BAYESIAN LEARNING

Bayesian reasoning provides a probabilistic approach to inference. It is based on the assumption that the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about these probabilities together with observed data. It is important to machine learning because it provides a quantitative approach to weighing the evidence supporting alternative hypotheses. Bayesian reasoning provides the basis for learning algorithms that directly manipulate probabilities, as well as a framework for analyzing the operation of other algorithms that do not explicitly manipulate probabilities.

6.1 INTRODUCTION

Bayesian learning methods are relevant to our study of machine learning for two different reasons. First, Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems. For example, Michie et al. (1994) provide a detailed study comparing the naive Bayes classifier to other learning algorithms, including decision tree and neural network algorithms. These researchers show that the naive Bayes classifier is competitive with these other learning algorithms in many cases and that in some cases it outperforms these other methods. In this chapter we describe the naive Bayes classifier and provide a detailed example of its use. In particular, we discuss its application to the problem of learning to classify text documents such as electronic news articles.

For such learning tasks, the naive Bayes classifier is among the most effective algorithms known.

The second reason that Bayesian methods are important to our study of machine learning is that they provide a useful perspective for understanding many learning algorithms that do not explicitly manipulate probabilities. For example, in this chapter we analyze algorithms such as the FIND-S and CANDIDATE-ELIMINATION algorithms of Chapter 2 to determine conditions under which they output the most probable hypothesis given the training data. We also use a Bayesian analysis to justify a key design choice in neural network learning algorithms: choosing to minimize the sum of squared errors when searching the space of possible neural networks. We also derive an alternative error function, cross entropy, that is more appropriate than sum of squared errors when learning target functions that predict probabilities. We use a Bayesian perspective to analyze the inductive bias of decision tree learning algorithms that favor short decision trees and examine the closely related Minimum Description Length principle. A basic familiarity with Bayesian methods is important to understanding and characterizing the operation of many algorithms in machine learning.

Features of Bayesian learning methods include:

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example.
- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian learning, prior knowledge is provided by asserting (1) a prior probability for each candidate hypothesis, and (2) a probability distribution over observed data for each possible hypothesis.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions (e.g., hypotheses such as "this pneumonia patient has a 93% chance of complete recovery").
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
- Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

One practical difficulty in applying Bayesian methods is that they typically require initial knowledge of many probabilities. When these probabilities are not known in advance they are often estimated based on background knowledge, previously available data, and assumptions about the form of the underlying distributions. A second practical difficulty is the significant computational cost required to determine the Bayes optimal hypothesis in the general case (linear in the number of candidate hypotheses). In certain specialized situations, this computational cost can be significantly reduced.

The remainder of this chapter is organized as follows. Section 6.2 introduces Bayes theorem and defines maximum likelihood and maximum a posteriori probability hypotheses. The four subsequent sections then apply this probabilistic framework to analyze several issues and learning algorithms discussed in earlier chapters. For example, we show that several previously described algorithms output maximum likelihood hypotheses, under certain assumptions. The remaining sections then introduce a number of learning algorithms that explicitly manipulate probabilities. These include the Bayes optimal classifier, Gibbs algorithm, and naive Bayes classifier. Finally, we discuss Bayesian belief networks, a relatively recent approach to learning based on probabilistic reasoning, and the EM algorithm, a widely used algorithm for learning in the presence of unobserved variables.

6.2 BAYES THEOREM

In machine learning we are often interested in determining the best hypothesis from some space H, given the observed training data D. One way to specify what we mean by the *best* hypothesis is to say that we demand the *most probable* hypothesis, given the data D plus any initial knowledge about the prior probabilities of the various hypotheses in H. Bayes theorem provides a direct method for calculating such probabilities. More precisely, Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

To define Bayes theorem precisely, let us first introduce a little notation. We shall write $P(h)$ to denote the initial probability that hypothesis h holds, before we have observed the training data. $P(h)$ is often called the *prior probability* of h and may reflect any background knowledge we have about the chance that h is a correct hypothesis. If we have no such prior knowledge, then we might simply assign the same prior probability to each candidate hypothesis. Similarly, we will write $P(D)$ to denote the prior probability that training data D will be observed (i.e., the probability of D given no knowledge about which hypothesis holds). Next, we will write $P(D|h)$ to denote the probability of observing data D given some world in which hypothesis h holds. More generally, we write $P(x|y)$ to denote the probability of x given y. In machine learning problems we are interested in the probability $P(h|D)$ that h holds given the observed training data D. $P(h|D)$ is called the *posterior probability* of h, because it reflects our confidence that h holds after we have seen the training data D. Notice the posterior probability $P(h|D)$ reflects the influence of the training data D, in contrast to the prior probability $P(h)$, which is independent of D.

Bayes theorem is the cornerstone of Bayesian learning methods because it provides a way to calculate the posterior probability $P(h|D)$, from the prior probability $P(h)$, together with $P(D)$ and $P(D|h)$.

Bayes theorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{6.1}$$

As one might intuitively expect, $P(h|D)$ increases with $P(h)$ and with $P(D|h)$ according to Bayes theorem. It is also reasonable to see that $P(h|D)$ decreases as $P(D)$ increases, because the more probable it is that D will be observed independent of h, the less evidence D provides in support of h.

In many learning scenarios, the learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis $h \in H$ given the observed data D (or at least one of the maximally probable if there are several). Any such maximally probable hypothesis is called a *maximum a posteriori* (MAP) hypothesis. We can determine the MAP hypotheses by using Bayes theorem to calculate the posterior probability of each candidate hypothesis. More precisely, we will say that h_{MAP} is a MAP hypothesis provided

$$\begin{aligned} h_{MAP} &\equiv \underset{h \in H}{\operatorname{argmax}} P(h|D) \\ &= \underset{h \in H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)} \\ &= \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h) \end{aligned} \tag{6.2}$$

Notice in the final step above we dropped the term $P(D)$ because it is a constant independent of h.

In some cases, we will assume that every hypothesis in H is equally probable a priori ($P(h_i) = P(h_j)$ for all h_i and h_j in H). In this case we can further simplify Equation (6.2) and need only consider the term $P(D|h)$ to find the most probable hypothesis. $P(D|h)$ is often called the *likelihood* of the data D given h, and any hypothesis that maximizes $P(D|h)$ is called a *maximum likelihood* (ML) hypothesis, h_{ML}.

$$h_{ML} \equiv \underset{h \in H}{\operatorname{argmax}} P(D|h) \tag{6.3}$$

In order to make clear the connection to machine learning problems, we introduced Bayes theorem above by referring to the data D as training examples of some target function and referring to H as the space of candidate target functions. In fact, Bayes theorem is much more general than suggested by this discussion. It can be applied equally well to any set H of mutually exclusive propositions whose probabilities sum to one (e.g., "the sky is blue," and "the sky is not blue"). In this chapter, we will at times consider cases where H is a hypothesis space containing possible target functions and the data D are training examples. At other times we will consider cases where H is some other set of mutually exclusive propositions, and D is some other kind of data.

6.2.1 An Example

To illustrate Bayes rule, consider a medical diagnosis problem in which there are two alternative hypotheses: (1) that the patient has a particular form of cancer, and (2) that the patient does not. The available data is from a particular laboratory

test with two possible outcomes: $\oplus$ (positive) and $\ominus$ (negative). We have prior knowledge that over the entire population of people only .008 have this disease. Furthermore, the lab test is only an imperfect indicator of the disease. The test returns a correct positive result in only 98% of the cases in which the disease is actually present and a correct negative result in only 97% of the cases in which the disease is not present. In other cases, the test returns the opposite result. The above situation can be summarized by the following probabilities:

$$\begin{aligned} P(cancer) &= .008, & P(\neg cancer) &= .992 \\ P(\oplus|cancer) &= .98, & P(\ominus|cancer) &= .02 \\ P(\oplus|\neg cancer) &= .03, & P(\ominus|\neg cancer) &= .97 \end{aligned}$$

Suppose we now observe a new patient for whom the lab test returns a positive result. Should we diagnose the patient as having cancer or not? The maximum a posteriori hypothesis can be found using Equation (6.2):

$$P(\oplus|cancer)P(cancer) = (.98).008 = .0078$$

$$P(\oplus|\neg cancer)P(\neg cancer) = (.03).992 = .0298$$

Thus, $h_{MAP} = \neg cancer$. The exact posterior probabilities can also be determined by normalizing the above quantities so that they sum to 1 (e.g., $P(cancer|\oplus) = \frac{.0078}{.0078+.0298} = .21$). This step is warranted because Bayes theorem states that the posterior probabilities are just the above quantities divided by the probability of the data, $P(\oplus)$. Although $P(\oplus)$ was not provided directly as part of the problem statement, we can calculate it in this fashion because we know that $P(cancer|\oplus)$ and $P(\neg cancer|\oplus)$ must sum to 1 (i.e., either the patient has cancer or they do not). Notice that while the posterior probability of *cancer* is significantly higher than its prior probability, the most probable hypothesis is still that the patient does not have cancer.

As this example illustrates, the result of Bayesian inference depends strongly on the prior probabilities, which must be available in order to apply the method directly. Note also that in this example the hypotheses are not completely accepted or rejected, but rather become more or less probable as more data is observed.

Basic formulas for calculating probabilities are summarized in Table 6.1.

6.3 BAYES THEOREM AND CONCEPT LEARNING

What is the relationship between Bayes theorem and the problem of concept learning? Since Bayes theorem provides a principled way to calculate the posterior probability of each hypothesis given the training data, we can use it as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, then outputs the most probable. This section considers such a brute-force Bayesian concept learning algorithm, then compares it to concept learning algorithms we considered in Chapter 2. As we shall see, one interesting result of this comparison is that under certain conditions several algorithms discussed in earlier chapters output the same hypotheses as this brute-force Bayesian

- *Product rule*: probability $P(A \wedge B)$ of a conjunction of two events A and B

$$P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$$

- *Sum rule*: probability of a disjunction of two events A and B

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

- *Bayes theorem*: the posterior probability $P(h|D)$ of h given D

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- *Theorem of total probability*: if events $A_1, \ldots, A_n$ are mutually exclusive with $\sum_{i=1}^{n} P(A_i) = 1$, then

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

TABLE 6.1
Summary of basic probability formulas.

algorithm, despite the fact that they do not explicitly manipulate probabilities and are considerably more efficient.

6.3.1 Brute-Force Bayes Concept Learning

Consider the concept learning problem first introduced in Chapter 2. In particular, assume the learner considers some finite hypothesis space H defined over the instance space X, in which the task is to learn some target concept $c : X \rightarrow \{0, 1\}$. As usual, we assume that the learner is given some sequence of training examples $\langle \langle x_1, d_1 \rangle \ldots \langle x_m, d_m \rangle \rangle$ where x_i is some instance from X and where d_i is the target value of x_i (i.e., $d_i = c(x_i)$). To simplify the discussion in this section, we assume the sequence of instances $\langle x_1 \ldots x_m \rangle$ is held fixed, so that the training data D can be written simply as the sequence of target values $D = \langle d_1 \ldots d_m \rangle$. It can be shown (see Exercise 6.4) that this simplification does not alter the main conclusions of this section.

We can design a straightforward concept learning algorithm to output the maximum a posteriori hypothesis, based on Bayes theorem, as follows:

BRUTE-FORCE MAP LEARNING algorithm

1. For each hypothesis h in H, calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(h|D)$$

This algorithm may require significant computation, because it applies Bayes theorem to each hypothesis in H to calculate $P(h|D)$. While this may prove impractical for large hypothesis spaces, the algorithm is still of interest because it provides a standard against which we may judge the performance of other concept learning algorithms.

In order specify a learning problem for the BRUTE-FORCE MAP LEARNING algorithm we must specify what values are to be used for $P(h)$ and for $P(D|h)$ (as we shall see, $P(D)$ will be determined once we choose the other two). We may choose the probability distributions $P(h)$ and $P(D|h)$ in any way we wish, to describe our prior knowledge about the learning task. Here let us choose them to be consistent with the following assumptions:

1. The training data D is noise free (i.e., $d_i = c(x_i)$).
2. The target concept c is contained in the hypothesis space H.
3. We have no a priori reason to believe that any hypothesis is more probable than any other.

Given these assumptions, what values should we specify for $P(h)$? Given no prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis h in H. Furthermore, because we assume the target concept is contained in H we should require that these prior probabilities sum to 1. Together these constraints imply that we should choose

$$P(h) = \frac{1}{|H|} \quad \text{for all } h \text{ in } H$$

What choice shall we make for $P(D|h)$? $P(D|h)$ is the probability of observing the target values $D = \langle d_1 \ldots d_m \rangle$ for the fixed set of instances $\langle x_1 \ldots x_m \rangle$, given a world in which hypothesis h holds (i.e., given a world in which h is the correct description of the target concept c). Since we assume noise-free training data, the probability of observing classification d_i given h is just 1 if $d_i = h(x_i)$ and 0 if $d_i \neq h(x_i)$. Therefore,

$$P(D|h) = \begin{cases} 1 \text{ if } d_i = h(x_i) \text{ for all } d_i \text{ in } D \\ 0 \text{ otherwise} \end{cases} \tag{6.4}$$

In other words, the probability of data D given hypothesis h is 1 if D is consistent with h, and 0 otherwise.

Given these choices for $P(h)$ and for $P(D|h)$ we now have a fully-defined problem for the above BRUTE-FORCE MAP LEARNING algorithm. Let us consider the first step of this algorithm, which uses Bayes theorem to compute the posterior probability $P(h|D)$ of each hypothesis h given the observed training data D.

Recalling Bayes theorem, we have

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

First consider the case where h is inconsistent with the training data D. Since Equation (6.4) defines $P(D|h)$ to be 0 when h is inconsistent with D, we have

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0 \text{ if } h \text{ is inconsistent with } D$$

The posterior probability of a hypothesis inconsistent with D is zero.

Now consider the case where h is consistent with D. Since Equation (6.4) defines $P(D|h)$ to be 1 when h is consistent with D, we have

$$\begin{aligned} P(h|D) &= \frac{1 \cdot \frac{1}{|H|}}{P(D)} \\ &= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} \\ &= \frac{1}{|VS_{H,D}|} \text{ if } h \text{ is consistent with } D \end{aligned}$$

where $VS_{H,D}$ is the subset of hypotheses from H that are consistent with D (i.e., $VS_{H,D}$ is the version space of H with respect to D as defined in Chapter 2). It is easy to verify that $P(D) = \frac{|VS_{H,D}|}{|H|}$ above, because the sum over all hypotheses of $P(h|D)$ must be one and because the number of hypotheses from H consistent with D is by definition $|VS_{H,D}|$. Alternatively, we can derive $P(D)$ from the theorem of total probability (see Table 6.1) and the fact that the hypotheses are mutually exclusive (i.e., $(\forall i \neq j)(P(h_i \wedge h_j) = 0)$)

$$\begin{aligned} P(D) &= \sum_{h_i \in H} P(D|h_i) P(h_i) \\ &= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} + \sum_{h_i \notin VS_{H,D}} 0 \cdot \frac{1}{|H|} \\ &= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} \\ &= \frac{|VS_{H,D}|}{|H|} \end{aligned}$$

To summarize, Bayes theorem implies that the posterior probability $P(h|D)$ under our assumed $P(h)$ and $P(D|h)$ is

$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

where $|VS_{H,D}|$ is the number of hypotheses from H consistent with D. The evolution of probabilities associated with hypotheses is depicted schematically in Figure 6.1. Initially (Figure 6.1*a*) all hypotheses have the same probability. As training data accumulates (Figures 6.1*b* and 6.1*c*), the posterior probability for inconsistent hypotheses becomes zero while the total probability summing to one is shared equally among the remaining consistent hypotheses.

The above analysis implies that under our choice for $P(h)$ and $P(D|h)$, every *consistent* hypothesis has posterior probability $(1/|VS_{H,D}|)$, and every inconsistent hypothesis has posterior probability 0. Every consistent hypothesis is, therefore, a MAP hypothesis.

6.3.2 MAP Hypotheses and Consistent Learners

The above analysis shows that in the given setting, every hypothesis consistent with D is a MAP hypothesis. This statement translates directly into an interesting statement about a general class of learners that we might call *consistent learners.* We will say that a learning algorithm is a *consistent learner* provided it outputs a hypothesis that commits zero errors over the training examples. Given the above analysis, we can conclude that *every consistent learner outputs a MAP hypothesis, if we assume a uniform prior probability distribution over H (i.e., $P(h_i) = P(h_j)$ for all i, j), and if we assume deterministic, noise-free training data (i.e., $P(D|h) =$ 1 if D and h are consistent, and 0 otherwise).*

Consider, for example, the concept learning algorithm FIND-S discussed in Chapter 2. FIND-S searches the hypothesis space H from specific to general hypotheses, outputting a maximally specific consistent hypothesis (i.e., a maximally specific member of the version space). Because FIND-S outputs a consistent hypothesis, we know that it will output a MAP hypothesis under the probability distributions $P(h)$ and $P(D|h)$ defined above. Of course FIND-S does not explicitly manipulate probabilities at all—it simply outputs a maximally specific member

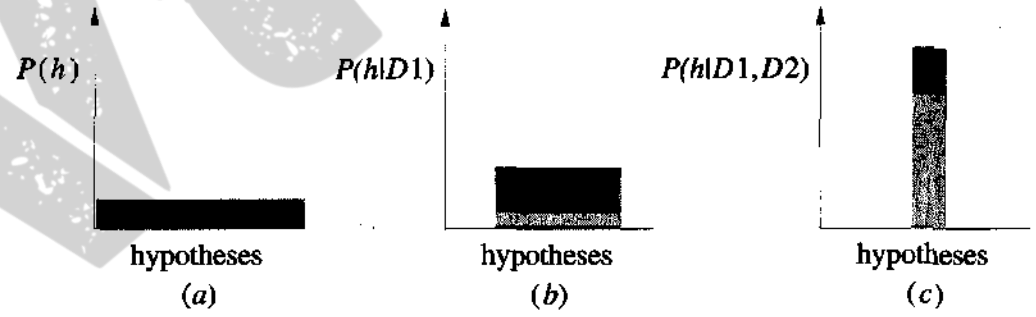


FIGURE 6.1
Evolution of posterior probabilities $P(h|D)$ with increasing training data. (*a*) Uniform priors assign equal probability to each hypothesis. As training data increases first to $D1$ (*b*), then to $D1 \wedge D2$ (*c*), the posterior probability of inconsistent hypotheses becomes zero, while posterior probabilities increase for hypotheses remaining in the version space.

of the version space. However, by identifying distributions for $P(h)$ and $P(D|h)$ under which its output hypotheses will be MAP hypotheses, we have a useful way of characterizing the behavior of FIND-S.

Are there other probability distributions for $P(h)$ and $P(D|h)$ under which FIND-S outputs MAP hypotheses? Yes. Because FIND-S outputs a *maximally specific* hypothesis from the version space, its output hypothesis will be a MAP hypothesis relative to any prior probability distribution that favors more specific hypotheses. More precisely, suppose $\mathcal{H}$ is any probability distribution $P(h)$ over H that assigns $P(h_1) \geq P(h_2)$ if h_1 is more specific than h_2. Then it can be shown that FIND-S outputs a MAP hypothesis assuming the prior distribution $\mathcal{H}$ and the same distribution $P(D|h)$ discussed above.

To summarize the above discussion, the Bayesian framework allows one way to characterize the behavior of learning algorithms (e.g., FIND-S), even when the learning algorithm does not explicitly manipulate probabilities. By identifying probability distributions $P(h)$ and $P(D|h)$ under which the algorithm outputs optimal (i.e., MAP) hypotheses, we can characterize the implicit assumptions under which this algorithm behaves optimally.

Using the Bayesian perspective to characterize learning algorithms in this way is similar in spirit to characterizing the inductive bias of the learner. Recall that in Chapter 2 we defined the inductive bias of a learning algorithm to be the set of assumptions B sufficient to *deductively* justify the inductive inference performed by the learner. For example, we described the inductive bias of the CANDIDATE-ELIMINATION algorithm as the assumption that the target concept c is included in the hypothesis space H. Furthermore, we showed there that the output of this learning algorithm follows deductively from its inputs plus this implicit inductive bias assumption. The above Bayesian interpretation provides an alternative way to characterize the assumptions implicit in learning algorithms. Here, instead of modeling the inductive inference method by an equivalent deductive system, we model it by an equivalent *probabilistic reasoning* system based on Bayes theorem. And here the implicit assumptions that we attribute to the learner are assumptions of the form "the prior probabilities over H are given by the distribution $P(h)$, and the strength of data in rejecting or accepting a hypothesis is given by $P(D|h)$." The definitions of $P(h)$ and $P(D|h)$ given in this section characterize the implicit assumptions of the CANDIDATE-ELIMINATION and FIND-S algorithms. A probabilistic reasoning system based on Bayes theorem will exhibit input-output behavior equivalent to these algorithms, provided it is given these assumed probability distributions.

The discussion throughout this section corresponds to a special case of Bayesian reasoning, because we considered the case where $P(D|h)$ takes on values of only 0 and 1, reflecting the deterministic predictions of hypotheses and the assumption of noise-free training data. As we shall see in the next section, we can also model learning from noisy training data, by allowing $P(D|h)$ to take on values other than 0 and 1, and by introducing into $P(D|h)$ additional assumptions about the probability distributions that govern the noise.

6.4 MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES

As illustrated in the above section, Bayesian analysis can sometimes be used to show that a particular learning algorithm outputs MAP hypotheses even though it may not explicitly use Bayes rule or calculate probabilities in any form.

In this section we consider the problem of learning a continuous-valued target function—a problem faced by many learning approaches such as neural network learning, linear regression, and polynomial curve fitting. A straightforward Bayesian analysis will show that *under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis.* The significance of this result is that it provides a Bayesian justification (under certain assumptions) for many neural network and other curve fitting methods that attempt to minimize the sum of squared errors over the training data.

Consider the following problem setting. Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X (i.e., each h in H is a function of the form $h : X \rightarrow \Re$, where $\Re$ represents the set of real numbers). The problem faced by L is to learn an unknown target function $f : X \rightarrow \Re$ drawn from H. A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution. More precisely, each training example is a pair of the form $\langle x_i, d_i \rangle$ where $d_i = f(x_i) + e_i$. Here $f(x_i)$ is the noise-free value of the target function and e_i is a random variable representing the noise. It is assumed that the values of the e_i are drawn independently and that they are distributed according to a Normal distribution with zero mean. The task of the learner is to output a maximum likelihood hypothesis, or, equivalently, a MAP hypothesis assuming all hypotheses are equally probable a priori.

A simple example of such a problem is learning a linear function, though our analysis applies to learning arbitrary real-valued functions. Figure 6.2 illustrates

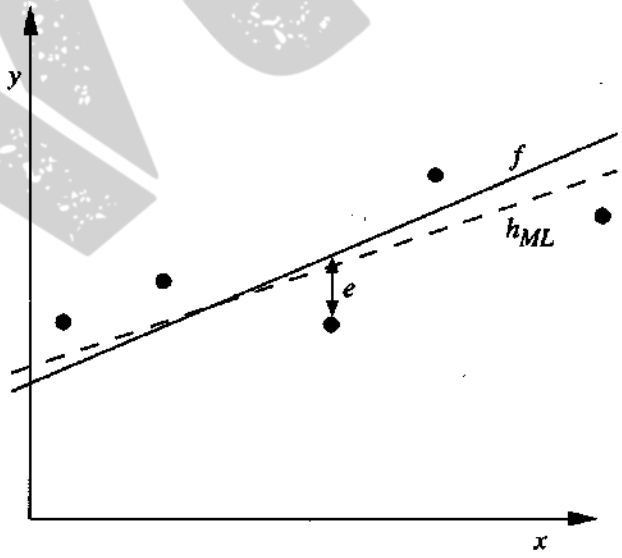


FIGURE 6.2
Learning a real-valued function. The target function f corresponds to the solid line. The training examples $\langle x_i, d_i \rangle$ are assumed to have Normally distributed noise e_i with zero mean added to the true target value $f(x_i)$. The dashed line corresponds to the linear function that minimizes the sum of squared errors. Therefore, it is the maximum likelihood hypothesis h_{ML}, given these five training examples.

a linear target function f depicted by the solid line, and a set of noisy training examples of this target function. The dashed line corresponds to the hypothesis h_{ML} with least-squared training error, hence the maximum likelihood hypothesis. Notice that the maximum likelihood hypothesis is not necessarily identical to the correct hypothesis, f, because it is inferred from only a limited sample of noisy training data.

Before showing why a hypothesis that minimizes the sum of squared errors in this setting is also a maximum likelihood hypothesis, let us quickly review two basic concepts from probability theory: probability densities and Normal distributions. First, in order to discuss probabilities over continuous variables such as e, we must introduce probability *densities*. The reason, roughly, is that we wish for the total probability over all possible values of the random variable to sum to one. In the case of continuous variables we cannot achieve this by assigning a finite probability to each of the infinite set of possible values for the random variable. Instead, we speak of a probability *density* for continuous variables such as e and require that the integral of this probability density over all possible values be one. In general we will use lower case p to refer to the probability density function, to distinguish it from a finite probability P (which we will sometimes refer to as a probability *mass*). The probability density $p(x_0)$ is the limit as ϵ goes to zero, of $\frac{1}{\epsilon}$ times the probability that x will take on a value in the interval $[x_0, x_0 + \epsilon)$.

Probability density function:

$$p(x_0) \equiv \lim_{\epsilon \to 0} \frac{1}{\epsilon} P(x_0 \leq x < x_0 + \epsilon)$$

Second, we stated that the random noise variable e is generated by a Normal probability distribution. A Normal distribution is a smooth, bell-shaped distribution that can be completely characterized by its mean μ and its standard deviation σ. See Table 5.4 for a precise definition.

Given this background we now return to the main issue: showing that the least-squared error hypothesis is, in fact, the maximum likelihood hypothesis within our problem setting. We will show this by deriving the maximum likelihood hypothesis starting with our earlier definition Equation (6.3), but using lower case p to refer to the probability density

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}}\, p(D|h)$$

As before, we assume a fixed set of training instances $\langle x_1 \ldots x_m \rangle$ and therefore consider the data D to be the corresponding sequence of target values $D = \langle d_1 \ldots d_m \rangle$. Here $d_i = f(x_i) + e_i$. Assuming the training examples are mutually independent given h, we can write $P(D|h)$ as the product of the various $p(d_i|h)$

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} p(d_i|h)$$

Given that the noise e_i obeys a Normal distribution with zero mean and unknown variance σ^2, each d_i must also obey a Normal distribution with variance σ^2 centered around the true target value $f(x_i)$ rather than zero. Therefore $p(d_i|h)$ can be written as a Normal distribution with variance σ^2 and mean $\mu = f(x_i)$. Let us write the formula for this Normal distribution to describe $p(d_i|h)$, beginning with the general formula for a Normal distribution from Table 5.4 and substituting the appropriate μ and σ^2. Because we are writing the expression for the probability of d_i given that h is the correct description of the target function f, we will also substitute $\mu = f(x_i) = h(x_i)$, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2}$$

$$= \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2}$$

We now apply a transformation that is common in maximum likelihood calculations: Rather than maximizing the above complicated expression we shall choose to maximize its (less complicated) logarithm. This is justified because $\ln p$ is a monotonic function of p. Therefore maximizing $\ln p$ also maximizes p.

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of h, and can therefore be discarded, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity.

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally, we can again discard constants that are independent of h.

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} (d_i - h(x_i))^2 \tag{6.6}$$

Thus, Equation (6.6) shows that the maximum likelihood hypothesis h_{ML} is the one that minimizes the sum of the squared errors between the observed training values d_i and the hypothesis predictions $h(x_i)$. This holds under the assumption that the observed training values d_i are generated by adding random noise to

the true target value, where this random noise is drawn independently for each example from a Normal distribution with zero mean. As the above derivation makes clear, the squared error term $(d_i - h(x_i))^2$ follows directly from the exponent in the definition of the Normal distribution. Similar derivations can be performed starting with other assumed noise distributions, producing different results.

Notice the structure of the above derivation involves selecting the hypothesis that maximizes the logarithm of the likelihood ($\ln p(D|h)$) in order to determine the most probable hypothesis. As noted earlier, this yields the same result as maximizing the likelihood $p(D|h)$. This approach of working with the log likelihood is common to many Bayesian analyses, because it is often more mathematically tractable than working directly with the likelihood. Of course, as noted earlier, the maximum likelihood hypothesis might not be the MAP hypothesis, but if one assumes uniform prior probabilities over the hypotheses then it is.

Why is it reasonable to choose the Normal distribution to characterize noise? One reason, it must be admitted, is that it allows for a mathematically straightforward analysis. A second reason is that the smooth, bell-shaped distribution is a good approximation to many types of noise in physical systems. In fact, the Central Limit Theorem discussed in Chapter 5 shows that the sum of a sufficiently large number of independent, identically distributed random variables itself obeys a Normal distribution, regardless of the distributions of the individual variables. This implies that noise generated by the sum of very many independent, but identically distributed factors will itself be Normally distributed. Of course, in reality, different components that contribute to noise might not follow identical distributions, in which case this theorem will not necessarily justify our choice.

Minimizing the sum of squared errors is a common approach in many neural network, curve fitting, and other approaches to approximating real-valued functions. Chapter 4 describes gradient descent methods that seek the least-squared error hypothesis in neural network learning.

Before leaving our discussion of the relationship between the maximum likelihood hypothesis and the least-squared error hypothesis, it is important to note some limitations of this problem setting. The above analysis considers noise only in the *target value* of the training example and does not consider noise in the *attributes describing the instances themselves*. For example, if the problem is to learn to predict the weight of someone based on that person's age and height, then the above analysis assumes noise in measurements of weight, but perfect measurements of age and height. The analysis becomes significantly more complex as these simplifying assumptions are removed.

6.5 MAXIMUM LIKELIHOOD HYPOTHESES FOR PREDICTING PROBABILITIES

In the problem setting of the previous section we determined that the maximum likelihood hypothesis is the one that minimizes the sum of squared errors over the training examples. In this section we derive an analogous criterion for a second setting that is common in neural network learning: learning to predict probabilities.

Consider the setting in which we wish to learn a nondeterministic (probabilistic) function $f : X \rightarrow \{0, 1\}$, which has two discrete output values. For example, the instance space X might represent medical patients in terms of their symptoms, and the target function $f(x)$ might be 1 if the patient survives the disease and 0 if not. Alternatively, X might represent loan applicants in terms of their past credit history, and $f(x)$ might be 1 if the applicant successfully repays their next loan and 0 if not. In both of these cases we might well expect f to be probabilistic. For example, among a collection of patients exhibiting the same set of observable symptoms, we might find that 92% survive, and 8% do not. This unpredictability could arise from our inability to observe all the important distinguishing features of the patients, or from some genuinely probabilistic mechanism in the evolution of the disease. Whatever the source of the problem, the effect is that we have a target function $f(x)$ whose output is a probabilistic function of the input.

Given this problem setting, we might wish to learn a neural network (or other real-valued function approximator) whose output is the *probability* that $f(x) = 1$. In other words, we seek to learn the target function, $f' : X \rightarrow [0, 1]$, such that $f'(x) = P(f(x) = 1)$. In the above medical patient example, if x is one of those indistinguishable patients of which 92% survive, then $f'(x) = 0.92$ whereas the probabilistic function $f(x)$ will be equal to 1 in 92% of cases and equal to 0 in the remaining 8%.

How can we learn f' using, say, a neural network? One obvious, brute-force way would be to first collect the observed frequencies of 1's and 0's for each possible value of x and to then train the neural network to output the target frequency for each x. As we shall see below, we can instead train a neural network directly from the observed training examples of f, yet still derive a maximum likelihood hypothesis for f'.

What criterion should we optimize in order to find a maximum likelihood hypothesis for f' in this setting? To answer this question we must first obtain an expression for $P(D|h)$. Let us assume the training data D is of the form $D = \{\langle x_1, d_1\rangle \ldots \langle x_m, d_m\rangle\}$, where d_i is the observed 0 or 1 value for $f(x_i)$.

Recall that in the maximum likelihood, least-squared error analysis of the previous section, we made the simplifying assumption that the instances $\langle x_1 \ldots x_m\rangle$ were fixed. This enabled us to characterize the data by considering only the target values d_i. Although we could make a similar simplifying assumption in this case, let us avoid it here in order to demonstrate that it has no impact on the final outcome. Thus treating both x_i and d_i as random variables, and assuming that each training example is drawn independently, we can write $P(D|h)$ as

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h) \tag{6.7}$$

It is reasonable to assume, furthermore, that the probability of encountering any particular instance x_i is independent of the hypothesis h. For example, the probability that our training set contains a particular *patient* x_i is independent of our hypothesis about survival rates (though of course the *survival* d_i of the patient

does depend strongly on h). When x is independent of h we can rewrite the above expression (applying the product rule from Table 6.1) as

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h) = \prod_{i=1}^{m} P(d_i|h, x_i)P(x_i) \tag{6.8}$$

Now what is the probability $P(d_i|h, x_i)$ of observing $d_i = 1$ for a single instance x_i, given a world in which hypothesis h holds? Recall that h is our hypothesis regarding the target function, which computes this very probability. Therefore, $P(d_i = 1|h, x_i) = h(x_i)$, and in general

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases} \tag{6.9}$$

In order to substitute this into the Equation (6.8) for $P(D|h)$, let us first re-express it in a more mathematically manipulable form, as

$$P(d_i|h, x_i) = h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} \tag{6.10}$$

It is easy to verify that the expressions in Equations (6.9) and (6.10) are equivalent. Notice that when $d_i = 1$, the second term from Equation (6.10), $(1 - h(x_i))^{1-d_i}$, becomes equal to 1. Hence $P(d_i = 1|h, x_i) = h(x_i)$, which is equivalent to the first case in Equation (6.9). A similar analysis shows that the two equations are also equivalent when $d_i = 0$.

We can use Equation (6.10) to substitute for $P(d_i|h, x_i)$ in Equation (6.8) to obtain

$$P(D|h) = \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} P(x_i) \tag{6.11}$$

Now we write an expression for the maximum likelihood hypothesis

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} P(x_i)$$

The last term is a constant independent of h, so it can be dropped

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} \tag{6.12}$$

The expression on the right side of Equation (6.12) can be seen as a generalization of the *Binomial distribution* described in Table 5.3. The expression in Equation (6.12) describes the probability that flipping each of m distinct coins will produce the outcome $\langle d_1 \ldots d_m \rangle$, assuming that each coin x_i has probability $h(x_i)$ of producing a heads. Note the Binomial distribution described in Table 5.3 is

similar, but makes the additional assumption that the coins have identical probabilities of turning up heads (i.e., that $h(x_i) = h(x_j), \forall i, j$). In both cases we assume the outcomes of the coin flips are mutually independent—an assumption that fits our current setting.

As in earlier cases, we will find it easier to work with the log of the likelihood, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)) \tag{6.13}$$

Equation (6.13) describes the quantity that must be maximized in order to obtain the maximum likelihood hypothesis in our current problem setting. This result is analogous to our earlier result showing that minimizing the sum of squared errors produces the maximum likelihood hypothesis in the earlier problem setting. Note the similarity between Equation (6.13) and the general form of the entropy function, $-\sum_i p_i \log p_i$, discussed in Chapter 3. Because of this similarity, the negation of the above quantity is sometimes called the *cross entropy*.

6.5.1 Gradient Search to Maximize Likelihood in a Neural Net

Above we showed that maximizing the quantity in Equation (6.13) yields the maximum likelihood hypothesis. Let us use $G(h, D)$ to denote this quantity. In this section we derive a weight-training rule for neural network learning that seeks to maximize $G(h, D)$ using gradient ascent.

As discussed in Chapter 4, the gradient of $G(h, D)$ is given by the vector of partial derivatives of $G(h, D)$ with respect to the various network weights that define the hypothesis h represented by the learned network (see Chapter 4 for a general discussion of gradient-descent search and for details of the terminology that we reuse here). In this case, the partial derivative of $G(h, D)$ with respect to weight w_{jk} from input k to unit j is

$$\begin{aligned}
\frac{\partial G(h, D)}{\partial w_{jk}} &= \sum_{i=1}^{m} \frac{\partial G(h, D)}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\
&= \sum_{i=1}^{m} \frac{\partial (d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)))}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\
&= \sum_{i=1}^{m} \frac{d_i - h(x_i)}{h(x_i)(1 - h(x_i))} \frac{\partial h(x_i)}{\partial w_{jk}}
\end{aligned} \tag{6.14}$$

To keep our analysis simple, suppose our neural network is constructed from a single layer of sigmoid units. In this case we have

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \sigma'(x_i) x_{ijk} = h(x_i)(1 - h(x_i)) x_{ijk}$$

where x_{ijk} is the kth input to unit j for the ith training example, and $\sigma'(x)$ is the derivative of the sigmoid squashing function (again, see Chapter 4). Finally,

substituting this expression into Equation (6.14), we obtain a simple expression for the derivatives that constitute the gradient

$$\frac{\partial G(h, D)}{\partial w_{jk}} = \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk}$$

Because we seek to maximize rather than minimize $P(D|h)$, we perform gradient ascent rather than gradient descent search. On each iteration of the search the weight vector is adjusted in the direction of the gradient, using the weight-update rule

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

where

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk} \tag{6.15}$$

and where η is a small positive constant that determines the step size of the gradient ascent search.

It is interesting to compare this weight-update rule to the weight-update rule used by the BACKPROPAGATION algorithm to minimize the sum of squared errors between predicted and observed network outputs. The BACKPROPAGATION update rule for output unit weights (see Chapter 4), re-expressed using our current notation, is

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

where

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} h(x_i)(1 - h(x_i))(d_i - h(x_i))\ x_{ijk}$$

Notice this is similar to the rule given in Equation (6.15) except for the extra term $h(x_i)(1 - h(x_i))$, which is the derivative of the sigmoid function.

To summarize, these two weight update rules converge toward maximum likelihood hypotheses in two different settings. The rule that minimizes sum of squared error seeks the maximum likelihood hypothesis under the assumption that the training data can be modeled by Normally distributed noise added to the target function value. The rule that minimizes cross entropy seeks the maximum likelihood hypothesis under the assumption that the observed boolean value is a probabilistic function of the input instance.

6.6 MINIMUM DESCRIPTION LENGTH PRINCIPLE

Recall from Chapter 3 the discussion of Occam's razor, a popular inductive bias that can be summarized as "choose the shortest explanation for the observed data." In that chapter we discussed several arguments in the long-standing debate regarding Occam's razor. Here we consider a Bayesian perspective on this issue

and a closely related principle called the Minimum Description Length (MDL) principle.

The Minimum Description Length principle is motivated by interpreting the definition of h_{MAP} in the light of basic concepts from information theory. Consider again the now familiar definition of h_{MAP}.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

which can be equivalently expressed in terms of maximizing the $\log_2$

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}} \log_2 P(D|h) + \log_2 P(h)$$

or alternatively, minimizing the negative of this quantity

$$h_{MAP} = \underset{h \in H}{\operatorname{argmin}} -\log_2 P(D|h) - \log_2 P(h) \tag{6.16}$$

Somewhat surprisingly, Equation (6.16) can be interpreted as a statement that short hypotheses are preferred, assuming a particular representation scheme for encoding hypotheses and data. To explain this, let us introduce a basic result from information theory: Consider the problem of designing a code to transmit messages drawn at random, where the probability of encountering message i is p_i. We are interested here in the most compact code; that is, we are interested in the code that minimizes the expected number of bits we must transmit in order to encode a message drawn at random. Clearly, to minimize the expected code length we should assign shorter codes to messages that are more probable. Shannon and Weaver (1949) showed that the optimal code (i.e., the code that minimizes the expected message length) assigns $-\log_2 p_i$ bits[†] to encode message i. We will refer to the number of bits required to encode message i using code C as the *description length of message i with respect to C*, which we denote by $L_C(i)$.

Let us interpret Equation (6.16) in light of the above result from coding theory.

- $-\log_2 P(h)$ is the description length of h under the optimal encoding for the hypothesis space H. In other words, this is the size of the description of hypothesis h using this optimal representation. In our notation, $L_{C_H}(h) = -\log_2 P(h)$, where C_H is the optimal code for hypothesis space H.
- $-\log_2 P(D|h)$ is the description length of the training data D given hypothesis h, under its optimal encoding. In our notation, $L_{C_{D|h}}(D|h) = -\log_2 P(D|h)$, where $C_{D|h}$ is the optimal code for describing data D assuming that both the sender and receiver know the hypothesis h.

- Therefore we can rewrite Equation (6.16) to show that h_{MAP} is the hypothesis h that minimizes the sum given by the description length of the hypothesis plus the description length of the data given the hypothesis.

$$h_{MAP} = \underset{h}{\operatorname{argmin}} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

where C_H and $C_{D|h}$ are the optimal encodings for H and for D given h, respectively.

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths. Of course to apply this principle in practice we must choose specific encodings or representations appropriate for the given learning task. Assuming we use the codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis, we can state the MDL principle as

Minimum Description Length principle: Choose h_{MDL} where

$$h_{MDL} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h) \tag{6.17}$$

The above analysis shows that if we choose C_1 to be the optimal encoding of hypotheses C_H, and if we choose C_2 to be the optimal encoding $C_{D|h}$, then $h_{MDL} = h_{MAP}$.

Intuitively, we can think of the MDL principle as recommending the shortest method for re-encoding the training data, where we count both the size of the hypothesis and any additional cost of encoding the data given this hypothesis.

Let us consider an example. Suppose we wish to apply the MDL principle to the problem of learning decision trees from some training data. What should we choose for the representations C_1 and C_2 of hypotheses and data? For C_1 we might naturally choose some obvious encoding of decision trees, in which the description length grows with the number of nodes in the tree and with the number of edges. How shall we choose the encoding C_2 of the data given a particular decision tree hypothesis? To keep things simple, suppose that the sequence of instances $\langle x_1 \ldots x_m \rangle$ is already known to both the transmitter and receiver, so that we need only transmit the classifications $\langle f(x_1) \ldots f(x_m) \rangle$. (Note the cost of transmitting the instances themselves is independent of the correct hypothesis, so it does not affect the selection of h_{MDL} in any case.) Now if the training classifications $\langle f(x_1) \ldots f(x_m) \rangle$ are identical to the predictions of the hypothesis, then there is no need to transmit any information about these examples (the receiver can compute these values once it has received the hypothesis). The description length of the classifications given the hypothesis in this case is, therefore, zero. In the case where some examples are misclassified by h, then for each misclassification we need to transmit a message that identifies which example is misclassified (which can be done using at most $\log_2 m$ bits) as well

as its correct classification (which can be done using at most $\log_2 k$ bits, where k is the number of possible classifications). The hypothesis h_{MDL} under the encodings C_1 and C_2 is just the one that minimizes the sum of these description lengths.

Thus the MDL principle provides a way of trading off hypothesis complexity for the number of errors committed by the hypothesis. It might select a shorter hypothesis that makes a few errors over a longer hypothesis that perfectly classifies the training data. Viewed in this light, it provides one method for dealing with the issue of *overfitting* the data.

Quinlan and Rivest (1989) describe experiments applying the MDL principle to choose the best size for a decision tree. They report that the MDL-based method produced learned trees whose accuracy was comparable to that of the standard tree-pruning methods discussed in Chapter 3. Mehta et al. (1995) describe an alternative MDL-based approach to decision tree pruning, and describe experiments in which an MDL-based approach produced results comparable to standard tree-pruning methods.

What shall we conclude from this analysis of the Minimum Description Length principle? Does this prove once and for all that short hypotheses are best? No. What we have shown is only that *if* a representation of hypotheses is chosen so that the size of hypothesis h is $-\log_2 P(h)$, and *if* a representation for exceptions is chosen so that the encoding length of D given h is equal to $-\log_2 P(D|h)$, *then* the MDL principle produces MAP hypotheses. However, to show that we have such a representation we must know all the prior probabilities $P(h)$, as well as the $P(D|h)$. There is no reason to believe that the MDL hypothesis relative to *arbitrary* encodings C_1 and C_2 should be preferred. As a practical matter it might sometimes be easier for a human designer to specify a representation that captures knowledge about the relative probabilities of hypotheses than it is to fully specify the probability of each hypothesis. Descriptions in the literature on the application of MDL to practical learning problems often include arguments providing some form of justification for the encodings chosen for C_1 and C_2.

6.7 BAYES OPTIMAL CLASSIFIER

So far we have considered the question "what is the most probable *hypothesis* given the training data?" In fact, the question that is often of most significance is the closely related question "what is the most probable *classification* of the new instance given the training data?" Although it may seem that this second question can be answered by simply applying the MAP hypothesis to the new instance, in fact it is possible to do better.

To develop some intuitions consider a hypothesis space containing three hypotheses, h_1, h_2, and h_3. Suppose that the posterior probabilities of these hypotheses given the training data are .4, .3, and .3 respectively. Thus, h_1 is the MAP hypothesis. Suppose a new instance x is encountered, which is classified positive by h_1, but negative by h_2 and h_3. Taking all hypotheses into account, the probability that x is positive is .4 (the probability associated with h_1), and

the probability that it is negative is therefore .6. The most probable classification (negative) in this case is different from the classification generated by the MAP hypothesis.

In general, the most probable classification of the new instance is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities. If the possible classification of the new example can take on any value v_j from some set V, then the probability $P(v_j|D)$ that the correct classification for the new instance is v_j, is just

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

The optimal classification of the new instance is the value v_j, for which $P(v_j|D)$ is maximum.

Bayes optimal classification:

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \tag{6.18}$$

To illustrate in terms of the above example, the set of possible classifications of the new instance is $V = \{\oplus, \ominus\}$, and

$$P(h_1|D) = .4, \; P(\ominus|h_1) = 0, \; P(\oplus|h_1) = 1$$
$$P(h_2|D) = .3, \; P(\ominus|h_2) = 1, \; P(\oplus|h_2) = 0$$
$$P(h_3|D) = .3, \; P(\ominus|h_3) = 1, \; P(\oplus|h_3) = 0$$

therefore

$$\sum_{h_i \in H} P(\oplus|h_i)P(h_i|D) = .4$$
$$\sum_{h_i \in H} P(\ominus|h_i)P(h_i|D) = .6$$

and

$$\underset{v_j \in \{\oplus, \ominus\}}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = \ominus$$

Any system that classifies new instances according to Equation (6.18) is called a *Bayes optimal classifier*, or Bayes optimal learner. No other classification method using the same hypothesis space and same prior knowledge can outperform this method on average. This method maximizes the probability that the new instance is classified correctly, given the available data, hypothesis space, and prior probabilities over the hypotheses.

For example, in learning boolean concepts using version spaces as in the earlier section, the Bayes optimal classification of a new instance is obtained by taking a weighted vote among all members of the version space, with each candidate hypothesis weighted by its posterior probability.

Note one curious property of the Bayes optimal classifier is that the predictions it makes can correspond to a hypothesis not contained in H! Imagine using Equation (6.18) to classify every instance in X. The labeling of instances defined in this way need not correspond to the instance labeling of any single hypothesis h from H. One way to view this situation is to think of the Bayes optimal classifier as effectively considering a hypothesis space H' different from the space of hypotheses H to which Bayes theorem is being applied. In particular, H' effectively includes hypotheses that perform comparisons between linear combinations of predictions from multiple hypotheses in H.

6.8 GIBBS ALGORITHM

Although the Bayes optimal classifier obtains the best performance that can be achieved from the given training data, it can be quite costly to apply. The expense is due to the fact that it computes the posterior probability for every hypothesis in H and then combines the predictions of each hypothesis to classify each new instance.

An alternative, less optimal method is the Gibbs algorithm (see Opper and Haussler 1991), defined as follows:

1. Choose a hypothesis h from H at random, according to the posterior probability distribution over H.
2. Use h to predict the classification of the next instance x.

Given a new instance to classify, the Gibbs algorithm simply applies a hypothesis drawn at random according to the current posterior probability distribution. Surprisingly, it can be shown that under certain conditions the expected misclassification error for the Gibbs algorithm is at most twice the expected error of the Bayes optimal classifier (Haussler et al. 1994). More precisely, the expected value is taken over target concepts drawn at random according to the prior probability distribution assumed by the learner. Under this condition, the expected value of the error of the Gibbs algorithm is at worst twice the expected value of the error of the Bayes optimal classifier.

This result has an interesting implication for the concept learning problem described earlier. In particular, it implies that if the learner assumes a uniform prior over H, and if target concepts are in fact drawn from such a distribution when presented to the learner, *then classifying the next instance according to a hypothesis drawn at random from the current version space (according to a uniform distribution), will have expected error at most twice that of the Bayes optimal classifier.* Again, we have an example where a Bayesian analysis of a non-Bayesian algorithm yields insight into the performance of that algorithm.

6.9 NAIVE BAYES CLASSIFIER

One highly practical Bayesian learning method is the naive Bayes learner, often called the *naive Bayes classifier*. In some domains its performance has been shown to be comparable to that of neural network and decision tree learning. This section introduces the naive Bayes classifier; the next section applies it to the practical problem of learning to classify natural language text documents.

The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function $f(x)$ can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values $\langle a_1, a_2 \ldots a_n \rangle$. The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value, v_{MAP}, given the attribute values $\langle a_1, a_2 \ldots a_n \rangle$ that describe the instance.

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

We can use Bayes theorem to rewrite this expression as

$$\begin{aligned} v_{MAP} &= \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)} \\ &= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j) \end{aligned} \tag{6.19}$$

Now we could attempt to estimate the two terms in Equation (6.19) based on the training data. It is easy to estimate each of the $P(v_j)$ simply by counting the frequency with which each target value v_j occurs in the training data. However, estimating the different $P(a_1, a_2 \ldots a_n | v_j)$ terms in this fashion is not feasible unless we have a very, very large set of training data. The problem is that the number of these terms is equal to the number of possible instances times the number of possible target values. Therefore, we need to see every instance in the instance space many times in order to obtain reliable estimates.

The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction $a_1, a_2 \ldots a_n$ is just the product of the probabilities for the individual attributes: $P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$. Substituting this into Equation (6.19), we have the approach used by the naive Bayes classifier.

Naive Bayes classifier:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j) \tag{6.20}$$

where v_{NB} denotes the target value output by the naive Bayes classifier. Notice that in a naive Bayes classifier the number of distinct $P(a_i | v_j)$ terms that must

be estimated from the training data is just the number of distinct attribute values times the number of distinct target values—a much smaller number than if we were to estimate the $P(a_1, a_2 \ldots a_n|v_j)$ terms as first contemplated.

To summarize, the naive Bayes learning method involves a learning step in which the various $P(v_j)$ and $P(a_i|v_j)$ terms are estimated, based on their frequencies over the training data. The set of these estimates corresponds to the learned hypothesis. This hypothesis is then used to classify each new instance by applying the rule in Equation (6.20). Whenever the naive Bayes assumption of conditional independence is satisfied, this naive Bayes classification v_{NB} is identical to the MAP classification.

One interesting difference between the naive Bayes learning method and other learning methods we have considered is that there is no explicit search through the space of possible hypotheses (in this case, the space of possible hypotheses is the space of possible values that can be assigned to the various $P(v_j)$ and $P(a_i|v_j)$ terms). Instead, the hypothesis is formed without searching, simply by counting the frequency of various data combinations within the training examples.

6.9.1 An Illustrative Example

Let us apply the naive Bayes classifier to a concept learning problem we considered during our discussion of decision tree learning: classifying days according to whether someone will play tennis. Table 3.2 from Chapter 3 provides a set of 14 training examples of the target concept *PlayTennis*, where each day is described by the attributes *Outlook*, *Temperature*, *Humidity*, and *Wind*. Here we use the naive Bayes classifier and the training data from this table to classify the following novel instance:

$$\langle Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong \rangle$$

Our task is to predict the target value (*yes* or *no*) of the target concept *PlayTennis* for this new instance. Instantiating Equation (6.20) to fit the current task, the target value v_{NB} is given by

$$\begin{aligned} v_{NB} &= \underset{v_j \in \{yes, no\}}{\operatorname{argmax}} P(v_j) \textstyle\prod_i P(a_i|v_j) \\ &= \underset{v_j \in \{yes, no\}}{\operatorname{argmax}} P(v_j) \quad P(Outlook = sunny|v_j) P(Temperature = cool|v_j) \\ &\qquad P(Humidity = high|v_j) P(Wind = strong|v_j) \end{aligned} \tag{6.21}$$

Notice in the final expression that a_i has been instantiated using the particular attribute values of the new instance. To calculate v_{NB} we now require 10 probabilities that can be estimated from the training data. First, the probabilities of the different target values can easily be estimated based on their frequencies over the 14 training examples

$$P(PlayTennis = yes) = 9/14 = .64$$

$$P(PlayTennis = no) = 5/14 = .36$$

Similarly, we can estimate the conditional probabilities. For example, those for $Wind = strong$ are

$$P(Wind = strong|PlayTennis = yes) = 3/9 = .33$$
$$P(Wind = strong|PlayTennis = no) = 3/5 = .60$$

Using these probability estimates and similar estimates for the remaining attribute values, we calculate v_{NB} according to Equation (6.21) as follows (now omitting attribute names for brevity)

$$P(yes)\ P(sunny|yes)\ P(cool|yes)\ P(high|yes)\ P(strong|yes) = .0053$$
$$P(no)\ P(sunny|no)\ P(cool|no)\ P(high|no)\ P(strong|no) = .0206$$

Thus, the naive Bayes classifier assigns the target value $PlayTennis = no$ to this new instance, based on the probability estimates learned from the training data. Furthermore, by normalizing the above quantities to sum to one we can calculate the conditional probability that the target value is *no*, given the observed attribute values. For the current example, this probability is $\frac{.0206}{.0206+.0053} = .795$.

6.9.1.1 ESTIMATING PROBABILITIES

Up to this point we have estimated probabilities by the fraction of times the event is observed to occur over the total number of opportunities. For example, in the above case we estimated $P(Wind = strong|PlayTennis = no)$ by the fraction $\frac{n_c}{n}$ where $n = 5$ is the total number of training examples for which $PlayTennis = no$, and $n_c = 3$ is the number of these for which $Wind = strong$.

While this observed fraction provides a good estimate of the probability in many cases, it provides poor estimates when n_c is very small. To see the difficulty, imagine that, in fact, the value of $P(Wind = strong|PlayTennis = no)$ is .08 and that we have a sample containing only 5 examples for which $PlayTennis = no$. Then the most probable value for n_c is 0. This raises two difficulties. First, $\frac{n_c}{n}$ produces a biased underestimate of the probability. Second, when this probability estimate is zero, this probability term will dominate the Bayes classifier if the future query contains $Wind = strong$. The reason is that the quantity calculated in Equation (6.20) requires multiplying all the other probability terms by this zero value.

To avoid this difficulty we can adopt a Bayesian approach to estimating the probability, using the m-estimate defined as follows.

m-estimate of probability:

$$\frac{n_c + mp}{n + m} \tag{6.22}$$

Here, n_c and n are defined as before, p is our prior estimate of the probability we wish to determine, and m is a constant called the *equivalent sample size*, which determines how heavily to weight p relative to the observed data. A typical method for choosing p in the absence of other information is to assume uniform

priors; that is, if an attribute has k possible values we set $p = \frac{1}{k}$. For example, in estimating $P(Wind = strong | PlayTennis = no)$ we note the attribute $Wind$ has two possible values, so uniform priors would correspond to choosing $p = .5$. Note that if m is zero, the m-estimate is equivalent to the simple fraction $\frac{n_c}{n}$. If both n and m are nonzero, then the observed fraction $\frac{n_c}{n}$ and prior p will be combined according to the weight m. The reason m is called the equivalent sample size is that Equation (6.22) can be interpreted as augmenting the n actual observations by an additional m virtual samples distributed according to p.

6.10 AN EXAMPLE: LEARNING TO CLASSIFY TEXT

To illustrate the practical importance of Bayesian learning methods, consider learning problems in which the instances are text documents. For example, we might wish to learn the target concept "electronic news articles that I find interesting," or "pages on the World Wide Web that discuss machine learning topics." In both cases, if a computer could learn the target concept accurately, it could automatically filter the large volume of online text documents to present only the most relevant documents to the user.

We present here a general algorithm for learning to classify text, based on the naive Bayes classifier. Interestingly, probabilistic approaches such as the one described here are among the most effective algorithms currently known for learning to classify text documents. Examples of such systems are described by Lewis (1991), Lang (1995), and Joachims (1996).

The naive Bayes algorithm that we shall present applies in the following general setting. Consider an instance space X consisting of all possible *text documents* (i.e., all possible strings of words and punctuation of all possible lengths). We are given training examples of some unknown target function $f(x)$, which can take on any value from some finite set V. The task is to learn from these training examples to predict the target value for subsequent text documents. For illustration, we will consider the target function classifying documents as interesting or uninteresting to a particular person, using the target values *like* and *dislike* to indicate these two classes.

The two main design issues involved in applying the naive Bayes classifier to such text classification problems are first to decide how to represent an arbitrary text document in terms of attribute values, and second to decide how to estimate the probabilities required by the naive Bayes classifier.

Our approach to representing arbitrary text documents is disturbingly simple: Given a text document, such as this paragraph, we define an attribute for each word position in the document and define the value of that attribute to be the English word found in that position. Thus, the current paragraph would be described by 111 attribute values, corresponding to the 111 word positions. The value of the first attribute is the word "our," the value of the second attribute is the word "approach," and so on. Notice that long text documents will require a larger number of attributes than short documents. As we shall see, this will not cause us any trouble.

Given this representation for text documents, we can now apply the naive Bayes classifier. For the sake of concreteness, let us assume we are given a set of 700 training documents that a friend has classified as *dislike* and another 300 she has classified as *like*. We are now given a new document and asked to classify it. Again, for concreteness let us assume the new text document is the preceding paragraph. In this case, we instantiate Equation (6.20) to calculate the naive Bayes classification as

$$\begin{aligned} v_{NB} &= \underset{v_j \in \{like, dislike\}}{\operatorname{argmax}} P(v_j) \prod_{i=1}^{111} P(a_i | v_j) \\ &= \underset{v_j \in \{like, dislike\}}{\operatorname{argmax}} P(v_j)\ P(a_1 = \text{"}our\text{"} | v_j) P(a_2 = \text{"}approach\text{"} | v_j) \\ &\qquad \ldots\ P(a_{111} = \text{"}trouble\text{"} | v_j) \end{aligned}$$

To summarize, the naive Bayes classification v_{NB} is the classification that maximizes the probability of observing the words that were actually found in the document, subject to the usual naive Bayes independence assumption. The independence assumption $P(a_1, \ldots a_{111} | v_j) = \prod_1^{111} P(a_i | v_j)$ states in this setting that the word probabilities for one text position are independent of the words that occur in other positions, given the document classification v_j. Note this assumption is clearly incorrect. For example, the probability of observing the word "learning" in some position may be greater if the preceding word is "machine." Despite the obvious inaccuracy of this independence assumption, we have little choice but to make it—without it, the number of probability terms that must be computed is prohibitive. Fortunately, in practice the naive Bayes learner performs remarkably well in many text classification problems despite the incorrectness of this independence assumption. Domingos and Pazzani (1996) provide an interesting analysis of this fortunate phenomenon.

To calculate v_{NB} using the above expression, we require estimates for the probability terms $P(v_j)$ and $P(a_i = w_k | v_j)$ (here we introduce w_k to indicate the kth word in the English vocabulary). The first of these can easily be estimated based on the fraction of each class in the training data ($P(like) = .3$ and $P(dislike) = .7$ in the current example). As usual, estimating the class conditional probabilities (e.g., $P(a_1 = \text{"}our\text{"} | dislike)$) is more problematic because we must estimate one such probability term for each combination of text position, English word, and target value. Unfortunately, there are approximately 50,000 distinct words in the English vocabulary, 2 possible target values, and 111 text positions in the current example, so we must estimate $2 \cdot 111 \cdot 50,000 \approx 10$ million such terms from the training data.

Fortunately, we can make an additional reasonable assumption that reduces the number of probabilities that must be estimated. In particular, we shall assume the probability of encountering a specific word w_k (e.g., "chocolate") is independent of the specific word position being considered (e.g., a_{23} versus a_{95}). More formally, this amounts to assuming that the attributes are independent and identically distributed, given the target classification; that is, $P(a_i = w_k | v_j) =$

$P(a_m = w_k|v_j)$ for all i, j, k, m. Therefore, we estimate the entire set of probabilities $P(a_1 = w_k|v_j)$, $P(a_2 = w_k|v_j)\ldots$ by the single position-independent probability $P(w_k|v_j)$, which we will use regardless of the word position. The net effect is that we now require only $2 \cdot 50,000$ distinct terms of the form $P(w_k|v_j)$. This is still a large number, but manageable. Notice in cases where training data is limited, the primary advantage of making this assumption is that it increases the number of examples available to estimate each of the required probabilities, thereby increasing the reliability of the estimates.

To complete the design of our learning algorithm, we must still choose a method for estimating the probability terms. We adopt the m-estimate—Equation (6.22)—with uniform priors and with m equal to the size of the word vocabulary. Thus, the estimate for $P(w_k|v_j)$ will be

$$\frac{n_k + 1}{n + |Vocabulary|}$$

where n is the total number of word positions in all training examples whose target value is v_j, n_k is the number of times word w_k is found among these n word positions, and $|Vocabulary|$ is the total number of distinct words (and other tokens) found within the training data.

To summarize, the final algorithm uses a naive Bayes classifier together with the assumption that the probability of word occurrence is independent of position within the text. The final algorithm is shown in Table 6.2. Notice the algorithm is quite simple. During learning, the procedure LEARN_NAIVE_BAYES_TEXT examines all training documents to extract the vocabulary of all words and tokens that appear in the text, then counts their frequencies among the different target classes to obtain the necessary probability estimates. Later, given a new document to be classified, the procedure CLASSIFY_NAIVE_BAYES_TEXT uses these probability estimates to calculate v_{NB} according to Equation (6.20). Note that any words appearing in the new document that were not observed in the training set are simply ignored by CLASSIFY_NAIVE_BAYES_TEXT. Code for this algorithm, as well as training data sets, are available on the World Wide Web at http://www.cs.cmu.edu/~tom/book.html.

6.10.1 Experimental Results

How effective is the learning algorithm of Table 6.2? In one experiment (see Joachims 1996), a minor variant of this algorithm was applied to the problem of classifying usenet news articles. The target classification for an article in this case was the name of the usenet newsgroup in which the article appeared. One can think of the task as creating a newsgroup posting service that learns to assign documents to the appropriate newsgroup. In the experiment described by Joachims (1996), 20 electronic newsgroups were considered (listed in Table 6.3). Then 1,000 articles were collected from each newsgroup, forming a data set of 20,000 documents. The naive Bayes algorithm was then applied using two-thirds of these 20,000 documents as training examples, and performance was measured

LEARN_NAIVE_BAYES_TEXT($Examples$, V)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k|v_j)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

1. *collect all words, punctuation, and other tokens that occur in Examples*
   - $Vocabulary \leftarrow$ the set of all distinct words and other tokens occurring in any text document from $Examples$
2. *calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms*
   - For each target value v_j in V do
     - $docs_j \leftarrow$ the subset of documents from $Examples$ for which the target value is v_j
     - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
     - $Text_j \leftarrow$ a single document created by concatenating all members of $docs_j$
     - $n \leftarrow$ total number of distinct word positions in $Text_j$
     - for each word w_k in $Vocabulary$
       - $n_k \leftarrow$ number of times word w_k occurs in $Text_j$
       - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT(Doc)

Return the estimated target value for the document Doc. a_i denotes the word found in the ith position within Doc.

- $positions \leftarrow$ all word positions in Doc that contain tokens found in $Vocabulary$
- Return v_{NB}, where

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

TABLE 6.2
Naive Bayes algorithms for learning and classifying text. In addition to the usual naive Bayes assumptions, these algorithms assume the probability of a word occurring is independent of its position within the text.

over the remaining third. Given 20 possible newsgroups, we would expect random guessing to achieve a classification accuracy of approximately 5%. The accuracy achieved by the program was 89%. The algorithm used in these experiments was exactly the algorithm of Table 6.2, with one exception: Only a subset of the words occurring in the documents were included as the value of the $Vocabulary$ variable in the algorithm. In particular, the 100 most frequent words were removed (these include words such as "the" and "of"), and any word occurring fewer than three times was also removed. The resulting vocabulary contained approximately 38,500 words.

Similarly impressive results have been achieved by others applying similar statistical learning approaches to text classification. For example, Lang (1995) describes another variant of the naive Bayes algorithm and its application to learning the target concept "usenet articles that I find interesting." He describes the NEWSWEEDER system—a program for reading netnews that allows the user to rate articles as he or she reads them. NEWSWEEDER then uses these rated articles as

comp.graphics	misc.forsale	soc.religion.christian	sci.space
comp.os.ms-windows.misc	rec.autos	talk.politics.guns	sci.crypt
comp.sys.ibm.pc.hardware	rec.motorcycles	talk.politics.mideast	sci.electronics
comp.sys.mac.hardware	rec.sport.baseball	talk.politics.misc	sci.med
comp.windows.x	rec.sport.hockey	talk.religion.misc	
		alt.atheism	

TABLE 6.3
Twenty usenet newsgroups used in the text classification experiment. After training on 667 articles from each newsgroup, a naive Bayes classifier achieved an accuracy of 89% predicting to which newsgroup subsequent articles belonged. Random guessing would produce an accuracy of only 5%.

training examples to learn to predict which subsequent articles will be of interest to the user, so that it can bring these to the user's attention. Lang (1995) reports experiments in which NEWSWEEDER used its learned profile of user interests to suggest the most highly rated new articles each day. By presenting the user with the top 10% of its automatically rated new articles each day, it created a pool of articles containing three to four times as many interesting articles as the general pool of articles read by the user. For example, for one user the fraction of articles rated "interesting" was 16% overall, but was 59% among the articles recommended by NEWSWEEDER.

Several other, non-Bayesian, statistical text learning algorithms are common, many based on similarity metrics initially developed for information retrieval (e.g., see Rocchio 1971; Salton 1991). Additional text learning algorithms are described in Hearst and Hirsh (1996).

6.11 BAYESIAN BELIEF NETWORKS

As discussed in the previous two sections, the naive Bayes classifier makes significant use of the assumption that the values of the attributes $a_1 \ldots a_n$ are conditionally independent given the target value v. This assumption dramatically reduces the complexity of learning the target function. When it is met, the naive Bayes classifier outputs the optimal Bayes classification. However, in many cases this conditional independence assumption is clearly overly restrictive.

A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities. In contrast to the naive Bayes classifier, which assumes that *all* the variables are conditionally independent given the value of the target variable, Bayesian belief networks allow stating conditional independence assumptions that apply to *subsets* of the variables. Thus, Bayesian belief networks provide an intermediate approach that is less constraining than the global assumption of conditional independence made by the naive Bayes classifier, but more tractable than avoiding conditional independence assumptions altogether. Bayesian belief networks are an active focus of current research, and a variety of algorithms have been proposed for learning them and for using them for inference.

In this section we introduce the key concepts and the representation of Bayesian belief networks. More detailed treatments are given by Pearl (1988), Russell and Norvig (1995), Heckerman et al. (1995), and Jensen (1996).

In general, a Bayesian belief network describes the probability distribution over a set of variables. Consider an arbitrary set of random variables $Y_1 \ldots Y_n$, where each variable Y_i can take on the set of possible values $V(Y_i)$. We define the *joint space* of the set of variables Y to be the cross product $V(Y_1) \times V(Y_2) \times \ldots V(Y_n)$. In other words, each item in the joint space corresponds to one of the possible assignments of values to the tuple of variables $\langle Y_1 \ldots Y_n \rangle$. The probability distribution over this joint space is called the *joint probability distribution*. The joint probability distribution specifies the probability for each of the possible variable bindings for the tuple $\langle Y_1 \ldots Y_n \rangle$. A Bayesian belief network describes the joint probability distribution for a set of variables.

6.11.1 Conditional Independence

Let us begin our discussion of Bayesian belief networks by defining precisely the notion of conditional independence. Let X, Y, and Z be three discrete-valued random variables. We say that X is *conditionally independent* of Y given Z if the probability distribution governing X is independent of the value of Y given a value for Z; that is, if

$$(\forall x_i, y_j, z_k)\ P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

where $x_i \in V(X)$, $y_j \in V(Y)$, and $z_k \in V(Z)$. We commonly write the above expression in abbreviated form as $P(X|Y, Z) = P(X|Z)$. This definition of conditional independence can be extended to sets of variables as well. We say that the set of variables $X_1 \ldots X_l$ is conditionally independent of the set of variables $Y_1 \ldots Y_m$ given the set of variables $Z_1 \ldots Z_n$ if

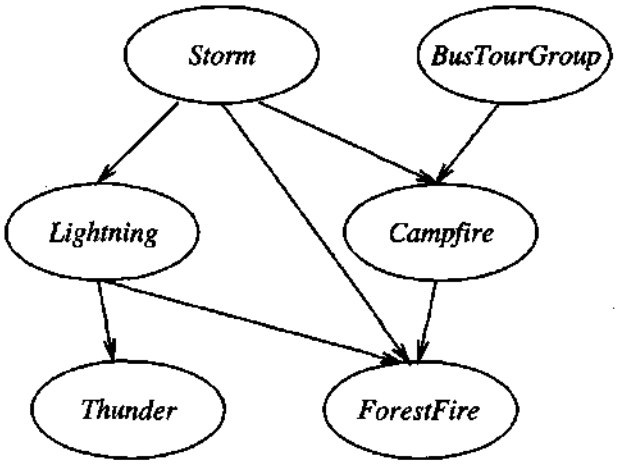
$$P(X_1 \ldots X_l | Y_1 \ldots Y_m, Z_1 \ldots Z_n) = P(X_1 \ldots X_l | Z_1 \ldots Z_n)$$

Note the correspondence between this definition and our use of conditional independence in the definition of the naive Bayes classifier. The naive Bayes classifier assumes that the instance attribute A_1 is conditionally independent of instance attribute A_2 given the target value V. This allows the naive Bayes classifier to calculate $P(A_1, A_2|V)$ in Equation (6.20) as follows

$$P(A_1, A_2|V) = P(A_1|A_2, V)P(A_2|V) \tag{6.23}$$

$$= P(A_1|V)P(A_2|V) \tag{6.24}$$

Equation (6.23) is just the general form of the product rule of probability from Table 6.1. Equation (6.24) follows because if A_1 is conditionally independent of A_2 given V, then by our definition of conditional independence $P(A_1|A_2, V) = P(A_1|V)$.



	S,B	$S,\neg B$	$\neg S,B$	$\neg S,\neg B$
C	0.4	0.1	0.8	0.2
$\neg C$	0.6	0.9	0.2	0.8

Campfire

FIGURE 6.3
A Bayesian belief network. The network on the left represents a set of conditional independence assumptions. In particular, each node is asserted to be conditionally independent of its nondescendants, given its immediate parents. Associated with each node is a conditional probability table, which specifies the conditional distribution for the variable given its immediate parents in the graph. The conditional probability table for the *Campfire* node is shown at the right, where *Campfire* is abbreviated to *C*, *Storm* abbreviated to *S*, and *BusTourGroup* abbreviated to *B*.

6.11.2 Representation

A *Bayesian belief network* (Bayesian network for short) represents the joint probability distribution for a set of variables. For example, the Bayesian network in Figure 6.3 represents the joint probability distribution over the boolean variables *Storm, Lightning, Thunder, ForestFire, Campfire,* and *BusTourGroup.* In general, a Bayesian network represents the joint probability distribution by specifying a set of conditional independence assumptions (represented by a directed acyclic graph), together with sets of local conditional probabilities. Each variable in the joint space is represented by a node in the Bayesian network. For each variable two types of information are specified. First, the network arcs represent the assertion that the variable is conditionally independent of its nondescendants in the network given its immediate predecessors in the network. We say X is a *descendant* of Y if there is a directed path from Y to X. Second, a conditional probability table is given for each variable, describing the probability distribution for that variable given the values of its immediate predecessors. The joint probability for any desired assignment of values $\langle y_1, \ldots, y_n \rangle$ to the tuple of network variables $\langle Y_1 \ldots Y_n \rangle$ can be computed by the formula

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

where $Parents(Y_i)$ denotes the set of immediate predecessors of Y_i in the network. Note the values of $P(y_i | Parents(Y_i))$ are precisely the values stored in the conditional probability table associated with node Y_i.

To illustrate, the Bayesian network in Figure 6.3 represents the joint probability distribution over the boolean variables *Storm, Lightning, Thunder, Forest-*

Fire, *Campfire*, and *BusTourGroup*. Consider the node *Campfire*. The network nodes and arcs represent the assertion that *Campfire* is conditionally independent of its nondescendants *Lightning* and *Thunder*, given its immediate parents *Storm* and *BusTourGroup*. This means that once we know the value of the variables *Storm* and *BusTourGroup*, the variables *Lightning* and *Thunder* provide no additional information about *Campfire*. The right side of the figure shows the conditional probability table associated with the variable *Campfire*. The top left entry in this table, for example, expresses the assertion that

$$P(Campfire = True | Storm = True, BusTourGroup = True) = 0.4$$

Note this table provides only the conditional probabilities of *Campfire* given its parent variables *Storm* and *BusTourGroup*. The set of local conditional probability tables for all the variables, together with the set of conditional independence assumptions described by the network, describe the full joint probability distribution for the network.

One attractive feature of Bayesian belief networks is that they allow a convenient way to represent causal knowledge such as the fact that *Lightning* causes *Thunder*. In the terminology of conditional independence, we express this by stating that *Thunder* is conditionally independent of other variables in the network, given the value of *Lightning*. Note this conditional independence assumption is implied by the arcs in the Bayesian network of Figure 6.3.

6.11.3 Inference

We might wish to use a Bayesian network to infer the value of some target variable (e.g., *ForestFire*) given the observed values of the other variables. Of course, given that we are dealing with random variables it will not generally be correct to assign the target variable a single determined value. What we really wish to infer is the probability distribution for the target variable, which specifies the probability that it will take on each of its possible values given the observed values of the other variables. This inference step can be straightforward if values for all of the other variables in the network are known exactly. In the more general case we may wish to infer the probability distribution for some variable (e.g., *ForestFire*) given observed values for only a subset of the other variables (e.g., *Thunder* and *BusTourGroup* may be the only observed values available). In general, a Bayesian network can be used to compute the probability distribution for any subset of network variables given the values or distributions for any subset of the remaining variables.

Exact inference of probabilities in general for an arbitrary Bayesian network is known to be NP-hard (Cooper 1990). Numerous methods have been proposed for probabilistic inference in Bayesian networks, including exact inference methods and approximate inference methods that sacrifice precision to gain efficiency. For example, Monte Carlo methods provide approximate solutions by randomly sampling the distributions of the unobserved variables (Pradham and Dagum 1996). In theory, even approximate inference of probabilities in Bayesian

networks can be NP-hard (Dagum and Luby 1993). Fortunately, in practice approximate methods have been shown to be useful in many cases. Discussions of inference methods for Bayesian networks are provided by Russell and Norvig (1995) and by Jensen (1996).

6.11.4 Learning Bayesian Belief Networks

Can we devise effective algorithms for learning Bayesian belief networks from training data? This question is a focus of much current research. Several different settings for this learning problem can be considered. First, the network structure might be given in advance, or it might have to be inferred from the training data. Second, all the network variables might be directly observable in each training example, or some might be unobservable.

In the case where the network structure is given in advance and the variables are fully observable in the training examples, learning the conditional probability tables is straightforward. We simply estimate the conditional probability table entries just as we would for a naive Bayes classifier.

In the case where the network structure is given but only some of the variable values are observable in the training data, the learning problem is more difficult. This problem is somewhat analogous to learning the weights for the hidden units in an artificial neural network, where the input and output node values are given but the hidden unit values are left unspecified by the training examples. In fact, Russell et al. (1995) propose a similar gradient ascent procedure that learns the entries in the conditional probability tables. This gradient ascent procedure searches through a space of hypotheses that corresponds to the set of all possible entries for the conditional probability tables. The objective function that is maximized during gradient ascent is the probability $P(D|h)$ of the observed training data D given the hypothesis h. By definition, this corresponds to searching for the maximum likelihood hypothesis for the table entries.

6.11.5 Gradient Ascent Training of Bayesian Networks

The gradient ascent rule given by Russell et al. (1995) maximizes $P(D|h)$ by following the gradient of $\ln P(D|h)$ with respect to the parameters that define the conditional probability tables of the Bayesian network. Let w_{ijk} denote a single entry in one of the conditional probability tables. In particular, let w_{ijk} denote the conditional probability that the network variable Y_i will take on the value y_{ij} given that its immediate parents U_i take on the values given by u_{ik}. For example, if w_{ijk} is the top right entry in the conditional probability table in Figure 6.3, then Y_i is the variable *Campfire*, U_i is the tuple of its parents $\langle Storm, BusTourGroup \rangle$, $y_{ij} = True$, and $u_{ik} = \langle False, False \rangle$. The gradient of $\ln P(D|h)$ is given by the derivatives $\frac{\partial \ln P(D|h)}{\partial w_{ijk}}$ for each of the w_{ijk}. As we show below, each of these derivatives can be calculated as

$$\frac{\partial \ln P(D|h)}{\partial w_{ij}} = \sum_{d \in D} \frac{P(Y_i = y_{ij}, U_i = u_{ik}|d)}{w_{ijk}} \tag{6.25}$$

For example, to calculate the derivative of $\ln P(D|h)$ with respect to the upper-rightmost entry in the table of Figure 6.3 we will have to calculate the quantity $P(Campfire = True, Storm = False, BusTourGroup = False|d)$ for each training example d in D. When these variables are unobservable for the training example d, this required probability can be calculated from the observed variables in d using standard Bayesian network inference. In fact, these required quantities are easily derived from the calculations performed during most Bayesian network inference, so learning can be performed at little additional cost whenever the Bayesian network is used for inference and new evidence is subsequently obtained.

Below we derive Equation (6.25) following Russell et al. (1995). The remainder of this section may be skipped on a first reading without loss of continuity. To simplify notation, in this derivation we will write the abbreviation $P_h(D)$ to represent $P(D|h)$. Thus, our problem is to derive the gradient defined by the set of derivatives $\frac{\partial P_h(D)}{\partial w_{ijk}}$ for all i, j, and k. Assuming the training examples d in the data set D are drawn independently, we write this derivative as

$$\begin{aligned}\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \frac{\partial}{\partial w_{ijk}} \ln \prod_{d \in D} P_h(d) \\ &= \sum_{d \in D} \frac{\partial \ln P_h(d)}{\partial w_{ijk}} \\ &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial P_h(d)}{\partial w_{ijk}}\end{aligned}$$

This last step makes use of the general equality $\frac{\partial \ln f(x)}{\partial x} = \frac{1}{f(x)} \frac{\partial f(x)}{\partial x}$. We can now introduce the values of the variables Y_i and $U_i = Parents(Y_i)$, by summing over their possible values $y_{ij'}$ and $u_{ik'}$.

$$\begin{aligned}\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}, u_{ik'}) \\ &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} \sum_{j',k'} P_h(d|y_{ij'}, u_{ik'}) P_h(y_{ij'}|u_{ik'}) P_h(u_{ik'})\end{aligned}$$

This last step follows from the product rule of probability, Table 6.1. Now consider the rightmost sum in the final expression above. Given that $w_{ijk} \equiv P_h(y_{ij}|u_{ik})$, the only term in this sum for which $\frac{\partial}{\partial w_{ijk}}$ is nonzero is the term for which $j' = j$ and $i' = i$. Therefore

$$\begin{aligned}\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) P_h(y_{ij}|u_{ik}) P_h(u_{ik}) \\ &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) w_{ijk} P_h(u_{ik}) \\ &= \sum_{d \in D} \frac{1}{P_h(d)} P_h(d|y_{ij}, u_{ik}) P_h(u_{ik})\end{aligned}$$

Applying Bayes theorem to rewrite $P_h(d|y_{ij}, u_{ik})$, we have

$$\begin{aligned}\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{P_h(y_{ij}, u_{ik}|d) P_h(d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})} \\ &= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})} \\ &= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{P_h(y_{ij}|u_{ik})} \\ &= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}} \end{aligned} \tag{6.26}$$

Thus, we have derived the gradient given in Equation (6.25). There is one more item that must be considered before we can state the gradient ascent training procedure. In particular, we require that as the weights w_{ijk} are updated they must remain valid probabilities in the interval [0,1]. We also require that the sum $\sum_j w_{ijk}$ remains 1 for all i, k. These constraints can be satisfied by updating weights in a two-step process. First we update each w_{ijk} by gradient ascent

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

where η is a small constant called the learning rate. Second, we renormalize the weights w_{ijk} to assure that the above constraints are satisfied. As discussed by Russell et al., this process will converge to a locally maximum likelihood hypothesis for the conditional probabilities in the Bayesian network.

As in other gradient-based approaches, this algorithm is guaranteed only to find some local optimum solution. An alternative to gradient ascent is the EM algorithm discussed in Section 6.12, which also finds locally maximum likelihood solutions.

6.11.6 Learning the Structure of Bayesian Networks

Learning Bayesian networks when the network structure is not known in advance is also difficult. Cooper and Herskovits (1992) present a Bayesian scoring metric for choosing among alternative networks. They also present a heuristic search algorithm called K2 for learning network structure when the data is fully observable. Like most algorithms for learning the structure of Bayesian networks, K2 performs a greedy search that trades off network complexity for accuracy over the training data. In one experiment K2 was given a set of 3,000 training examples generated at random from a manually constructed Bayesian network containing 37 nodes and 46 arcs. This particular network described potential anesthesia problems in a hospital operating room. In addition to the data, the program was also given an initial ordering over the 37 variables that was consistent with the partial

ordering of variable dependencies in the actual network. The program succeeded in reconstructing the correct Bayesian network structure almost exactly, with the exception of one incorrectly deleted arc and one incorrectly added arc.

Constraint-based approaches to learning Bayesian network structure have also been developed (e.g., Spirtes et al. 1993). These approaches infer independence and dependence relationships from the data, and then use these relationships to construct Bayesian networks. Surveys of current approaches to learning Bayesian networks are provided by Heckerman (1995) and Buntine (1994).

6.12 THE EM ALGORITHM

In many practical learning settings, only a subset of the relevant instance features might be observable. For example, in training or using the Bayesian belief network of Figure 6.3, we might have data where only a subset of the network variables *Storm, Lightning, Thunder, ForestFire, Campfire,* and *BusTourGroup* have been observed. Many approaches have been proposed to handle the problem of learning in the presence of unobserved variables. As we saw in Chapter 3, if some variable is sometimes observed and sometimes not, then we can use the cases for which it has been observed to learn to predict its values when it is not. In this section we describe the EM algorithm (Dempster et al. 1977), a widely used approach to learning in the presence of unobserved variables. The EM algorithm can be used even for variables whose value is never directly observed, provided the general form of the probability distribution governing these variables is known. The EM algorithm has been used to train Bayesian belief networks (see Heckerman 1995) as well as radial basis function networks discussed in Section 8.4. The EM algorithm is also the basis for many unsupervised clustering algorithms (e.g., Cheeseman et al. 1988), and it is the basis for the widely used Baum-Welch forward-backward algorithm for learning Partially Observable Markov Models (Rabiner 1989).

6.12.1 Estimating Means of k Gaussians

The easiest way to introduce the EM algorithm is via an example. Consider a problem in which the data D is a set of instances generated by a probability distribution that is a mixture of k distinct Normal distributions. This problem setting is illustrated in Figure 6.4 for the case where $k = 2$ and where the instances are the points shown along the x axis. Each instance is generated using a two-step process. First, one of the k Normal distributions is selected at random. Second, a single random instance x_i is generated according to this selected distribution. This process is repeated to generate a set of data points as shown in the figure. To simplify our discussion, we consider the special case where the selection of the single Normal distribution at each step is based on choosing each with uniform probability, where each of the k Normal distributions has the same variance σ^2, and where σ^2 is known. The learning task is to output a hypothesis $h = \langle \mu_1, \ldots \mu_k \rangle$ that describes the means of each of the k distributions. We would like to find

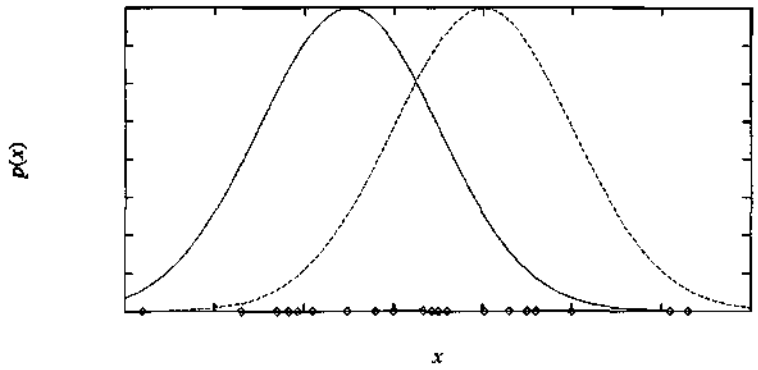


FIGURE 6.4
Instances generated by a mixture of two Normal distributions with identical variance σ. The instances are shown by the points along the x axis. If the means of the Normal distributions are unknown, the EM algorithm can be used to search for their maximum likelihood estimates.

a maximum likelihood hypothesis for these means; that is, a hypothesis h that maximizes $p(D|h)$.

Note it is easy to calculate the maximum likelihood hypothesis for the mean of a single Normal distribution given the observed data instances $x_1, x_2, \ldots, x_m$ drawn from this single distribution. This problem of finding the mean of a single distribution is just a special case of the problem discussed in Section 6.4, Equation (6.6), where we showed that the maximum likelihood hypothesis is the one that minimizes the sum of squared errors over the m training instances. Restating Equation (6.6) using our current notation, we have

$$\mu_{ML} = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{m} (x_i - \mu)^2 \tag{6.27}$$

In this case, the sum of squared errors is minimized by the sample mean

$$\mu_{ML} = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{6.28}$$

Our problem here, however, involves a mixture of k different Normal distributions, and we cannot observe which instances were generated by which distribution. Thus, we have a prototypical example of a problem involving hidden variables. In the example of Figure 6.4, we can think of the full description of each instance as the triple $\langle x_i, z_{i1}, z_{i2} \rangle$, where x_i is the observed value of the ith instance and where z_{i1} and z_{i2} indicate which of the two Normal distributions was used to generate the value x_i. In particular, z_{ij} has the value 1 if x_i was created by the jth Normal distribution and 0 otherwise. Here x_i is the observed variable in the description of the instance, and z_{i1} and z_{i2} are hidden variables. If the values of z_{i1} and z_{i2} were observed, we could use Equation (6.27) to solve for the means μ_1 and μ_2. Because they are not, we will instead use the EM algorithm.

Applied to our k-means problem the EM algorithm searches for a maximum likelihood hypothesis by repeatedly re-estimating the expected values of the hidden variables z_{ij} given its current hypothesis $\langle \mu_1 \ldots \mu_k \rangle$, then recalculating the

maximum likelihood hypothesis using these expected values for the hidden variables. We will first describe this instance of the EM algorithm, and later state the EM algorithm in its general form.

Applied to the problem of estimating the two means for Figure 6.4, the EM algorithm first initializes the hypothesis to $h = \langle \mu_1, \mu_2 \rangle$, where μ_1 and μ_2 are arbitrary initial values. It then iteratively re-estimates h by repeating the following two steps until the procedure converges to a stationary value for h.

Step 1: Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

Step 2: Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in Step 1. Then replace the hypothesis $h = \langle \mu_1, \mu_2 \rangle$ by the new hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$ and iterate.

Let us examine how both of these steps can be implemented in practice. Step 1 must calculate the expected value of each z_{ij}. This $E[z_{ij}]$ is just the probability that instance x_i was generated by the jth Normal distribution

$$E[z_{ij}] = \frac{p(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i | \mu = \mu_n)}$$

$$= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$$

Thus the first step is implemented by substituting the current values $\langle \mu_1, \mu_2 \rangle$ and the observed x_i into the above expression.

In the second step we use the $E[z_{ij}]$ calculated during Step 1 to derive a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$. As we will discuss later, the maximum likelihood hypothesis in this case is given by

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}] \; x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

Note this expression is similar to the sample mean from Equation (6.28) that is used to estimate μ for a single Normal distribution. Our new expression is just the weighted sample mean for μ_j, with each instance weighted by the expectation $E[z_{ij}]$ that it was generated by the jth Normal distribution.

The above algorithm for estimating the means of a mixture of k Normal distributions illustrates the essence of the EM approach: The current hypothesis is used to estimate the unobserved variables, and the expected values of these variables are then used to calculate an improved hypothesis. It can be proved that on each iteration through this loop, the EM algorithm increases the likelihood $P(D|h)$ unless it is at a local maximum. The algorithm thus converges to a local maximum likelihood hypothesis for $\langle \mu_1, \mu_2 \rangle$.

6.12.2 General Statement of EM Algorithm

Above we described an EM algorithm for the problem of estimating means of a mixture of Normal distributions. More generally, the EM algorithm can be applied in many settings where we wish to estimate some set of parameters θ that describe an underlying probability distribution, given only the observed portion of the full data produced by this distribution. In the above two-means example the parameters of interest were $\theta = \langle \mu_1, \mu_2 \rangle$, and the full data were the triples $\langle x_i, z_{i1}, z_{i2} \rangle$ of which only the x_i were observed. In general let $X = \{x_1, \ldots, x_m\}$ denote the observed data in a set of m independently drawn instances, let $Z = \{z_1, \ldots, z_m\}$ denote the unobserved data in these same instances, and let $Y = X \cup Z$ denote the full data. Note the unobserved Z can be treated as a random variable whose probability distribution depends on the unknown parameters θ and on the observed data X. Similarly, Y is a random variable because it is defined in terms of the random variable Z. In the remainder of this section we describe the general form of the EM algorithm. We use h to denote the current hypothesized values of the parameters θ, and h' to denote the revised hypothesis that is estimated on each iteration of the EM algorithm.

The EM algorithm searches for the maximum likelihood hypothesis h' by seeking the h' that maximizes $E[\ln P(Y|h')]$. This expected value is taken over the probability distribution governing Y, which is determined by the unknown parameters θ. Let us consider exactly what this expression signifies. First, $P(Y|h')$ is the likelihood of the full data Y given hypothesis h'. It is reasonable that we wish to find a h' that maximizes some function of this quantity. Second, maximizing the logarithm of this quantity $\ln P(Y|h')$ also maximizes $P(Y|h')$, as we have discussed on several occasions already. Third, we introduce the expected value $E[\ln P(Y|h')]$ because the full data Y is itself a random variable. Given that the full data Y is a combination of the observed data X and unobserved data Z, we must average over the possible values of the unobserved Z, weighting each according to its probability. In other words we take the expected value $E[\ln P(Y|h')]$ over the probability distribution governing the random variable Y. The distribution governing Y is determined by the completely known values for X, plus the distribution governing Z.

What is the probability distribution governing Y? In general we will not know this distribution because it is determined by the parameters θ that we are trying to estimate. Therefore, the EM algorithm uses its current hypothesis h in place of the actual parameters θ to estimate the distribution governing Y. Let us define a function $Q(h'|h)$ that gives $E[\ln P(Y|h')]$ as a function of h', under the assumption that $\theta = h$ and given the observed portion X of the full data Y.

$$Q(h'|h) = E[\ln p(Y|h')|h, X]$$

We write this function Q in the form $Q(h'|h)$ to indicate that it is defined in part by the assumption that the current hypothesis h is equal to θ. In its general form, the EM algorithm repeats the following two steps until convergence:

Step 1: *Estimation (E) step:* Calculate $Q(h'|h)$ using the current hypothesis h and the observed data X to estimate the probability distribution over Y.

$$Q(h'|h) \leftarrow E[\ln P(Y|h')|h, X]$$

Step 2: *Maximization (M) step:* Replace hypothesis h by the hypothesis h' that maximizes this Q function.

$$h \leftarrow \underset{h'}{\operatorname{argmax}}\, Q(h'|h)$$

When the function Q is continuous, the EM algorithm converges to a stationary point of the likelihood function $P(Y|h')$. When this likelihood function has a single maximum, EM will converge to this global maximum likelihood estimate for h'. Otherwise, it is guaranteed only to converge to a local maximum. In this respect, EM shares some of the same limitations as other optimization methods such as gradient descent, line search, and conjugate gradient discussed in Chapter 4.

6.12.3 Derivation of the k Means Algorithm

To illustrate the general EM algorithm, let us use it to derive the algorithm given in Section 6.12.1 for estimating the means of a mixture of k Normal distributions. As discussed above, the k-means problem is to estimate the parameters $\theta = \langle \mu_1 \ldots \mu_k \rangle$ that define the means of the k Normal distributions. We are given the observed data $X = \{\langle x_i \rangle\}$. The hidden variables $Z = \{\langle z_{i1}, \ldots, z_{ik} \rangle\}$ in this case indicate which of the k Normal distributions was used to generate x_i.

To apply EM we must derive an expression for $Q(h|h')$ that applies to our k-means problem. First, let us derive an expression for $\ln p(Y|h')$. Note the probability $p(y_i|h')$ of a single instance $y_i = \langle x_i, z_{i1}, \ldots z_{ik} \rangle$ of the full data can be written

$$p(y_i|h') = p(x_i, z_{i1}, \ldots, z_{ik}|h') = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{k} z_{ij}(x_i - \mu'_j)^2}$$

To verify this note that only one of the z_{ij} can have the value 1, and all others must be 0. Therefore, this expression gives the probability distribution for x_i generated by the selected Normal distribution. Given this probability for a single instance $p(y_i|h')$, the logarithm of the probability $\ln P(Y|h')$ for all m instances in the data is

$$\begin{aligned}
\ln P(Y|h') &= \ln \prod_{i=1}^{m} p(y_i|h') \\
&= \sum_{i=1}^{m} \ln p(y_i|h') \\
&= \sum_{i=1}^{m} \left(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} z_{ij}(x_i - \mu'_j)^2 \right)
\end{aligned}$$

Finally we must take the expected value of this $\ln P(Y|h')$ over the probability distribution governing Y or, equivalently, over the distribution governing the unobserved components z_{ij} of Y. Note the above expression for $\ln P(Y|h')$ is a linear function of these z_{ij}. In general, for any function $f(z)$ that is a *linear* function of z, the following equality holds

$$E[f(z)] = f(E[z])$$

This general fact about linear functions allows us to write

$$E[\ln P(Y|h')] = E\left[\sum_{i=1}^{m}\left(\ln\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} z_{ij}(x_i - \mu'_j)^2\right)\right]$$

$$= \sum_{i=1}^{m}\left(\ln\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2\right)$$

To summarize, the function $Q(h'|h)$ for the k means problem is

$$Q(h'|h) = \sum_{i=1}^{m}\left(\ln\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2\right)$$

where $h' = \langle\mu'_1, \ldots, \mu'_k\rangle$ and where $E[z_{ij}]$ is calculated based on the current hypothesis h and observed data X. As discussed earlier

$$E[z_{ij}] = \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{k} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}} \tag{6.29}$$

Thus, the first (estimation) step of the EM algorithm defines the Q function based on the estimated $E[z_{ij}]$ terms. The second (maximization) step then finds the values $\mu'_1, \ldots, \mu'_k$ that maximize this Q function. In the current case

$$\underset{h'}{\text{argmax}}\, Q(h'|h) = \underset{h'}{\text{argmax}} \sum_{i=1}^{m}\left(\ln\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2\right)$$

$$= \underset{h'}{\text{argmin}} \sum_{i=1}^{m}\sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2 \tag{6.30}$$

Thus, the maximum likelihood hypothesis here minimizes a weighted sum of squared errors, where the contribution of each instance x_i to the error that defines μ'_j is weighted by $E[z_{ij}]$. The quantity given by Equation (6.30) is minimized by setting each μ'_j to the weighted sample mean

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}]\ x_i}{\sum_{i=1}^{m} E[z_{ij}]} \tag{6.31}$$

Note that Equations (6.29) and (6.31) define the two steps in the k-means algorithm described in Section 6.12.1.