal knowledge about the domain that is often easy for an expert, or even a naive user, to supply. In some cases, however, the causal model may be unavailable or subject to dispute—for example, certain corporations have long claimed that smoking does not cause cancer—so it is important to understand how the structure of a Bayes net can be learned from data. This section gives a brief sketch of the main ideas.

The most obvious approach is to *search* for a good model. We can start with a model containing no links and begin adding parents for each node, fitting the parameters with the methods we have just covered and measuring the accuracy of the resulting model. Alternatively, we can start with an initial guess at the structure and use hill-climbing or simulated annealing search to make modifications, retuning the parameters after each change in the structure. Modifications can include reversing, adding, or deleting links. We must not introduce cycles in the process, so many algorithms assume that an ordering is given for the variables, and that a node can have parents only among those nodes that come earlier in the ordering (just as in the construction process in Chapter 14). For full generality, we also need to search over possible orderings.

There are two alternative methods for deciding when a good structure has been found. The first is to test whether the conditional independence assertions implicit in the structure are actually satisfied in the data. For example, the use of a naive Bayes model for the restaurant problem assumes that

$$\mathbf{P}(Fri/Sat, Bar \mid WillWait) = \mathbf{P}(Fri/Sat \mid WillWait)\mathbf{P}(Bar \mid WillWait)$$

and we can check in the data that the same equation holds between the corresponding conditional frequencies. But even if the structure describes the true causal nature of the domain, statistical fluctuations in the data set mean that the equation will never be satisfied *exactly*, so we need to perform a suitable statistical test to see if there is sufficient evidence that the independence hypothesis is violated. The complexity of the resulting network will depend on the threshold used for this test—the stricter the independence test, the more links will be added and the greater the danger of overfitting.

An approach more consistent with the ideas in this chapter is to assess the degree to which the proposed model explains the data (in a probabilistic sense). We must be careful how we measure this, however. If we just try to find the maximum-likelihood hypothesis, we will end up with a fully connected network, because adding more parents to a node cannot decrease the likelihood (Exercise 20.8). We are forced to penalize model complexity in some way. The MAP (or MDL) approach simply subtracts a penalty from the likelihood of each structure (after parameter tuning) before comparing different structures. The Bayesian approach places a joint prior over structures and parameters. There are usually far too many structures to sum over (superexponential in the number of variables), so most practitioners use MCMC to sample over structures.

Penalizing complexity (whether by MAP or Bayesian methods) introduces an important connection between the optimal structure and the nature of the representation for the conditional distributions in the network. With tabular distributions, the complexity penalty for a node's distribution grows exponentially with the number of parents, but with, say, noisy-OR distributions, it grows only linearly. This means that learning with noisy-OR (or other compactly parameterized) models tends to produce learned structures with more parents than does learning with tabular distributions.

20.2.6 Density estimation with nonparametric models

It is possible to learn a probability model without making any assumptions about its structure and parameterization by adopting the nonparametric methods of Section 18.8. The task of **nonparametric density estimation** is typically done in continuous domains, such as that shown in Figure 20.7(a). The figure shows a probability density function on a space defined by two continuous variables. In Figure 20.7(b) we see a sample of data points from this density function. The question is, can we recover the model from the samples?

First we will consider **k -nearest-neighbors** models. (In Chapter 18 we saw nearest-neighbor models for classification and regression; here we see them for density estimation.) Given a sample of data points, to estimate the unknown probability density at a query point \mathbf{x} we can simply measure the density of the data points in the neighborhood of \mathbf{x} . Figure 20.7(b) shows two query points (small squares). For each query point we have drawn the smallest circle that encloses 10 neighbors—the 10-nearest-neighborhood. We can see that the central circle is large, meaning there is a low density there, and the circle on the right is small, meaning there is a high density there. In Figure 20.8 we show three plots of density estimation using k -nearest-neighbors, for different values of k . It seems clear that (b) is about right, while (a) is too spiky (k is too small) and (c) is too smooth (k is too big).

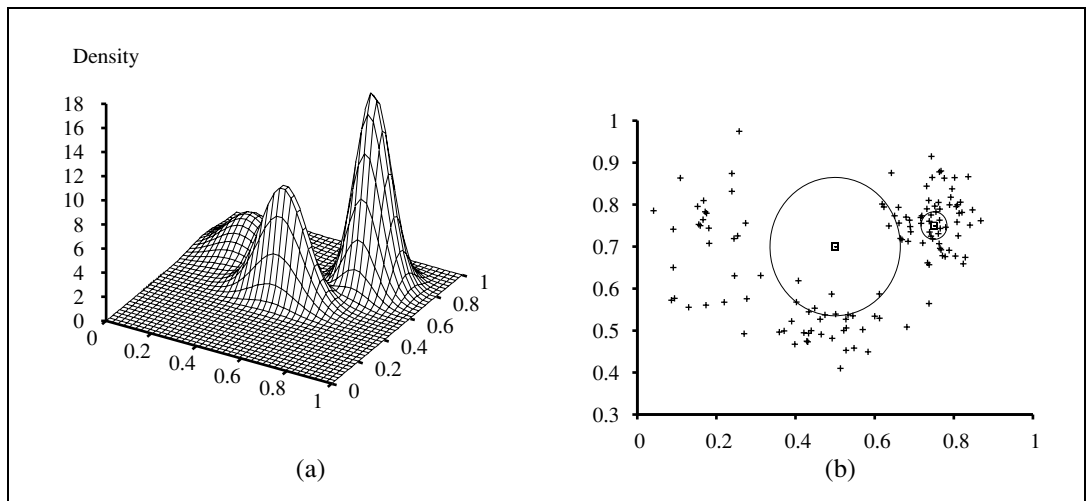


Figure 20.7 (a) A 3D plot of the mixture of Gaussians from Figure 20.11(a). (b) A 128-point sample of points from the mixture, together with two query points (small squares) and their 10-nearest-neighborhoods (medium and large circles).

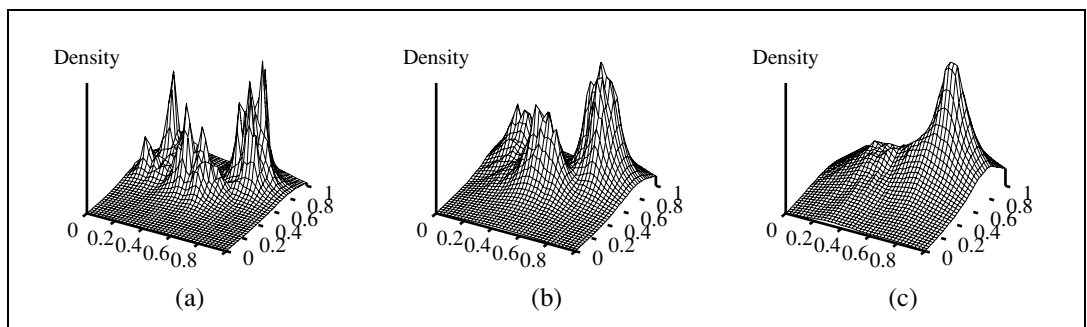


Figure 20.8 Density estimation using k -nearest-neighbors, applied to the data in Figure 20.7(b), for $k = 3, 10$, and 40 respectively. $k = 3$ is too spiky, 40 is too smooth, and 10 is just about right. The best value for k can be chosen by cross-validation.

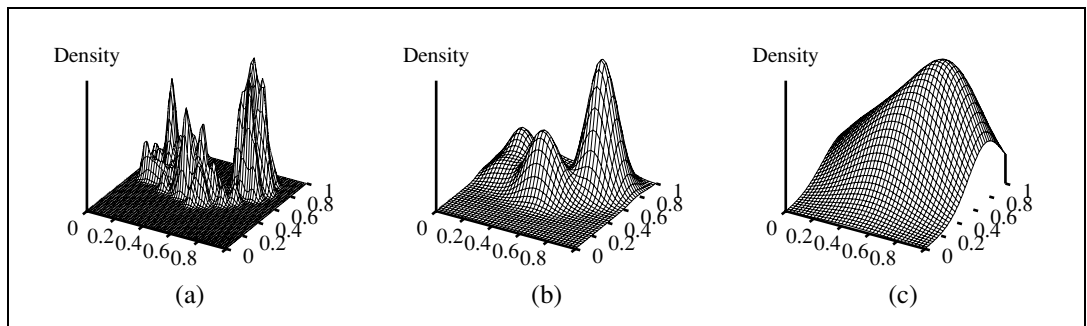


Figure 20.9 Kernel density estimation for the data in Figure 20.7(b), using Gaussian kernels with $w = 0.02, 0.07$, and 0.20 respectively. $w = 0.07$ is about right.

Another possibility is to use **kernel functions**, as we did for locally weighted regression. To apply a kernel model to density estimation, assume that each data point generates its own little density function, using a Gaussian kernel. The estimated density at a query point \mathbf{x} is then the average density as given by each kernel function:

$$P(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^N \mathcal{K}(\mathbf{x}, \mathbf{x}_j).$$

We will assume spherical Gaussians with standard deviation w along each axis:

$$\mathcal{K}(\mathbf{x}, \mathbf{x}_j) = \frac{1}{(w^2 \sqrt{2\pi})^d} e^{-\frac{D(\mathbf{x}, \mathbf{x}_j)^2}{2w^2}},$$

where d is the number of dimensions in \mathbf{x} and D is the Euclidean distance function. We still have the problem of choosing a suitable value for kernel width w ; Figure 20.9 shows values that are too small, just right, and too large. A good value of w can be chosen by using cross-validation.

20.3 LEARNING WITH HIDDEN VARIABLES: THE EM ALGORITHM

LATENT VARIABLE



The preceding section dealt with the fully observable case. Many real-world problems have **hidden variables** (sometimes called **latent variables**), which are not observable in the data that are available for learning. For example, medical records often include the observed symptoms, the physician's diagnosis, the treatment applied, and perhaps the outcome of the treatment, but they seldom contain a direct observation of the disease itself! (Note that the *diagnosis* is not the *disease*; it is a causal consequence of the observed symptoms, which are in turn caused by the disease.) One might ask, "If the disease is not observed, why not construct a model without it?" The answer appears in Figure 20.10, which shows a small, fictitious diagnostic model for heart disease. There are three observable predisposing factors and three observable symptoms (which are too depressing to name). Assume that each variable has three possible values (e.g., *none*, *moderate*, and *severe*). Removing the hidden variable from the network in (a) yields the network in (b); the total number of parameters increases from 78 to 708. Thus, *latent variables can dramatically reduce the number of parameters required to specify a Bayesian network*. This, in turn, can dramatically reduce the amount of data needed to learn the parameters.

EXPECTATION-
MAXIMIZATION

Hidden variables are important, but they do complicate the learning problem. In Figure 20.10(a), for example, it is not obvious how to learn the conditional distribution for *HeartDisease*, given its parents, because we do not know the value of *HeartDisease* in each case; the same problem arises in learning the distributions for the symptoms. This section describes an algorithm called **expectation-maximization**, or EM, that solves this problem in a very general way. We will show three examples and then provide a general description. The algorithm seems like magic at first, but once the intuition has been developed, one can find applications for EM in a huge range of learning problems.

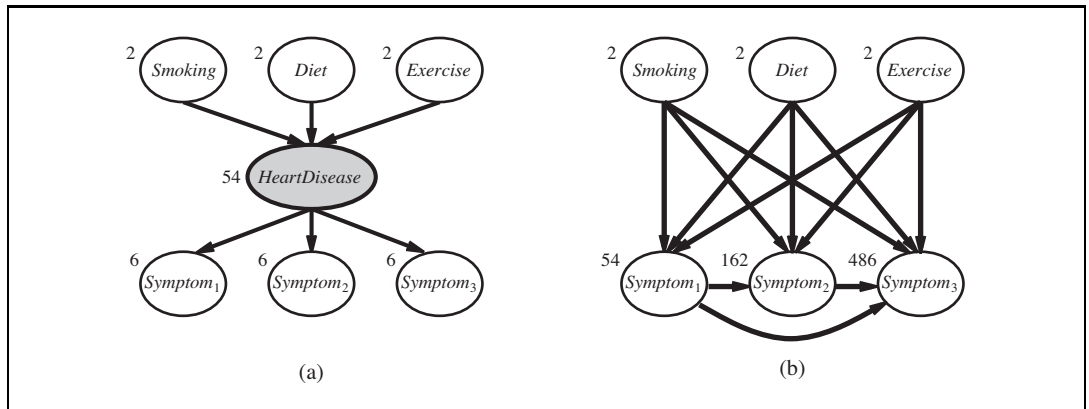


Figure 20.10 (a) A simple diagnostic network for heart disease, which is assumed to be a hidden variable. Each variable has three possible values and is labeled with the number of independent parameters in its conditional distribution; the total number is 78. (b) The equivalent network with *HeartDisease* removed. Note that the symptom variables are no longer conditionally independent given their parents. This network requires 708 parameters.

20.3.1 Unsupervised clustering: Learning mixtures of Gaussians

UNSUPERVISED
CLUSTERING

Unsupervised clustering is the problem of discerning multiple categories in a collection of objects. The problem is unsupervised because the category labels are not given. For example, suppose we record the spectra of a hundred thousand stars; are there different *types* of stars revealed by the spectra, and, if so, how many types and what are their characteristics? We are all familiar with terms such as “red giant” and “white dwarf,” but the stars do not carry these labels on their hats—astronomers had to perform unsupervised clustering to identify these categories. Other examples include the identification of species, genera, orders, and so on in the Linnæan taxonomy and the creation of natural kinds for ordinary objects (see Chapter 12).

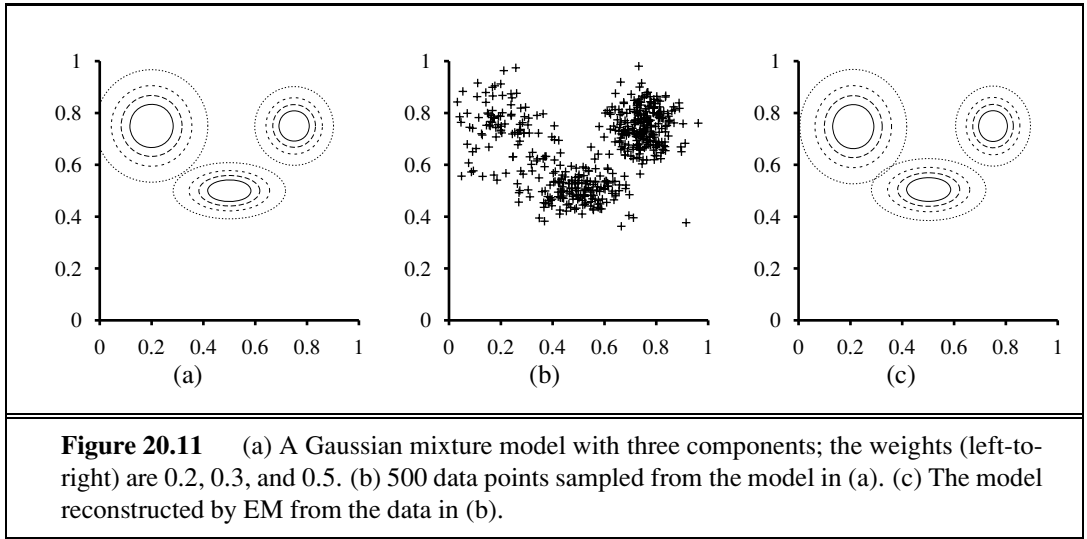
MIXTURE
DISTRIBUTION
COMPONENT

Unsupervised clustering begins with data. Figure 20.11(b) shows 500 data points, each of which specifies the values of two continuous attributes. The data points might correspond to stars, and the attributes might correspond to spectral intensities at two particular frequencies. Next, we need to understand what kind of probability distribution might have generated the data. Clustering presumes that the data are generated from a **mixture distribution**, P . Such a distribution has k **components**, each of which is a distribution in its own right. A data point is generated by first choosing a component and then generating a sample from that component. Let the random variable C denote the component, with values $1, \dots, k$; then the mixture distribution is given by

$$P(\mathbf{x}) = \sum_{i=1}^k P(C=i) P(\mathbf{x} | C=i),$$

where \mathbf{x} refers to the values of the attributes for a data point. For continuous data, a natural choice for the component distributions is the multivariate Gaussian, which gives the so-called **mixture of Gaussians** family of distributions. The parameters of a mixture of Gaussians are

MIXTURE OF
GAUSSIANS



$w_i = P(C = i)$ (the weight of each component), μ_i (the mean of each component), and Σ_i (the covariance of each component). Figure 20.11(a) shows a mixture of three Gaussians; this mixture is in fact the source of the data in (b) as well as being the model shown in Figure 20.7(a) on page 815.

The unsupervised clustering problem, then, is to recover a mixture model like the one in Figure 20.11(a) from raw data like that in Figure 20.11(b). Clearly, if we *knew* which component generated each data point, then it would be easy to recover the component Gaussians: we could just select all the data points from a given component and then apply (a multivariate version of) Equation (20.4) (page 809) for fitting the parameters of a Gaussian to a set of data. On the other hand, if we *knew* the parameters of each component, then we could, at least in a probabilistic sense, assign each data point to a component. The problem is that we know neither the assignments nor the parameters.

The basic idea of EM in this context is to *pretend* that we know the parameters of the model and then to infer the probability that each data point belongs to each component. After that, we refit the components to the data, where each component is fitted to the entire data set with each point weighted by the probability that it belongs to that component. The process iterates until convergence. Essentially, we are “completing” the data by inferring probability distributions over the hidden variables—which component each data point belongs to—based on the current model. For the mixture of Gaussians, we initialize the mixture-model parameters arbitrarily and then iterate the following two steps:

1. **E-step:** Compute the probabilities $p_{ij} = P(C = i | \mathbf{x}_j)$, the probability that datum \mathbf{x}_j was generated by component i . By Bayes’ rule, we have $p_{ij} = \alpha P(\mathbf{x}_j | C = i) P(C = i)$. The term $P(\mathbf{x}_j | C = i)$ is just the probability at \mathbf{x}_j of the i th Gaussian, and the term $P(C = i)$ is just the weight parameter for the i th Gaussian. Define $n_i = \sum_j p_{ij}$, the effective number of data points currently assigned to component i .
2. **M-step:** Compute the new mean, covariance, and component weights using the following steps in sequence:

$$\begin{aligned}\boldsymbol{\mu}_i &\leftarrow \sum_j p_{ij} \mathbf{x}_j / n_i \\ \boldsymbol{\Sigma}_i &\leftarrow \sum_j p_{ij} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^\top / n_i \\ w_i &\leftarrow n_i / N\end{aligned}$$

INDICATOR VARIABLE

where N is the total number of data points. The E-step, or *expectation* step, can be viewed as computing the expected values p_{ij} of the hidden **indicator variables** Z_{ij} , where Z_{ij} is 1 if datum \mathbf{x}_j was generated by the i th component and 0 otherwise. The M-step, or *maximization* step, finds the new values of the parameters that maximize the log likelihood of the data, given the expected values of the hidden indicator variables.

The final model that EM learns when it is applied to the data in Figure 20.11(a) is shown in Figure 20.11(c); it is virtually indistinguishable from the original model from which the data were generated. Figure 20.12(a) plots the log likelihood of the data according to the current model as EM progresses.



There are two points to notice. First, the log likelihood for the final learned model slightly *exceeds* that of the original model, from which the data were generated. This might seem surprising, but it simply reflects the fact that the data were generated randomly and might not provide an exact reflection of the underlying model. The second point is that *EM increases the log likelihood of the data at every iteration*. This fact can be proved in general. Furthermore, under certain conditions (that hold in most cases), EM can be proven to reach a local maximum in likelihood. (In rare cases, it could reach a saddle point or even a local minimum.) In this sense, EM resembles a gradient-based hill-climbing algorithm, but notice that it has no “step size” parameter.

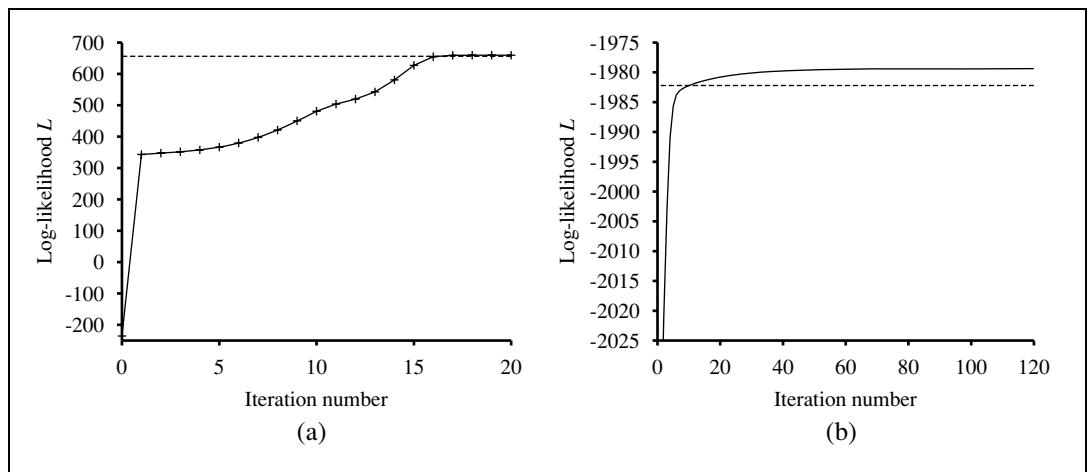
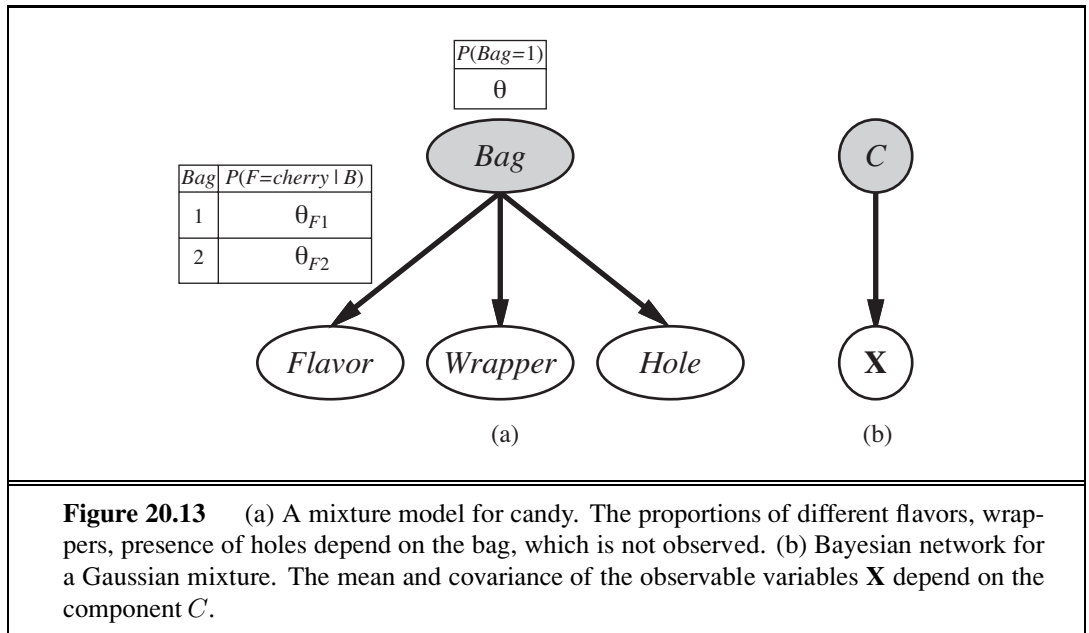


Figure 20.12 Graphs showing the log likelihood of the data, L , as a function of the EM iteration. The horizontal line shows the log likelihood according to the true model. (a) Graph for the Gaussian mixture model in Figure 20.11. (b) Graph for the Bayesian network in Figure 20.13(a).



Things do not always go as well as Figure 20.12(a) might suggest. It can happen, for example, that one Gaussian component shrinks so that it covers just a single data point. Then its variance will go to zero and its likelihood will go to infinity! Another problem is that two components can “merge,” acquiring identical means and variances and sharing their data points. These kinds of degenerate local maxima are serious problems, especially in high dimensions. One solution is to place priors on the model parameters and to apply the MAP version of EM. Another is to restart a component with new random parameters if it gets too small or too close to another component. Sensible initialization also helps.

20.3.2 Learning Bayesian networks with hidden variables

To learn a Bayesian network with hidden variables, we apply the same insights that worked for mixtures of Gaussians. Figure 20.13 represents a situation in which there are two bags of candies that have been mixed together. Candies are described by three features: in addition to the *Flavor* and the *Wrapper*, some candies have a *Hole* in the middle and some do not. The distribution of candies in each bag is described by a **naïve Bayes** model: the features are independent, given the bag, but the conditional probability distribution for each feature depends on the bag. The parameters are as follows: θ is the prior probability that a candy comes from Bag 1; θ_{F1} and θ_{F2} are the probabilities that the flavor is cherry, given that the candy comes from Bag 1 or Bag 2 respectively; θ_{W1} and θ_{W2} give the probabilities that the wrapper is red; and θ_{H1} and θ_{H2} give the probabilities that the candy has a hole. Notice that the overall model is a mixture model. (In fact, we can also model the mixture of Gaussians as a Bayesian network, as shown in Figure 20.13(b).) In the figure, the bag is a hidden variable because, once the candies have been mixed together, we no longer know which bag each candy came from. In such a case, can we recover the descriptions of the two bags by

observing candies from the mixture? Let us work through an iteration of EM for this problem. First, let's look at the data. We generated 1000 samples from a model whose true parameters are as follows:

$$\theta = 0.5, \quad \theta_{F1} = \theta_{W1} = \theta_{H1} = 0.8, \quad \theta_{F2} = \theta_{W2} = \theta_{H2} = 0.3. \quad (20.7)$$

That is, the candies are equally likely to come from either bag; the first is mostly cherries with red wrappers and holes; the second is mostly limes with green wrappers and no holes. The counts for the eight possible kinds of candy are as follows:

| | <i>W = red</i> | | <i>W = green</i> | |
|-------------------|----------------|--------------|------------------|--------------|
| | <i>H = 1</i> | <i>H = 0</i> | <i>H = 1</i> | <i>H = 0</i> |
| <i>F = cherry</i> | 273 | 93 | 104 | 90 |
| <i>F = lime</i> | 79 | 100 | 94 | 167 |

We start by initializing the parameters. For numerical simplicity, we arbitrarily choose⁵

$$\theta^{(0)} = 0.6, \quad \theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6, \quad \theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4. \quad (20.8)$$

First, let us work on the θ parameter. In the fully observable case, we would estimate this directly from the *observed* counts of candies from bags 1 and 2. Because the bag is a hidden variable, we calculate the *expected* counts instead. The expected count $\hat{N}(\text{Bag} = 1)$ is the sum, over all candies, of the probability that the candy came from bag 1:

$$\theta^{(1)} = \hat{N}(\text{Bag} = 1)/N = \sum_{j=1}^N P(\text{Bag} = 1 \mid \text{flavor}_j, \text{wrapper}_j, \text{holes}_j)/N.$$

These probabilities can be computed by any inference algorithm for Bayesian networks. For a naive Bayes model such as the one in our example, we can do the inference “by hand,” using Bayes’ rule and applying conditional independence:

$$\theta^{(1)} = \frac{1}{N} \sum_{j=1}^N \frac{P(\text{flavor}_j \mid \text{Bag} = 1)P(\text{wrapper}_j \mid \text{Bag} = 1)P(\text{holes}_j \mid \text{Bag} = 1)P(\text{Bag} = 1)}{\sum_i P(\text{flavor}_j \mid \text{Bag} = i)P(\text{wrapper}_j \mid \text{Bag} = i)P(\text{holes}_j \mid \text{Bag} = i)P(\text{Bag} = i)}.$$

Applying this formula to, say, the 273 red-wrapped cherry candies with holes, we get a contribution of

$$\frac{273}{1000} \cdot \frac{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)}}{\theta_{F1}^{(0)}\theta_{W1}^{(0)}\theta_{H1}^{(0)}\theta^{(0)} + \theta_{F2}^{(0)}\theta_{W2}^{(0)}\theta_{H2}^{(0)}(1 - \theta^{(0)})} \approx 0.22797.$$

Continuing with the other seven kinds of candy in the table of counts, we obtain $\theta^{(1)} = 0.6124$.

Now let us consider the other parameters, such as θ_{F1} . In the fully observable case, we would estimate this directly from the *observed* counts of cherry and lime candies from bag 1. The *expected* count of cherry candies from bag 1 is given by

$$\sum_{j: \text{Flavor}_j = \text{cherry}} P(\text{Bag} = 1 \mid \text{Flavor}_j = \text{cherry}, \text{wrapper}_j, \text{holes}_j).$$

⁵ It is better in practice to choose them randomly, to avoid local maxima due to symmetry.

Again, these probabilities can be calculated by any Bayes net algorithm. Completing this process, we obtain the new values of all the parameters:

$$\begin{aligned} \theta_{F1}^{(1)} &= 0.6124, \theta_{F1}^{(1)} = 0.6684, \theta_{W1}^{(1)} = 0.6483, \theta_{H1}^{(1)} = 0.6558, \\ \theta_{F2}^{(1)} &= 0.3887, \theta_{W2}^{(1)} = 0.3817, \theta_{H2}^{(1)} = 0.3827. \end{aligned} \quad (20.9)$$

The log likelihood of the data increases from about -2044 initially to about -2021 after the first iteration, as shown in Figure 20.12(b). That is, the update improves the likelihood itself by a factor of about $e^{23} \approx 10^{10}$. By the tenth iteration, the learned model is a better fit than the original model ($L = -1982.214$). Thereafter, progress becomes very slow. This is not uncommon with EM, and many practical systems combine EM with a gradient-based algorithm such as Newton–Raphson (see Chapter 4) for the last phase of learning.



The general lesson from this example is that *the parameter updates for Bayesian network learning with hidden variables are directly available from the results of inference on each example. Moreover, only local posterior probabilities are needed for each parameter.* Here, “local” means that the CPT for each variable X_i can be learned from posterior probabilities involving just X_i and its parents \mathbf{U}_i . Defining θ_{ijk} to be the CPT parameter $P(X_i = x_{ij} \mid \mathbf{U}_i = \mathbf{u}_{ik})$, the update is given by the normalized expected counts as follows:

$$\theta_{ijk} \leftarrow \hat{N}(X_i = x_{ij}, \mathbf{U}_i = \mathbf{u}_{ik}) / \hat{N}(\mathbf{U}_i = \mathbf{u}_{ik}).$$

The expected counts are obtained by summing over the examples, computing the probabilities $P(X_i = x_{ij}, \mathbf{U}_i = \mathbf{u}_{ik})$ for each by using any Bayes net inference algorithm. For the exact algorithms—including variable elimination—all these probabilities are obtainable directly as a by-product of standard inference, with no need for extra computations specific to learning. Moreover, the information needed for learning is available *locally* for each parameter.

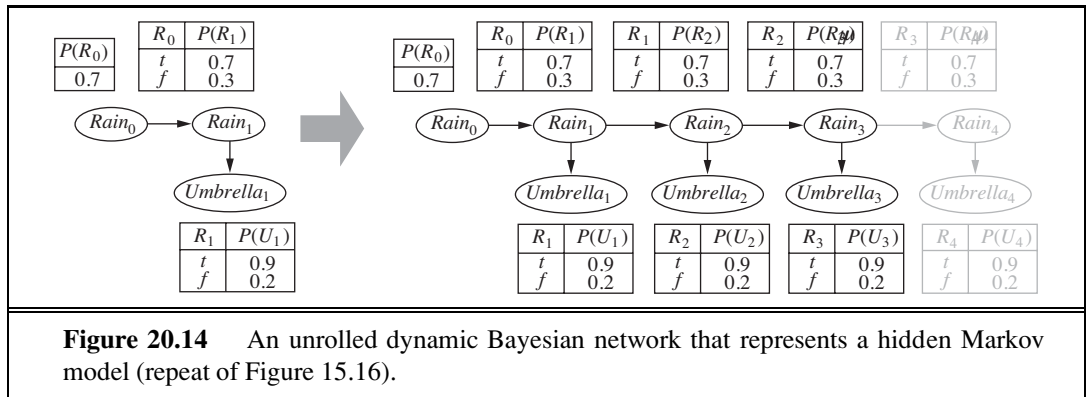
20.3.3 Learning hidden Markov models

Our final application of EM involves learning the transition probabilities in hidden Markov models (HMMs). Recall from Section 15.3 that a hidden Markov model can be represented by a dynamic Bayes net with a single discrete state variable, as illustrated in Figure 20.14. Each data point consists of an observation *sequence* of finite length, so the problem is to learn the transition probabilities from a set of observation sequences (or from just one long sequence).

We have already worked out how to learn Bayes nets, but there is one complication: in Bayes nets, each parameter is distinct; in a hidden Markov model, on the other hand, the individual transition probabilities from state i to state j at time t , $\theta_{ijt} = P(X_{t+1} = j \mid X_t = i)$, are *repeated* across time—that is, $\theta_{ijt} = \theta_{ij}$ for all t . To estimate the transition probability from state i to state j , we simply calculate the expected proportion of times that the system undergoes a transition to state j when in state i :

$$\theta_{ij} \leftarrow \sum_t \hat{N}(X_{t+1} = j, X_t = i) / \sum_t \hat{N}(X_t = i).$$

The expected counts are computed by an HMM inference algorithm. The **forward–backward** algorithm shown in Figure 15.4 can be modified very easily to compute the necessary probabilities. One important point is that the probabilities required are obtained by **smoothing**



rather than **filtering**; that is, we need to pay attention to subsequent evidence in estimating the probability that a particular transition occurred. The evidence in a murder case is usually obtained *after* the crime (i.e., the transition from state i to state j) has taken place.

20.3.4 The general form of the EM algorithm

We have seen several instances of the EM algorithm. Each involves computing expected values of hidden variables for each example and then recomputing the parameters, using the expected values as if they were observed values. Let \mathbf{x} be all the observed values in all the examples, let \mathbf{Z} denote all the hidden variables for all the examples, and let θ be all the parameters for the probability model. Then the EM algorithm is

$$\theta^{(i+1)} = \operatorname{argmax}_{\theta} \sum_{\mathbf{z}} P(\mathbf{Z} = \mathbf{z} | \mathbf{x}, \theta^{(i)}) L(\mathbf{x}, \mathbf{Z} = \mathbf{z} | \theta) .$$

This equation is the EM algorithm in a nutshell. The E-step is the computation of the summation, which is the expectation of the log likelihood of the “completed” data with respect to the distribution $P(\mathbf{Z} = \mathbf{z} | \mathbf{x}, \theta^{(i)})$, which is the posterior over the hidden variables, given the data. The M-step is the maximization of this expected log likelihood with respect to the parameters. For mixtures of Gaussians, the hidden variables are the Z_{ij} s, where Z_{ij} is 1 if example j was generated by component i . For Bayes nets, Z_{ij} is the value of unobserved variable X_i in example j . For HMMs, Z_{jt} is the state of the sequence in example j at time t . Starting from the general form, it is possible to derive an EM algorithm for a specific application once the appropriate hidden variables have been identified.

As soon as we understand the general idea of EM, it becomes easy to derive all sorts of variants and improvements. For example, in many cases the E-step—the computation of posteriors over the hidden variables—is intractable, as in large Bayes nets. It turns out that one can use an *approximate* E-step and still obtain an effective learning algorithm. With a sampling algorithm such as MCMC (see Section 14.5), the learning process is very intuitive: each state (configuration of hidden and observed variables) visited by MCMC is treated exactly as if it were a complete observation. Thus, the parameters can be updated directly after each MCMC transition. Other forms of approximate inference, such as variational and loopy methods, have also proved effective for learning very large networks.

20.3.5 Learning Bayes net structures with hidden variables

In Section 20.2.5, we discussed the problem of learning Bayes net structures with complete data. When unobserved variables may be influencing the data that are observed, things get more difficult. In the simplest case, a human expert might tell the learning algorithm that certain hidden variables exist, leaving it to the algorithm to find a place for them in the network structure. For example, an algorithm might try to learn the structure shown in Figure 20.10(a) on page 817, given the information that *HeartDisease* (a three-valued variable) should be included in the model. As in the complete-data case, the overall algorithm has an outer loop that searches over structures and an inner loop that fits the network parameters given the structure.

If the learning algorithm is not told which hidden variables exist, then there are two choices: either pretend that the data is really complete—which may force the algorithm to learn a parameter-intensive model such as the one in Figure 20.10(b)—or *invent* new hidden variables in order to simplify the model. The latter approach can be implemented by including new modification choices in the structure search: in addition to modifying links, the algorithm can add or delete a hidden variable or change its arity. Of course, the algorithm will not know that the new variable it has invented is called *HeartDisease*; nor will it have meaningful names for the values. Fortunately, newly invented hidden variables will usually be connected to preexisting variables, so a human expert can often inspect the local conditional distributions involving the new variable and ascertain its meaning.

As in the complete-data case, pure maximum-likelihood structure learning will result in a completely connected network (moreover, one with no hidden variables), so some form of complexity penalty is required. We can also apply MCMC to sample many possible network structures, thereby approximating Bayesian learning. For example, we can learn mixtures of Gaussians with an unknown number of components by sampling over the number; the approximate posterior distribution for the number of Gaussians is given by the sampling frequencies of the MCMC process.

For the complete-data case, the inner loop to learn the parameters is very fast—just a matter of extracting conditional frequencies from the data set. When there are hidden variables, the inner loop may involve many iterations of EM or a gradient-based algorithm, and each iteration involves the calculation of posteriors in a Bayes net, which is itself an NP-hard problem. To date, this approach has proved impractical for learning complex models. One possible improvement is the so-called **structural EM** algorithm, which operates in much the same way as ordinary (parametric) EM except that the algorithm can update the structure as well as the parameters. Just as ordinary EM uses the current parameters to compute the expected counts in the E-step and then applies those counts in the M-step to choose new parameters, structural EM uses the current structure to compute expected counts and then applies those counts in the M-step to evaluate the likelihood for potential new structures. (This contrasts with the outer-loop/inner-loop method, which computes new expected counts for each potential structure.) In this way, structural EM may make several structural alterations to the network without once recomputing the expected counts, and is capable of learning non-trivial Bayes net structures. Nonetheless, much work remains to be done before we can say that the structure-learning problem is solved.

20.4 SUMMARY

Statistical learning methods range from simple calculation of averages to the construction of complex models such as Bayesian networks. They have applications throughout computer science, engineering, computational biology, neuroscience, psychology, and physics. This chapter has presented some of the basic ideas and given a flavor of the mathematical underpinnings. The main points are as follows:

- **Bayesian learning** methods formulate learning as a form of probabilistic inference, using the observations to update a prior distribution over hypotheses. This approach provides a good way to implement Ockham's razor, but quickly becomes intractable for complex hypothesis spaces.
- **Maximum a posteriori** (MAP) learning selects a single most likely hypothesis given the data. The hypothesis prior is still used and the method is often more tractable than full Bayesian learning.
- **Maximum-likelihood** learning simply selects the hypothesis that maximizes the likelihood of the data; it is equivalent to MAP learning with a uniform prior. In simple cases such as linear regression and fully observable Bayesian networks, maximum-likelihood solutions can be found easily in closed form. **Naive Bayes** learning is a particularly effective technique that scales well.
- When some variables are hidden, local maximum likelihood solutions can be found using the EM algorithm. Applications include clustering using mixtures of Gaussians, learning Bayesian networks, and learning hidden Markov models.
- Learning the structure of Bayesian networks is an example of **model selection**. This usually involves a discrete search in the space of structures. Some method is required for trading off model complexity against degree of fit.
- **Nonparametric models** represent a distribution using the collection of data points. Thus, the number of parameters grows with the training set. Nearest-neighbors methods look at the examples nearest to the point in question, whereas **kernel** methods form a distance-weighted combination of all the examples.

Statistical learning continues to be a very active area of research. Enormous strides have been made in both theory and practice, to the point where it is possible to learn almost any model for which exact or approximate inference is feasible.

BIBLIOGRAPHICAL AND HISTORICAL NOTES

The application of statistical learning techniques in AI was an active area of research in the early years (see Duda and Hart, 1973) but became separated from mainstream AI as the latter field concentrated on symbolic methods. A resurgence of interest occurred shortly after the introduction of Bayesian network models in the late 1980s; at roughly the same time,

a statistical view of neural network learning began to emerge. In the late 1990s, there was a noticeable convergence of interests in machine learning, statistics, and neural networks, centered on methods for creating large probabilistic models from data.

The naive Bayes model is one of the oldest and simplest forms of Bayesian network, dating back to the 1950s. Its origins were mentioned in Chapter 13. Its surprising success is partially explained by Domingos and Pazzani (1997). A boosted form of naive Bayes learning won the first KDD Cup data mining competition (Elkan, 1997). Heckerman (1998) gives an excellent introduction to the general problem of Bayes net learning. Bayesian parameter learning with Dirichlet priors for Bayesian networks was discussed by Spiegelhalter *et al.* (1993). The BUGS software package (Gilks *et al.*, 1994) incorporates many of these ideas and provides a very powerful tool for formulating and learning complex probability models. The first algorithms for learning Bayes net structures used conditional independence tests (Pearl, 1988; Pearl and Verma, 1991). Spirtes *et al.* (1993) developed a comprehensive approach embodied in the TETRAD package for Bayes net learning. Algorithmic improvements since then led to a clear victory in the 2001 KDD Cup data mining competition for a Bayes net learning method (Cheng *et al.*, 2002). (The specific task here was a bioinformatics problem with 139,351 features!) A structure-learning approach based on maximizing likelihood was developed by Cooper and Herskovits (1992) and improved by Heckerman *et al.* (1994). Several algorithmic advances since that time have led to quite respectable performance in the complete-data case (Moore and Wong, 2003; Teyssier and Koller, 2005). One important component is an efficient data structure, the AD-tree, for caching counts over all possible combinations of variables and values (Moore and Lee, 1997). Friedman and Goldszmidt (1996) pointed out the influence of the representation of local conditional distributions on the learned structure.

The general problem of learning probability models with hidden variables and missing data was addressed by Hartley (1958), who described the general idea of what was later called EM and gave several examples. Further impetus came from the Baum–Welch algorithm for HMM learning (Baum and Petrie, 1966), which is a special case of EM. The paper by Dempster, Laird, and Rubin (1977), which presented the EM algorithm in general form and analyzed its convergence, is one of the most cited papers in both computer science and statistics. (Dempster himself views EM as a schema rather than an algorithm, since a good deal of mathematical work may be required before it can be applied to a new family of distributions.) McLachlan and Krishnan (1997) devote an entire book to the algorithm and its properties. The specific problem of learning mixture models, including mixtures of Gaussians, is covered by Titterton *et al.* (1985). Within AI, the first successful system that used EM for mixture modeling was AUTOCLASS (Cheeseman *et al.*, 1988; Cheeseman and Stutz, 1996). AUTOCLASS has been applied to a number of real-world scientific classification tasks, including the discovery of new types of stars from spectral data (Goebel *et al.*, 1989) and new classes of proteins and introns in DNA/protein sequence databases (Hunter and States, 1992).

For maximum-likelihood parameter learning in Bayes nets with hidden variables, EM and gradient-based methods were introduced around the same time by Lauritzen (1995), Russell *et al.* (1995), and Binder *et al.* (1997a). The structural EM algorithm was developed by Friedman (1998) and applied to maximum-likelihood learning of Bayes net structures with

latent variables. Friedman and Koller (2003). describe Bayesian structure learning.

The ability to learn the structure of Bayesian networks is closely connected to the issue of recovering *causal* information from data. That is, is it possible to learn Bayes nets in such a way that the recovered network structure indicates real causal influences? For many years, statisticians avoided this question, believing that observational data (as opposed to data generated from experimental trials) could yield only correlational information—after all, any two variables that appear related might in fact be influenced by a third, unknown causal factor rather than influencing each other directly. Pearl (2000) has presented convincing arguments to the contrary, showing that there are in fact many cases where causality can be ascertained and developing the **causal network** formalism to express causes and the effects of intervention as well as ordinary conditional probabilities.

CAUSAL NETWORK

Nonparametric density estimation, also called **Parzen window** density estimation, was investigated initially by Rosenblatt (1956) and Parzen (1962). Since that time, a huge literature has developed investigating the properties of various estimators. Devroye (1987) gives a thorough introduction. There is also a rapidly growing literature on nonparametric Bayesian methods, originating with the seminal work of Ferguson (1973) on the **Dirichlet process**, which can be thought of as a distribution over Dirichlet distributions. These methods are particularly useful for mixtures with unknown numbers of components. Ghahramani (2005) and Jordan (2005) provide useful tutorials on the many applications of these ideas to statistical learning. The text by Rasmussen and Williams (2006) covers the **Gaussian process**, which gives a way of defining prior distributions over the space of continuous functions.

DIRICHLET PROCESS

GAUSSIAN PROCESS

The material in this chapter brings together work from the fields of statistics and pattern recognition, so the story has been told many times in many ways. Good texts on Bayesian statistics include those by DeGroot (1970), Berger (1985), and Gelman *et al.* (1995). Bishop (2007) and Hastie *et al.* (2009) provide an excellent introduction to statistical machine learning. For pattern classification, the classic text for many years has been Duda and Hart (1973), now updated (Duda *et al.*, 2001). The annual NIPS (Neural Information Processing Conference) conference, whose proceedings are published as the series *Advances in Neural Information Processing Systems*, is now dominated by Bayesian papers. Papers on learning Bayesian networks also appear in the *Uncertainty in AI* and *Machine Learning* conferences and in several statistics conferences. Journals specific to neural networks include *Neural Computation*, *Neural Networks*, and the *IEEE Transactions on Neural Networks*. Specifically Bayesian venues include the Valencia International Meetings on Bayesian Statistics and the journal *Bayesian Analysis*.

EXERCISES

20.1 The data used for Figure 20.1 on page 804 can be viewed as being generated by h_5 . For each of the other four hypotheses, generate a data set of length 100 and plot the corresponding graphs for $P(h_i | d_1, \dots, d_N)$ and $P(D_{N+1} = \text{lime} | d_1, \dots, d_N)$. Comment on your results.

20.2 Suppose that Ann’s utilities for cherry and lime candies are c_A and ℓ_A , whereas Bob’s utilities are c_B and ℓ_B . (But once Ann has unwrapped a piece of candy, Bob won’t buy it.) Presumably, if Bob likes lime candies much more than Ann, it would be wise for Ann to sell her bag of candies once she is sufficiently sure of its lime content. On the other hand, if Ann unwraps too many candies in the process, the bag will be worth less. Discuss the problem of determining the optimal point at which to sell the bag. Determine the expected utility of the optimal procedure, given the prior distribution from Section 20.1.

20.3 Two statisticians go to the doctor and are both given the same prognosis: A 40% chance that the problem is the deadly disease A , and a 60% chance of the fatal disease B . Fortunately, there are anti- A and anti- B drugs that are inexpensive, 100% effective, and free of side-effects. The statisticians have the choice of taking one drug, both, or neither. What will the first statistician (an avid Bayesian) do? How about the second statistician, who always uses the maximum likelihood hypothesis?

The doctor does some research and discovers that disease B actually comes in two versions, dextro- B and levo- B , which are equally likely and equally treatable by the anti- B drug. Now that there are three hypotheses, what will the two statisticians do?

20.4 Explain how to apply the boosting method of Chapter 18 to naive Bayes learning. Test the performance of the resulting algorithm on the restaurant learning problem.

20.5 Consider N data points (x_j, y_j) , where the y_j s are generated from the x_j s according to the linear Gaussian model in Equation (20.5). Find the values of θ_1 , θ_2 , and σ that maximize the conditional log likelihood of the data.

20.6 Consider the noisy-OR model for fever described in Section 14.3. Explain how to apply maximum-likelihood learning to fit the parameters of such a model to a set of complete data. (*Hint*: use the chain rule for partial derivatives.)

20.7 This exercise investigates properties of the Beta distribution defined in Equation (20.6).

- a. By integrating over the range $[0, 1]$, show that the normalization constant for the distribution $\text{beta}[a, b]$ is given by $\alpha = \Gamma(a + b) / \Gamma(a)\Gamma(b)$ where $\Gamma(x)$ is the **Gamma function**, defined by $\Gamma(x + 1) = x \cdot \Gamma(x)$ and $\Gamma(1) = 1$. (For integer x , $\Gamma(x + 1) = x!$.)
- b. Show that the mean is $a / (a + b)$.
- c. Find the mode(s) (the most likely value(s) of θ).
- d. Describe the distribution $\text{beta}[\epsilon, \epsilon]$ for very small ϵ . What happens as such a distribution is updated?

20.8 Consider an arbitrary Bayesian network, a complete data set for that network, and the likelihood for the data set according to the network. Give a simple proof that the likelihood of the data cannot decrease if we add a new link to the network and recompute the maximum-likelihood parameter values.

20.9 Consider a single Boolean random variable Y (the “classification”). Let the prior probability $P(Y = \text{true})$ be π . Let’s try to find π , given a training set $D = (y_1, \dots, y_N)$ with N independent samples of Y . Furthermore, suppose p of the N are positive and n of the N are negative.

- a. Write down an expression for the likelihood of D (i.e., the probability of seeing this particular sequence of examples, given a fixed value of π) in terms of π , p , and n .
- b. By differentiating the log likelihood L , find the value of π that maximizes the likelihood.
- c. Now suppose we add in k Boolean random variables X_1, X_2, \dots, X_k (the “attributes”) that describe each sample, and suppose we assume that the attributes are conditionally independent of each other given the goal Y . Draw the Bayes net corresponding to this assumption.
- d. Write down the likelihood for the data including the attributes, using the following additional notation:
 - α_i is $P(X_i = \text{true} | Y = \text{true})$.
 - β_i is $P(X_i = \text{true} | Y = \text{false})$.
 - p_i^+ is the count of samples for which $X_i = \text{true}$ and $Y = \text{true}$.
 - n_i^+ is the count of samples for which $X_i = \text{false}$ and $Y = \text{true}$.
 - p_i^- is the count of samples for which $X_i = \text{true}$ and $Y = \text{false}$.
 - n_i^- is the count of samples for which $X_i = \text{false}$ and $Y = \text{false}$.

[Hint: consider first the probability of seeing a single example with specified values for X_1, X_2, \dots, X_k and Y .]
- e. By differentiating the log likelihood L , find the values of α_i and β_i (in terms of the various counts) that maximize the likelihood and say in words what these values represent.
- f. Let $k = 2$, and consider a data set with 4 all four possible examples of the XOR function. Compute the maximum likelihood estimates of π , α_1 , α_2 , β_1 , and β_2 .
- g. Given these estimates of π , α_1 , α_2 , β_1 , and β_2 , what are the posterior probabilities $P(Y = \text{true} | x_1, x_2)$ for each example?

20.10 Consider the application of EM to learn the parameters for the network in Figure 20.13(a), given the true parameters in Equation (20.7).

- a. Explain why the EM algorithm would not work if there were just two attributes in the model rather than three.
- b. Show the calculations for the first iteration of EM starting from Equation (20.8).
- c. What happens if we start with all the parameters set to the same value p ? (Hint: you may find it helpful to investigate this empirically before deriving the general result.)
- d. Write out an expression for the log likelihood of the tabulated candy data on page 821 in terms of the parameters, calculate the partial derivatives with respect to each parameter, and investigate the nature of the fixed point reached in part (c).