estimation bias is a numerical quantity, whereas the inductive bias is a set of assertions.

A second important property of any estimator is its variance. Given a choice among alternative unbiased estimators, it makes sense to choose the one with least variance. By our definition of variance, this choice will yield the smallest expected squared error between the estimate and the true value of the parameter.

To illustrate these concepts, suppose we test a hypothesis and find that it commits $r = 12$ errors on a sample of $n = 40$ randomly drawn test examples. Then an unbiased estimate for $error_{\mathcal{D}}(h)$ is given by $error_S(h) = r/n = 0.3$. The variance in this estimate arises completely from the variance in r, because n is a constant. Because r is Binomially distributed, its variance is given by Equation (5.7) as $np(1-p)$. Unfortunately p is unknown, but we can substitute our estimate r/n for p. This yields an estimated variance in r of $40 \cdot 0.3(1 - 0.3) = 8.4$, or a corresponding standard deviation of $\sqrt{8.4} \approx 2.9$. This implies that the standard deviation in $error_S(h) = r/n$ is approximately $2.9/40 = .07$. To summarize, $error_S(h)$ in this case is observed to be 0.30, with a standard deviation of approximately 0.07. (See Exercise 5.1.)

In general, given r errors in a sample of n independently drawn test examples, the standard deviation for $error_S(h)$ is given by

$$\sigma_{error_S(h)} = \frac{\sigma_r}{n} = \sqrt{\frac{p(1-p)}{n}} \tag{5.8}$$

which can be approximated by substituting $r/n = error_S(h)$ for p

$$\sigma_{error_S(h)} \approx \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}} \tag{5.9}$$

5.3.5 Confidence Intervals

One common way to describe the uncertainty associated with an estimate is to give an interval within which the true value is expected to fall, along with the probability with which it is expected to fall into this interval. Such estimates are called *confidence interval* estimates.

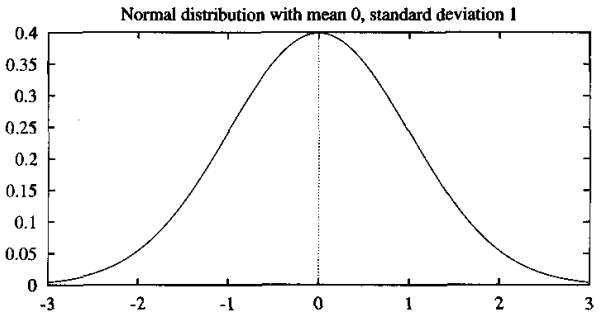
> ***Definition:*** An $N\%$ **confidence interval** for some parameter p is an interval that is expected with probability $N\%$ to contain p.

For example, if we observe $r = 12$ errors in a sample of $n = 40$ independently drawn examples, we can say with approximately 95% probability that the interval 0.30 ± 0.14 contains the true error $error_{\mathcal{D}}(h)$.

How can we derive confidence intervals for $error_{\mathcal{D}}(h)$? The answer lies in the fact that we know the Binomial probability distribution governing the estimator $error_S(h)$. The mean value of this distribution is $error_{\mathcal{D}}(h)$, and the standard deviation is given by Equation (5.9). Therefore, to derive a 95% confidence interval, we need only find the interval centered around the mean value $error_{\mathcal{D}}(h)$,

which is wide enough to contain 95% of the total probability under this distribution. This provides an interval surrounding $error_{\mathcal{D}}(h)$ into which $error_S(h)$ must fall 95% of the time. Equivalently, it provides the size of the interval surrounding $error_S(h)$ into which $error_{\mathcal{D}}(h)$ must fall 95% of the time.

For a given value of N how can we find the size of the interval that contains $N\%$ of the probability mass? Unfortunately, for the Binomial distribution this calculation can be quite tedious. Fortunately, however, an easily calculated and very good approximation can be found in most cases, based on the fact that for sufficiently large sample sizes the Binomial distribution can be closely approximated by the Normal distribution. The Normal distribution, summarized in Table 5.4, is perhaps the most well-studied probability distribution in statistics. As illustrated in Table 5.4, it is a bell-shaped distribution fully specified by its



A Normal distribution (also called a Gaussian distribution) is a bell-shaped distribution defined by the probability density function

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

A Normal distribution is fully determined by two parameters in the above formula: μ and σ.

If the random variable X follows a normal distribution, then:

- The probability that X will fall into the interval (a, b) is given by

$$\int_a^b p(x)dx$$

- The expected, or mean value of X, $E[X]$, is

$$E[X] = \mu$$

- The variance of X, $Var(X)$, is

$$Var(X) = \sigma^2$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sigma$$

The Central Limit Theorem (Section 5.4.1) states that the sum of a large number of independent, identically distributed random variables follows a distribution that is approximately Normal.

TABLE 5.4
The Normal or Gaussian distribution.

mean μ and standard deviation σ. For large n, any Binomial distribution is very closely approximated by a Normal distribution with the same mean and variance.

One reason that we prefer to work with the Normal distribution is that most statistics references give tables specifying the size of the interval about the mean that contains $N\%$ of the probability mass under the Normal distribution. This is precisely the information needed to calculate our $N\%$ confidence interval. In fact, Table 5.1 is such a table. The constant z_N given in Table 5.1 defines the width of the smallest interval about the mean that includes $N\%$ of the total probability mass under the bell-shaped Normal distribution. More precisely, z_N gives half the width of the interval (i.e., the distance from the mean in either direction) measured in standard deviations. Figure 5.1(a) illustrates such an interval for $z_{.80}$.

To summarize, if a random variable Y obeys a Normal distribution with mean μ and standard deviation σ, then the measured random value y of Y will fall into the following interval $N\%$ of the time

$$\mu \pm z_N \sigma \tag{5.10}$$

Equivalently, the mean μ will fall into the following interval $N\%$ of the time

$$y \pm z_N \sigma \tag{5.11}$$

We can easily combine this fact with earlier facts to derive the general expression for $N\%$ confidence intervals for discrete-valued hypotheses given in Equation (5.1). First, we know that $error_S(h)$ follows a Binomial distribution with mean value $error_{\mathcal{D}}(h)$ and standard deviation as given in Equation (5.9). Second, we know that for sufficiently large sample size n, this Binomial distribution is well approximated by a Normal distribution. Third, Equation (5.11) tells us how to find the $N\%$ confidence interval for estimating the mean value of a Normal distribution. Therefore, substituting the mean and standard deviation of $error_S(h)$ into Equation (5.11) yields the expression from Equation (5.1) for $N\%$ confidence

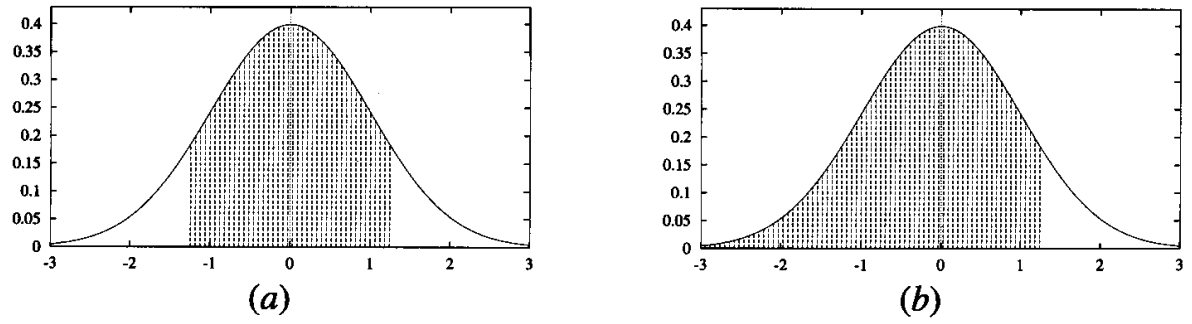


FIGURE 5.1
A Normal distribution with mean 0, standard deviation 1. (a) With 80% confidence, the value of the random variable will lie in the two-sided interval $[-1.28, 1.28]$. Note $z_{.80} = 1.28$. With 10% confidence it will lie to the right of this interval, and with 10% confidence it will lie to the left. (b) With 90% confidence, it will lie in the one-sided interval $[-\infty, 1.28]$.

intervals for discrete-valued hypotheses

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Recall that two approximations were involved in deriving this expression, namely:

1. in estimating the standard deviation σ of $error_S(h)$, we have approximated $error_{\mathcal{D}}(h)$ by $error_S(h)$ [i.e., in going from Equation (5.8) to (5.9)], and
2. the Binomial distribution has been approximated by the Normal distribution.

The common rule of thumb in statistics is that these two approximations are very good as long as $n \geq 30$, or when $np(1-p) \geq 5$. For smaller values of n it is wise to use a table giving exact values for the Binomial distribution.

5.3.6 Two-Sided and One-Sided Bounds

Notice that the above confidence interval is a *two-sided* bound; that is, it bounds the estimated quantity from above and from below. In some cases, we will be interested only in a *one-sided* bound. For example, we might be interested in the question "What is the probability that $error_{\mathcal{D}}(h)$ is at most U?" This kind of one-sided question is natural when we are only interested in bounding the maximum error of h and do not mind if the true error is much smaller than estimated.

There is an easy modification to the above procedure for finding such one-sided error bounds. It follows from the fact that the Normal distribution is symmetric about its mean. Because of this fact, any two-sided confidence interval based on a Normal distribution can be converted to a corresponding one-sided interval with twice the confidence (see Figure 5.1(*b*)). That is, a $100(1-\alpha)\%$ confidence interval with lower bound L and upper bound U implies a $100(1-\alpha/2)\%$ confidence interval with lower bound L and no upper bound. It also implies a $100(1-\alpha/2)\%$ confidence interval with upper bound U and no lower bound. Here α corresponds to the probability that the correct value lies outside the stated interval. In other words, α is the probability that the value will fall into the *unshaded* region in Figure 5.1(*a*), and $\alpha/2$ is the probability that it will fall into the unshaded region in Figure 5.1(*b*).

To illustrate, consider again the example in which h commits $r = 12$ errors over a sample of $n = 40$ independently drawn examples. As discussed above, this leads to a (two-sided) 95% confidence interval of 0.30 ± 0.14. In this case, $100(1-\alpha) = 95\%$, so $\alpha = 0.05$. Thus, we can apply the above rule to say with $100(1-\alpha/2) = 97.5\%$ confidence that $error_{\mathcal{D}}(h)$ is at most $0.30 + 0.14 = 0.44$, making no assertion about the lower bound on $error_{\mathcal{D}}(h)$. Thus, we have a one-sided error bound on $error_{\mathcal{D}}(h)$ with double the confidence that we had in the corresponding two-sided bound (see Exercise 5.3).

5.4 A GENERAL APPROACH FOR DERIVING CONFIDENCE INTERVALS

The previous section described in detail how to derive confidence interval estimates for one particular case: estimating $error_{\mathcal{D}}(h)$ for a discrete-valued hypothesis h, based on a sample of n independently drawn instances. The approach described there illustrates a general approach followed in many estimation problems. In particular, we can see this as a problem of estimating the mean (expected value) of a population based on the mean of a randomly drawn sample of size n. The general process includes the following steps:

1. Identify the underlying population parameter p to be estimated, for example, $error_{\mathcal{D}}(h)$.
2. Define the estimator Y (e.g., $error_S(h)$). It is desirable to choose a minimum-variance, unbiased estimator.
3. Determine the probability distribution $\mathcal{D}_Y$ that governs the estimator Y, including its mean and variance.
4. Determine the $N\%$ confidence interval by finding thresholds L and U such that $N\%$ of the mass in the probability distribution $\mathcal{D}_Y$ falls between L and U.

In later sections of this chapter we apply this general approach to several other estimation problems common in machine learning. First, however, let us discuss a fundamental result from estimation theory called the *Central Limit Theorem.*

5.4.1 Central Limit Theorem

One essential fact that simplifies attempts to derive confidence intervals is the Central Limit Theorem. Consider again our general setting, in which we observe the values of n independently drawn random variables $Y_1 \ldots Y_n$ that obey the same unknown underlying probability distribution (e.g., n tosses of the same coin). Let μ denote the mean of the unknown distribution governing each of the Y_i and let σ denote the standard deviation. We say that these variables Y_i are *independent, identically distributed* random variables, because they describe independent experiments, each obeying the same underlying probability distribution. In an attempt to estimate the mean μ of the distribution governing the Y_i, we calculate the sample mean $\bar{Y}_n \equiv \frac{1}{n}\sum_{i=1}^{n} Y_i$ (e.g., the fraction of heads among the n coin tosses). The Central Limit Theorem states that the probability distribution governing $\bar{Y}_n$ approaches a Normal distribution as $n \to \infty$, *regardless of the distribution that governs the underlying random variables* Y_i. Furthermore, the mean of the distribution governing $\bar{Y}_n$ approaches μ and the standard deviation approaches $\frac{\sigma}{\sqrt{n}}$. More precisely,

Theorem 5.1. Central Limit Theorem. Consider a set of independent, identically distributed random variables $Y_1 \ldots Y_n$ governed by an arbitrary probability distribution with mean μ and finite variance σ^2. Define the sample mean, $\bar{Y}_n \equiv \frac{1}{n}\sum_{i=1}^{n} Y_i$.

Then as $n \to \infty$, the distribution governing

$$\frac{\bar{Y}_n - \mu}{\frac{\sigma}{\sqrt{n}}}$$

approaches a Normal distribution, with zero mean and standard deviation equal to 1.

This is a quite surprising fact, because it states that we know the form of the distribution that governs the sample mean $\bar{Y}$ even when we do not know the form of the underlying distribution that governs the individual Y_i that are being observed! Furthermore, the Central Limit Theorem describes how the mean and variance of $\bar{Y}$ can be used to determine the mean and variance of the individual Y_i.

The Central Limit Theorem is a very useful fact, because it implies that whenever we define an estimator that is the mean of some sample (e.g., $error_S(h)$ is the mean error), the distribution governing this estimator can be approximated by a Normal distribution for sufficiently large n. If we also know the variance for this (approximately) Normal distribution, then we can use Equation (5.11) to compute confidence intervals. A common rule of thumb is that we can use the Normal approximation when $n \geq 30$. Recall that in the preceding section we used such a Normal distribution to approximate the Binomial distribution that more precisely describes $error_S(h)$.

5.5 DIFFERENCE IN ERROR OF TWO HYPOTHESES

Consider the case where we have two hypotheses h_1 and h_2 for some discrete-valued target function. Hypothesis h_1 has been tested on a sample S_1 containing n_1 randomly drawn examples, and h_2 has been tested on an independent sample S_2 containing n_2 examples drawn from the same distribution. Suppose we wish to estimate the difference d between the true errors of these two hypotheses.

$$d \equiv error_{\mathcal{D}}(h_1) - error_{\mathcal{D}}(h_2)$$

We will use the generic four-step procedure described at the beginning of Section 5.4 to derive a confidence interval estimate for d. Having identified d as the parameter to be estimated, we next define an estimator. The obvious choice for an estimator in this case is the difference between the sample errors, which we denote by $\hat{d}$

$$\hat{d} \equiv error_{S_1}(h_1) - error_{S_2}(h_2)$$

Although we will not prove it here, it can be shown that $\hat{d}$ gives an unbiased estimate of d; that is $E[\hat{d}\,] = d$.

What is the probability distribution governing the random variable $\hat{d}$? From earlier sections, we know that for large n_1 and n_2 (e.g., both ≥ 30), both $error_{S_1}(h_1)$ and $error_{S_2}(h_2)$ follow distributions that are approximately Normal. Because the difference of two Normal distributions is also a Normal distribution, $\hat{d}$ will also

follow a distribution that is approximately Normal, with mean d. It can also be shown that the variance of this distribution is the sum of the variances of $error_{S_1}(h_1)$ and $error_{S_2}(h_2)$. Using Equation (5.9) to obtain the approximate variance of each of these distributions, we have

$$\sigma_{\hat{d}}^2 \approx \frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2} \tag{5.12}$$

Now that we have determined the probability distribution that governs the estimator $\hat{d}$, it is straightforward to derive confidence intervals that characterize the likely error in employing $\hat{d}$ to estimate d. For a random variable $\hat{d}$ obeying a Normal distribution with mean d and variance σ^2, the N% confidence interval estimate for d is $\hat{d} \pm z_N \sigma$. Using the approximate variance $\sigma_{\hat{d}}^2$ given above, this approximate $N\%$ confidence interval estimate for d is

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2}} \tag{5.13}$$

where z_N is the same constant described in Table 5.1. The above expression gives the general two-sided confidence interval for estimating the difference between errors of two hypotheses. In some situations we might be interested in one-sided bounds—either bounding the largest possible difference in errors or the smallest, with some confidence level. One-sided confidence intervals can be obtained by modifying the above expression as described in Section 5.3.6.

Although the above analysis considers the case in which h_1 and h_2 are tested on independent data samples, it is often acceptable to use the confidence interval seen in Equation (5.13) in the setting where h_1 and h_2 are tested on a single sample S (where S is still independent of h_1 and h_2). In this later case, we redefine $\hat{d}$ as

$$\hat{d} \equiv error_S(h_1) - error_S(h_2)$$

The variance in this new $\hat{d}$ will usually be smaller than the variance given by Equation (5.12), when we set S_1 and S_2 to S. This is because using a single sample S eliminates the variance due to random differences in the compositions of S_1 and S_2. In this case, the confidence interval given by Equation (5.13) will generally be an overly conservative, but still correct, interval.

5.5.1 Hypothesis Testing

In some cases we are interested in the probability that some specific conjecture is true, rather than in confidence intervals for some parameter. Suppose, for example, that we are interested in the question "what is the probability that $error_{\mathcal{D}}(h_1) > error_{\mathcal{D}}(h_2)$?" Following the setting in the previous section, suppose we measure the sample errors for h_1 and h_2 using two independent samples S_1 and S_2 of size 100 and find that $error_{S_1}(h_1) = .30$ and $error_{S_2}(h_2) = .20$, hence the observed difference is $\hat{d} = .10$. Of course, due to random variation in the data sample,

we might observe this difference in the sample errors even when $error_{\mathcal{D}}(h_1) \leq error_{\mathcal{D}}(h_2)$. What is the probability that $error_{\mathcal{D}}(h_1) > error_{\mathcal{D}}(h_2)$, given the observed difference in sample errors $\hat{d} = .10$ in this case? Equivalently, what is the probability that $d > 0$, given that we observed $\hat{d} = .10$?

Note the probability $\Pr(d > 0)$ is equal to the probability that $\hat{d}$ has not overestimated d by more than .10. Put another way, this is the probability that $\hat{d}$ falls into the one-sided interval $\hat{d} < d + .10$. Since d is the mean of the distribution governing $\hat{d}$, we can equivalently express this one-sided interval as $\hat{d} < \mu_{\hat{d}} + .10$.

To summarize, the probability $\Pr(d > 0)$ equals the probability that $\hat{d}$ falls into the one-sided interval $\hat{d} < \mu_{\hat{d}} + .10$. Since we already calculated the approximate distribution governing $\hat{d}$ in the previous section, we can determine the probability that $\hat{d}$ falls into this one-sided interval by calculating the probability mass of the $\hat{d}$ distribution within this interval.

Let us begin this calculation by re-expressing the interval $\hat{d} < \mu_{\hat{d}} + .10$ in terms of the number of standard deviations it allows deviating from the mean. Using Equation (5.12) we find that $\sigma_{\hat{d}} \approx .061$, so we can re-express the interval as approximately

$$\hat{d} < \mu_{\hat{d}} + 1.64\sigma_{\hat{d}}$$

What is the confidence level associated with this one-sided interval for a Normal distribution? Consulting Table 5.1, we find that 1.64 standard deviations about the mean corresponds to a two-sided interval with confidence level 90%. Therefore, the one-sided interval will have an associated confidence level of 95%.

Therefore, given the observed $\hat{d} = .10$, the probability that $error_{\mathcal{D}}(h_1) > error_{\mathcal{D}}(h_2)$ is approximately .95. In the terminology of the statistics literature, we say that we accept the hypothesis that "$error_{\mathcal{D}}(h_1) > error_{\mathcal{D}}(h_2)$" with confidence 0.95. Alternatively, we may state that we reject the opposite hypothesis (often called the null hypothesis) at a $(1 - 0.95) = .05$ level of significance.

5.6 COMPARING LEARNING ALGORITHMS

Often we are interested in comparing the performance of two learning algorithms L_A and L_B, rather than two specific hypotheses. What is an appropriate test for comparing learning algorithms, and how can we determine whether an observed difference between the algorithms is statistically significant? Although there is active debate within the machine-learning research community regarding the best method for comparison, we present here one reasonable approach. A discussion of alternative methods is given by Dietterich (1996).

As usual, we begin by specifying the parameter we wish to estimate. Suppose we wish to determine which of L_A and L_B is the better learning method on average for learning some particular target function f. A reasonable way to define "on average" is to consider the relative performance of these two algorithms averaged over all the training sets of size n that might be drawn from the underlying instance distribution $\mathcal{D}$. In other words, we wish to estimate the expected value

of the difference in their errors

$$\underset{S \subset \mathcal{D}}{E}[error_{\mathcal{D}}(L_A(S)) - error_{\mathcal{D}}(L_B(S))] \tag{5.14}$$

where $L(S)$ denotes the hypothesis output by learning method L when given the sample S of training data and where the subscript $S \subset \mathcal{D}$ indicates that the expected value is taken over samples S drawn according to the underlying instance distribution $\mathcal{D}$. The above expression describes the expected value of the difference in errors between learning methods L_A and L_B.

Of course in practice we have only a limited sample D_0 of data when comparing learning methods. In such cases, one obvious approach to estimating the above quantity is to divide D_0 into a training set S_0 and a disjoint test set T_0. The training data can be used to train both L_A and L_B, and the test data can be used to compare the accuracy of the two learned hypotheses. In other words, we measure the quantity

$$error_{T_0}(L_A(S_0)) - error_{T_0}(L_B(S_0)) \tag{5.15}$$

Notice two key differences between this estimator and the quantity in Equation (5.14). First, we are using $error_{T_0}(h)$ to approximate $error_{\mathcal{D}}(h)$. Second, we are only measuring the difference in errors for one training set S_0 rather than taking the expected value of this difference over all samples S that might be drawn from the distribution $\mathcal{D}$.

One way to improve on the estimator given by Equation (5.15) is to repeatedly partition the data D_0 into disjoint training and test sets and to take the mean of the test set errors for these different experiments. This leads to the procedure shown in Table 5.5 for estimating the difference between errors of two learning methods, based on a fixed sample D_0 of available data. This procedure first partitions the data into k disjoint subsets of equal size, where this size is at least 30. It then trains and tests the learning algorithms k times, using each of the k subsets in turn as the test set, and using all remaining data as the training set. In this way, the learning algorithms are tested on k independent test sets, and the mean difference in errors $\bar{\delta}$ is returned as an estimate of the difference between the two learning algorithms.

The quantity $\bar{\delta}$ returned by the procedure of Table 5.5 can be taken as an estimate of the desired quantity from Equation 5.14. More appropriately, we can view $\bar{\delta}$ as an estimate of the quantity

$$\underset{S \subset D_0}{E}[error_{\mathcal{D}}(L_A(S)) - error_{\mathcal{D}}(L_B(S))] \tag{5.16}$$

where S represents a random sample of size $\frac{k-1}{k}|D_0|$ drawn uniformly from D_0. The only difference between this expression and our original expression in Equation (5.14) is that this new expression takes the expected value over subsets of the available data D_0, rather than over subsets drawn from the full instance distribution $\mathcal{D}$.

1. Partition the available data D_0 into k disjoint subsets $T_1, T_2, \ldots, T_k$ of equal size, where this size is at least 30.
2. For i from 1 to k, do
   use T_i for the test set, and the remaining data for training set S_i
   - $S_i \leftarrow \{D_0 - T_i\}$
   - $h_A \leftarrow L_A(S_i)$
   - $h_B \leftarrow L_B(S_i)$
   - $\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$
3. Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i \tag{T5.1}$$

TABLE 5.5
A procedure to estimate the difference in error between two learning methods L_A and L_B. Approximate confidence intervals for this estimate are given in the text.

The approximate $N\%$ confidence interval for estimating the quantity in Equation (5.16) using $\bar{\delta}$ is given by

$$\bar{\delta} \pm t_{N,k-1}\, s_{\bar{\delta}} \tag{5.17}$$

where $t_{N,k-1}$ is a constant that plays a role analogous to that of z_N in our earlier confidence interval expressions, and where $s_{\bar{\delta}}$ is an estimate of the standard deviation of the distribution governing $\bar{\delta}$. In particular, $s_{\bar{\delta}}$ is defined as

$$s_{\bar{\delta}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \bar{\delta})^2} \tag{5.18}$$

Notice the constant $t_{N,k-1}$ in Equation (5.17) has two subscripts. The first specifies the desired confidence level, as it did for our earlier constant z_N. The second parameter, called the number of *degrees of freedom* and usually denoted by ν, is related to the number of independent random events that go into producing the value for the random variable $\bar{\delta}$. In the current setting, the number of degrees of freedom is $k - 1$. Selected values for the parameter t are given in Table 5.6. Notice that as $k \rightarrow \infty$, the value of $t_{N,k-1}$ approaches the constant z_N.

Note the procedure described here for comparing two learning methods involves testing the two learned hypotheses on identical test sets. This contrasts with the method described in Section 5.5 for comparing hypotheses that have been evaluated using two independent test sets. Tests where the hypotheses are evaluated over identical samples are called *paired tests*. Paired tests typically produce tighter confidence intervals because any differences in observed errors in a paired test are due to differences between the hypotheses. In contrast, when the hypotheses are tested on separate data samples, differences in the two sample errors might be partially attributable to differences in the makeup of the two samples.

	Confidence level N			
	90%	95%	98%	99%
$\nu = 2$	2.92	4.30	6.96	9.92
$\nu = 5$	2.02	2.57	3.36	4.03
$\nu = 10$	1.81	2.23	2.76	3.17
$\nu = 20$	1.72	2.09	2.53	2.84
$\nu = 30$	1.70	2.04	2.46	2.75
$\nu = 120$	1.66	1.98	2.36	2.62
$\nu = \infty$	1.64	1.96	2.33	2.58

TABLE 5.6
Values of $t_{N,\nu}$ for two-sided confidence intervals. As $\nu \to \infty$, $t_{N,\nu}$ approaches z_N.

5.6.1 Paired t Tests

Above we described one procedure for comparing two learning methods given a fixed set of data. This section discusses the statistical justification for this procedure, and for the confidence interval defined by Equations (5.17) and (5.18). It can be skipped or skimmed on a first reading without loss of continuity.

The best way to understand the justification for the confidence interval estimate given by Equation (5.17) is to consider the following estimation problem:

- We are given the observed values of a set of independent, identically distributed random variables $Y_1, Y_2, \ldots, Y_k$.
- We wish to estimate the mean μ of the probability distribution governing these Y_i.
- The estimator we will use is the sample mean $\bar{Y}$

$$\bar{Y} \equiv \frac{1}{k} \sum_{i=1}^{k} Y_i$$

This problem of estimating the distribution mean μ based on the sample mean $\bar{Y}$ is quite general. For example, it covers the problem discussed earlier of using $error_S(h)$ to estimate $error_{\mathcal{D}}(h)$. (In that problem, the Y_i are 1 or 0 to indicate whether h commits an error on an individual example from S, and $error_{\mathcal{D}}(h)$ is the mean μ of the underlying distribution.) The t test, described by Equations (5.17) and (5.18), applies to a special case of this problem—the case in which the individual Y_i follow a Normal distribution.

Now consider the following idealization of the method in Table 5.5 for comparing learning methods. Assume that instead of having a fixed sample of data D_0, we can request new training examples drawn according to the underlying instance distribution. In particular, in this idealized method we modify the procedure of Table 5.5 so that on each iteration through the loop it generates a new random training set S_i and new random test set T_i by drawing from this underlying instance distribution instead of drawing from the fixed sample D_0. This idealized method

perfectly fits the form of the above estimation problem. In particular, the δ_i measured by the procedure now correspond to the independent, identically distributed random variables Y_i. The mean μ of their distribution corresponds to the expected difference in error between the two learning methods [i.e., Equation (5.14)]. The sample mean $\bar{Y}$ is the quantity $\bar{\delta}$ computed by this idealized version of the method. We wish to answer the question "how good an estimate of μ is provided by $\bar{\delta}$?"

First, note that the size of the test sets T_i has been chosen to contain at least 30 examples. Because of this, the individual δ_i will each follow an approximately Normal distribution (due to the Central Limit Theorem). Hence, we have a special case in which the Y_i are governed by an approximately Normal distribution. It can be shown in general that when the individual Y_i each follow a Normal distribution, then the sample mean $\bar{Y}$ follows a Normal distribution as well. Given that $\bar{Y}$ is Normally distributed, we might consider using the earlier expression for confidence intervals (Equation [5.11]) that applies to estimators governed by Normal distributions. Unfortunately, that equation requires that we know the standard deviation of this distribution, which we do not.

The t test applies to precisely these situations, in which the task is to estimate the sample mean of a collection of independent, identically and Normally distributed random variables. In this case, we can use the confidence interval given by Equations (5.17) and (5.18), which can be restated using our current notation as

$$\mu = \bar{Y} \pm t_{N,k-1}\, s_{\bar{Y}}$$

where $s_{\bar{Y}}$ is the estimated standard deviation of the sample mean

$$s_{\bar{Y}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (Y_i - \bar{Y})^2}$$

and where $t_{N,k-1}$ is a constant analogous to our earlier z_N. In fact, the constant $t_{N,k-1}$ characterizes the area under a probability distribution known as the t distribution, just as the constant z_N characterizes the area under a Normal distribution. The t distribution is a bell-shaped distribution similar to the Normal distribution, but wider and shorter to reflect the greater variance introduced by using $s_{\bar{Y}}$ to approximate the true standard deviation $\sigma_{\bar{Y}}$. The t distribution approaches the Normal distribution (and therefore $t_{N,k-1}$ approaches z_N) as k approaches infinity. This is intuitively satisfying because we expect $s_{\bar{Y}}$ to converge toward the true standard deviation $\sigma_{\bar{Y}}$ as the sample size k grows, and because we can use z_N when the standard deviation is known exactly.

5.6.2 Practical Considerations

Note the above discussion justifies the use of the confidence interval estimate given by Equation (5.17) in the case where we wish to use the sample mean $\bar{Y}$ to estimate the mean of a sample containing k independent, identically and Normally distributed random variables. This fits the idealized method described

above, in which we assume unlimited access to examples of the target function. In practice, given a limited set of data D_0 and the more practical method described by Table 5.5, this justification does not strictly apply. In practice, the problem is that the only way to generate new δ_i is to resample D_0, dividing it into training and test sets in different ways. The δ_i are not independent of one another in this case, because they are based on overlapping sets of training examples drawn from the limited subset D_0 of data, rather than from the full distribution $\mathcal{D}$.

When only a limited sample of data D_0 is available, several methods can be used to resample D_0. Table 5.5 describes a k-fold method in which D_0 is partitioned into k disjoint, equal-sized subsets. In this k-fold approach, each example from D_0 is used exactly once in a test set, and $k - 1$ times in a training set. A second popular approach is to randomly choose a test set of at least 30 examples from D_0, use the remaining examples for training, then repeat this process as many times as desired. This randomized method has the advantage that it can be repeated an indefinite number of times, to shrink the confidence interval to the desired width. In contrast, the k-fold method is limited by the total number of examples, by the use of each example only once in a test set, and by our desire to use samples of size at least 30. However, the randomized method has the disadvantage that the test sets no longer qualify as being independently drawn with respect to the underlying instance distribution $\mathcal{D}$. In contrast, the test sets generated by k-fold cross validation are independent because each instance is included in only one test set.

To summarize, no single procedure for comparing learning methods based on limited data satisfies all the constraints we would like. It is wise to keep in mind that statistical models rarely fit perfectly the practical constraints in testing learning algorithms when available data is limited. Nevertheless, they do provide approximate confidence intervals that can be of great help in interpreting experimental comparisons of learning methods.

5.7 SUMMARY AND FURTHER READING

The main points of this chapter include:

- Statistical theory provides a basis for estimating the true error ($error_{\mathcal{D}}(h)$) of a hypothesis h, based on its observed error ($error_S(h)$) over a sample S of data. For example, if h is a discrete-valued hypothesis and the data sample S contains $n \geq 30$ examples drawn independently of h and of one another, then the $N\%$ confidence interval for $error_{\mathcal{D}}(h)$ is approximately

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

  where values for z_N are given in Table 5.1.
- In general, the problem of estimating confidence intervals is approached by identifying the parameter to be estimated (e.g., $error_{\mathcal{D}}(h)$) and an estimator

(e.g., $error_S(h)$) for this quantity. Because the estimator is a random variable (e.g., $error_S(h)$ depends on the random sample S), it can be characterized by the probability distribution that governs its value. Confidence intervals can then be calculated by determining the interval that contains the desired probability mass under this distribution.

- One possible cause of errors in estimating hypothesis accuracy is *estimation bias*. If Y is an estimator for some parameter p, the estimation bias of Y is the difference between p and the expected value of Y. For example, if S is the training data used to formulate hypothesis h, then $error_S(h)$ gives an optimistically biased estimate of the true error $error_{\mathcal{D}}(h)$.
- A second cause of estimation error is *variance* in the estimate. Even with an unbiased estimator, the observed value of the estimator is likely to vary from one experiment to another. The variance σ^2 of the distribution governing the estimator characterizes how widely this estimate is likely to vary from the correct value. This variance decreases as the size of the data sample is increased.
- Comparing the relative effectiveness of two learning algorithms is an estimation problem that is relatively easy when data and time are unlimited, but more difficult when these resources are limited. One possible approach described in this chapter is to run the learning algorithms on different subsets of the available data, testing the learned hypotheses on the remaining data, then averaging the results of these experiments.
- In most cases considered here, deriving confidence intervals involves making a number of assumptions and approximations. For example, the above confidence interval for $error_{\mathcal{D}}(h)$ involved approximating a Binomial distribution by a Normal distribution, approximating the variance of this distribution, and assuming instances are generated by a fixed, unchanging probability distribution. While intervals based on such approximations are only approximate confidence intervals, they nevertheless provide useful guidance for designing and interpreting experimental results in machine learning.

The key statistical definitions presented in this chapter are summarized in Table 5.2.

An ocean of literature exists on the topic of statistical methods for estimating means and testing significance of hypotheses. While this chapter introduces the basic concepts, more detailed treatments of these issues can be found in many books and articles. Billingsley et al. (1986) provide a very readable introduction to statistics that elaborates on the issues discussed here. Other texts on statistics include DeGroot (1986); Casella and Berger (1990). Duda and Hart (1973) provide a treatment of these issues in the context of numerical pattern recognition.

Segre et al. (1991, 1996), Etzioni and Etzioni (1994), and Gordon and Segre (1996) discuss statistical significance tests for evaluating learning algorithms whose performance is measured by their ability to improve computational efficiency.

Geman et al. (1992) discuss the tradeoff involved in attempting to minimize bias and variance simultaneously. There is ongoing debate regarding the best way to learn and compare hypotheses from limited data. For example, Dietterich (1996) discusses the risks of applying the paired-difference t test repeatedly to different train-test splits of the data.

EXERCISES

5.1. Suppose you test a hypothesis h and find that it commits $r = 300$ errors on a sample S of $n = 1000$ randomly drawn test examples. What is the standard deviation in $error_S(h)$? How does this compare to the standard deviation in the example at the end of Section 5.3.4?

5.2. Consider a learned hypothesis, h, for some boolean concept. When h is tested on a set of 100 examples, it classifies 83 correctly. What is the standard deviation and the 95% confidence interval for the true error rate for $Error_{\mathcal{D}}(h)$?

5.3. Suppose hypothesis h commits $r = 10$ errors over a sample of $n = 65$ independently drawn examples. What is the 90% confidence interval (two-sided) for the true error rate? What is the 95% one-sided interval (i.e., what is the upper bound U such that $error_{\mathcal{D}}(h) \leq U$ with 95% confidence)? What is the 90% one-sided interval?

5.4. You are about to test a hypothesis h whose $error_{\mathcal{D}}(h)$ is known to be in the range between 0.2 and 0.6. What is the minimum number of examples you must collect to assure that the width of the two-sided 95% confidence interval will be smaller than 0.1?

5.5. Give general expressions for the upper and lower one-sided $N\%$ confidence intervals for the difference in errors between two hypotheses tested on different samples of data. Hint: Modify the expression given in Section 5.5.

5.6. Explain why the confidence interval estimate given in Equation (5.17) applies to estimating the quantity in Equation (5.16), and not the quantity in Equation (5.14).

REFERENCES

Billingsley, P., Croft, D. J., Huntsberger, D. V., & Watson, C. J. (1986). *Statistical inference for management and economics*. Boston: Allyn and Bacon, Inc.

Casella, G., & Berger, R. L. (1990). *Statistical inference*. Pacific Grove, CA: Wadsworth and Brooks/Cole.

DeGroot, M. H. (1986). *Probability and statistics*. (2d ed.) Reading, MA: Addison Wesley.

Dietterich, T. G. (1996). *Proper statistical tests for comparing supervised classification learning algorithms* (Technical Report). Department of Computer Science, Oregon State University, Corvallis, OR.

Dietterich, T. G., & Kong, E. B. (1995). *Machine learning bias, statistical bias, and statistical variance of decision tree algorithms* (Technical Report). Department of Computer Science, Oregon State University, Corvallis, OR.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis*. New York: John Wiley & Sons.

Efron, B., & Tibshirani, R. (1991). Statistical data analysis in the computer age. *Science*, 253, 390–395.

Etzioni, O., & Etzioni, R. (1994). Statistical methods for analyzing speedup learning experiments. *Machine Learning*, 14, 333–347.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4, 1–58.

Gordon, G., & Segre, A.M. (1996). Nonparametric statistical methods for experimental evaluations of speedup learning. *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy.

Maisel, L. (1971). *Probability, statistics, and random processes*. Simon and Schuster Tech Outlines. New York: Simon and Schuster.

Segre, A., Elkan, C., & Russell, A. (1991). A critical look at experimental evaluations of EBL. *Machine Learning*, 6(2).

Segre, A.M, Gordon G., & Elkan, C. P. (1996). Exploratory analysis of speedup learning data using expectation maximization. *Artificial Intelligence*, 85, 301–319.

Speigel, M. R. (1991). *Theory and problems of probability and statistics*. Schaum's Outline Series. New York: McGraw Hill.

Thompson, M.L., & Zucchini, W. (1989). On the statistical analysis of ROC curves. *Statistics in Medicine*, 8, 1277–1290.

White, A. P., & Liu, W. Z. (1994). Bias in information-based measures in decision tree induction. *Machine Learning*, 15, 321–329.

CHAPTER 6

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

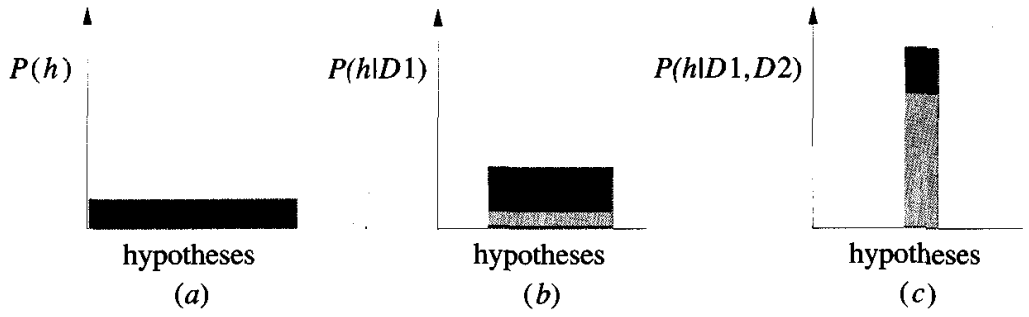


FIGURE 6.1
Evolution of posterior probabilities $P(h|D)$ with increasing training data. (*a*) Uniform priors assign equal probability to each hypothesis. As training data increases first to $D1$ (*b*), then to $D1 \wedge D2$ (*c*), the posterior probability of inconsistent hypotheses becomes zero, while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

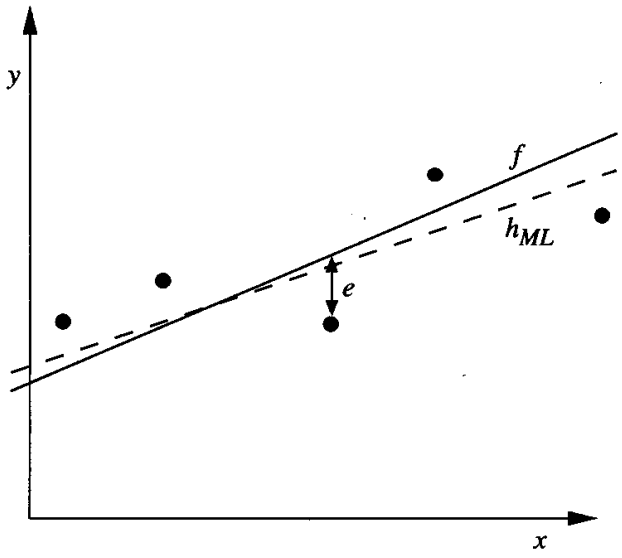


FIGURE 6.2
Learning a real-valued function. The target function f corresponds to the solid line. The training examples $\langle x_i, d_i \rangle$ are assumed to have Normally distributed noise e_i with zero mean added to the true target value $f(x_i)$. The dashed line corresponds to the linear function 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.

[†]Notice the expected length for transmitting one message is therefore $\sum_i -p_i \log_2 p_i$, the formula for the *entropy* (see Chapter 3) of the set of possible messages.

- 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

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

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

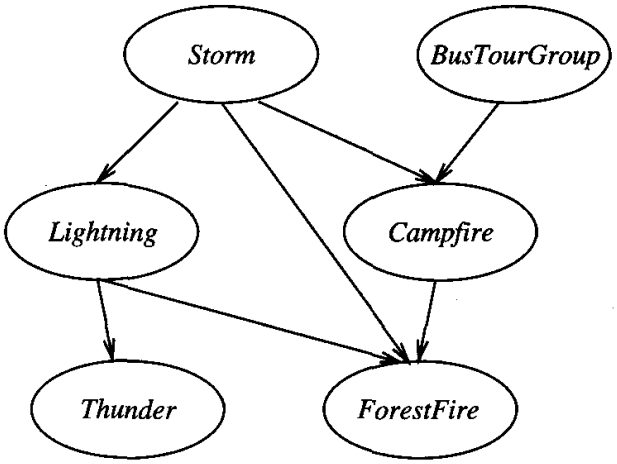
$$P(X_1 \ldots X_l | Y_1 \ldots Y_m, Z_1 \ldots Z_n) = P(X_1 \ldots X_l | Z_1 \ldots Z_n)$$

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

$$P(A_1, A_2|V) = P(A_1|A_2, V)P(A_2|V) \quad (6.23)$$

$$= P(A_1|V)P(A_2|V) \quad (6.24)$$

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

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

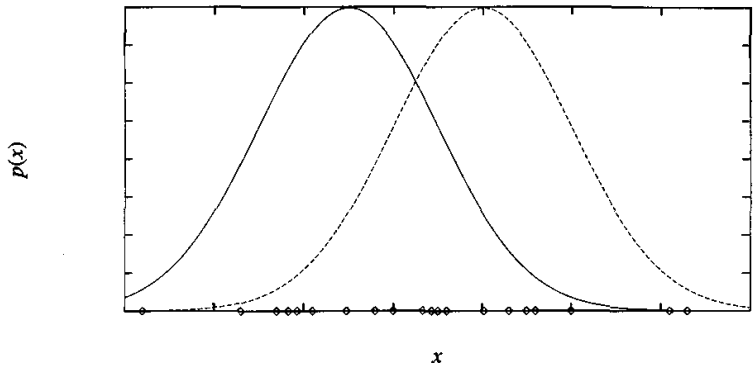


FIGURE 6.4
Instances generated by a mixture of two Normal distributions with identical variance σ. The instances are shown by the points along the x axis. If the means of the Normal distributions are unknown, the EM algorithm can be used to search for their maximum likelihood estimates.

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.

6.13 SUMMARY AND FURTHER READING

The main points of this chapter include:

- Bayesian methods provide the basis for probabilistic learning methods that accommodate (and require) knowledge about the prior probabilities of alternative hypotheses and about the probability of observing various data given the hypothesis. Bayesian methods allow assigning a posterior probability to each candidate hypothesis, based on these assumed priors and the observed data.
- Bayesian methods can be used to determine the most probable hypothesis given the data—the maximum a posteriori (MAP) hypothesis. This is the optimal hypothesis in the sense that no other hypothesis is more likely.
- The Bayes optimal classifier combines the predictions of all alternative hypotheses, weighted by their posterior probabilities, to calculate the most probable classification of each new instance.
- The naive Bayes classifier is a Bayesian learning method that has been found to be useful in many practical applications. It is called "naive" because it incorporates the simplifying assumption that attribute values are conditionally independent, given the classification of the instance. When this assumption is met, the naive Bayes classifier outputs the MAP classification. Even when this assumption is not met, as in the case of learning to classify text, the naive Bayes classifier is often quite effective. Bayesian belief networks provide a more expressive representation for sets of conditional independence assumptions among subsets of the attributes.
- The framework of Bayesian reasoning can provide a useful basis for analyzing certain learning methods that do not directly apply Bayes theorem. For example, under certain conditions it can be shown that minimizing the squared error when learning a real-valued target function corresponds to computing the maximum likelihood hypothesis.
- The Minimum Description Length principle recommends choosing the hypothesis that minimizes the description length of the hypothesis plus the description length of the data given the hypothesis. Bayes theorem and basic results from information theory can be used to provide a rationale for this principle.
- In many practical learning tasks, some of the relevant instance variables may be unobservable. The EM algorithm provides a quite general approach to learning in the presence of unobservable variables. This algorithm begins with an arbitrary initial hypothesis. It then repeatedly calculates the expected values of the hidden variables (assuming the current hypothesis is correct), and then recalculates the maximum likelihood hypothesis (assuming the hidden variables have the expected values calculated by the first step). This procedure converges to a local maximum likelihood hypothesis, along with estimated values for the hidden variables.

There are many good introductory texts on probability and statistics, such as Casella and Berger (1990). Several quick-reference books (e.g., Maisel 1971; Speigel 1991) also provide excellent treatments of the basic notions of probability and statistics relevant to machine learning.

Many of the basic notions of Bayesian classifiers and least-squared error classifiers are discussed by Duda and Hart (1973). Domingos and Pazzani (1996) provide an analysis of conditions under which naive Bayes will output optimal classifications, even when its independence assumption is violated (the key here is that there are conditions under which it will output optimal classifications even when the associated posterior probability estimates are incorrect).

Cestnik (1990) provides a discussion of using the m-estimate to estimate probabilities.

Experimental results comparing various Bayesian approaches to decision tree learning and other algorithms can be found in Michie et al. (1994). Chauvin and Rumelhart (1995) provide a Bayesian analysis of neural network learning based on the BACKPROPAGATION algorithm.

A discussion of the Minimum Description Length principle can be found in Rissanen (1983, 1989). Quinlan and Rivest (1989) describe its use in avoiding overfitting in decision trees.

EXERCISES

6.1. Consider again the example application of Bayes rule in Section 6.2.1. Suppose the doctor decides to order a second laboratory test for the same patient, and suppose the second test returns a positive result as well. What are the posterior probabilities of *cancer* and $\neg cancer$ following these two tests? Assume that the two tests are independent.

6.2. In the example of Section 6.2.1 we computed the posterior probability of cancer by normalizing the quantities $P(+|cancer) \cdot P(cancer)$ and $P(+|\neg cancer) \cdot P(\neg cancer)$ so that they summed to one. Use Bayes theorem and the theorem of total probability (see Table 6.1) to prove that this method is valid (i.e., that normalizing in this way yields the correct value for $P(cancer|+)$).

6.3. Consider the concept learning algorithm $FindG$, which outputs a maximally general consistent hypothesis (e.g., some maximally general member of the version space).

(a) Give a distribution for $P(h)$ and $P(D|h)$ under which $FindG$ is guaranteed to output a MAP hypothesis.

(b) Give a distribution for $P(h)$ and $P(D|h)$ under which $FindG$ is not guaranteed to output a MAP hypothesis.

(c) Give a distribution for $P(h)$ and $P(D|h)$ under which $FindG$ is guaranteed to output a ML hypothesis but not a MAP hypothesis.

6.4. In the analysis of concept learning in Section 6.3 we assumed that the sequence of instances $\langle x_1 \ldots x_m \rangle$ was held fixed. Therefore, in deriving an expression for $P(D|h)$ we needed only consider the probability of observing the sequence of target values $\langle d_1 \ldots d_m \rangle$ for this fixed instance sequence. Consider the more general setting in which the instances are not held fixed, but are drawn independently from some probability distribution defined over the instance space X. The data D must now be described as the set of ordered pairs $\{\langle x_i, d_i \rangle\}$, and $P(D|h)$ must now reflect the

probability of encountering the specific instance x_1, as well as the probability of the observed target value d_i. Show that Equation (6.5) holds even under this more general setting. Hint: Consider the analysis of Section 6.5.

6.5. Consider the Minimum Description Length principle applied to the hypothesis space H consisting of conjunctions of up to n boolean attributes (e.g., $Sunny \wedge Warm$). Assume each hypothesis is encoded simply by listing the attributes present in the hypothesis, where the number of bits needed to encode any one of the n boolean attributes is $\log_2 n$. Suppose the encoding of an example given the hypothesis uses zero bits if the example is consistent with the hypothesis and uses $\log_2 m$ bits otherwise (to indicate which of the m examples was misclassified—the correct classification can be inferred to be the opposite of that predicted by the hypothesis).

(a) Write down the expression for the quantity to be minimized according to the Minimum Description Length principle.

(b) Is it possible to construct a set of training data such that a consistent hypothesis exists, but MDL chooses a less consistent hypothesis? If so, give such a training set. If not, explain why not.

(c) Give probability distributions for $P(h)$ and $P(D|h)$ such that the above MDL algorithm outputs MAP hypotheses.

6.6. Draw the Bayesian belief network that represents the conditional independence assumptions of the naive Bayes classifier for the *PlayTennis* problem of Section 6.9.1. Give the conditional probability table associated with the node *Wind*.

REFERENCES

Buntine W. L. (1994). Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2, 159–225. http://www.cs.washington.edu/research/jair/home.html.

Casella, G., & Berger, R. L. (1990). *Statistical inference*. Pacific Grove, CA: Wadsworth & Brooks/Cole.

Cestnik, B. (1990). Estimating probabilities: A crucial task in machine learning. *Proceedings of the Ninth European Conference on Artificial Intelligence* (pp. 147–149). London: Pitman.

Chauvin, Y., & Rumelhart, D. (1995). *Backpropagation: Theory, architectures, and applications*, (edited collection). Hillsdale, NJ: Lawrence Erlbaum Assoc.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). AUTOCLASS: A bayesian classification system. *Proceedings of AAAI 1988* (pp. 607–611).

Cooper, G. (1990). Computational complexity of probabilistic inference using Bayesian belief networks (research note). *Artificial Intelligence*, 42, 393–405.

Cooper, G., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9, 309–347.

Dagum, P., & Luby, M. (1993). Approximating probabilistic reasoning in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1), 141–153.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, Series B, 39(1), 1–38.

Domingos, P., & Pazzani, M. (1996). Beyond independence: Conditions for the optimality of the simple Bayesian classifier. *Proceedings of the 13th International Conference on Machine Learning* (pp. 105–112).

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New York: John Wiley & Sons.

Hearst, M., & Hirsh, H. (Eds.) (1996). Papers from the AAAI Spring Symposium on Machine Learning in Information Access, Stanford, March 25–27. http://www.parc.xerox.com/istl/projects/mlia/

Heckerman, D., Geiger, D., & Chickering, D. (1995) Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20, 197. Kluwer Academic Publishers.

Jensen, F. V. (1996). *An introduction to Bayesian networks*. New York: Springer Verlag.

Joachims, T. (1996). *A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization*, (Computer Science Technical Report CMU-CS-96-118). Carnegie Mellon University.

Lang, K. (1995). Newsweeder: Learning to filter netnews. In Prieditis and Russell (Eds.), *Proceedings of the 12th International Conference on Machine Learning* (pp. 331–339). San Francisco: Morgan Kaufmann Publishers.

Lewis, D. (1991). *Representation and learning in information retrieval*, (Ph.D. thesis), (COINS Technical Report 91-93). Dept. of Computer and Information Science, University of Massachusetts.

Madigan, D., & Rafferty, A. (1994). Model selection and accounting for model uncertainty in graphical models using Occam's window. *Journal of the American Statistical Association*, 89, 1535–1546.

Maisel, L. (1971). *Probability, statistics, and random processes*. Simon and Schuster Tech Outlines. New York: Simon and Schuster.

Mehta, M., Rissanen, J., & Agrawal, R. (1995). MDL-based decision tree pruning. In U. M. Fayyard and R. Uthurusamy (Eds.), *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press.

Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). *Machine learning, neural and statistical classification*, (edited collection). New York: Ellis Horwood.

Opper, M., & Haussler, D. (1991). Generalization performance of Bayes optimal prediction algorithm for learning a perceptron. *Physical Review Letters*, 66, 2677–2681.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan-Kaufmann.

Pradham, M., & Dagum, P. (1996). Optimal Monte Carlo estimation of belief network inference. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (pp. 446–453).

Quinlan, J. R., & Rivest, R. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation*, 80, 227–248.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.

Rissanen, J. (1983). A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2), 416–431.

Rissanen, J., (1989). *Stochastic complexity in statistical inquiry*. New Jersey: World Scientific Pub.

Rissanen, J. (1991). *Information theory and neural nets*. IBM Research Report RJ 8438 (76446), IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

Rocchio, J. (1971). Relevance feedback in information retrieval. In *The SMART retrieval system: Experiments in automatic document processing*, (Chap. 14, pp. 313–323). Englewood Cliffs, NJ: Prentice-Hall.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Englewood Cliffs, NJ: Prentice-Hall.

Russell, S., Binder, J., Koller, D., & Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal. San Francisco: Morgan Kaufmann.

Salton, G. (1991). Developments in automatic text retrieval. *Science*, 253, 974–979.

Shannon, C. E., & Weaver, W. (1949). *The mathematical theory of communication*. Urbana: University of Illinois Press.

Speigel, M. R. (1991). *Theory and problems of probability and statistics*. Schaum's Outline Series. New York: McGraw Hill.

Spirtes, P., Glymour, C., & Scheines, R. (1993). *Causation, prediction, and search*. New York: Springer Verlag. http://hss.cmu.edu/html/departments/philosophy/TETRAD.BOOK/book.html

CHAPTER 7

COMPUTATIONAL LEARNING THEORY

This chapter presents a theoretical characterization of the difficulty of several types of machine learning problems and the capabilities of several types of machine learning algorithms. This theory seeks to answer questions such as "Under what conditions is successful learning possible and impossible?" and "Under what conditions is a particular learning algorithm assured of learning successfully?" Two specific frameworks for analyzing learning algorithms are considered. Within the probably approximately correct (PAC) framework, we identify classes of hypotheses that can and cannot be learned from a polynomial number of training examples and we define a natural measure of complexity for hypothesis spaces that allows bounding the number of training examples required for inductive learning. Within the mistake bound framework, we examine the number of training errors that will be made by a learner before it determines the correct hypothesis.

7.1 INTRODUCTION

When studying machine learning it is natural to wonder what general laws may govern machine (and nonmachine) learners. Is it possible to identify classes of learning problems that are inherently difficult or easy, independent of the learning algorithm? Can one characterize the number of training examples necessary or sufficient to assure successful learning? How is this number affected if the learner is allowed to pose queries to the trainer, versus observing a random sample of training examples? Can one characterize the number of mistakes that a learner

will make before learning the target function? Can one characterize the inherent computational complexity of classes of learning problems?

Although general answers to all these questions are not yet known, fragments of a computational theory of learning have begun to emerge. This chapter presents key results from this theory, providing answers to these questions within particular problem settings. We focus here on the problem of inductively learning an unknown target function, given only training examples of this target function and a space of candidate hypotheses. Within this setting, we will be chiefly concerned with questions such as how many training examples are sufficient to successfully learn the target function, and how many mistakes will the learner make before succeeding. As we shall see, it is possible to set quantitative bounds on these measures, depending on attributes of the learning problem such as:

- the size or complexity of the hypothesis space considered by the learner
- the accuracy to which the target concept must be approximated
- the probability that the learner will output a successful hypothesis
- the manner in which training examples are presented to the learner

For the most part, we will focus not on individual learning algorithms, but rather on broad classes of learning algorithms characterized by the hypothesis spaces they consider, the presentation of training examples, etc. Our goal is to answer questions such as:

- *Sample complexity.* How many training examples are needed for a learner to converge (with high probability) to a successful hypothesis?
- *Computational complexity.* How much computational effort is needed for a learner to converge (with high probability) to a successful hypothesis?
- *Mistake bound.* How many training examples will the learner misclassify before converging to a successful hypothesis?

Note there are many specific settings in which we could pursue such questions. For example, there are various ways to specify what it means for the learner to be "successful." We might specify that to succeed, the learner must output a hypothesis identical to the target concept. Alternatively, we might simply require that it output a hypothesis that agrees with the target concept most of the time, or that it usually output such a hypothesis. Similarly, we must specify how training examples are to be obtained by the learner. We might specify that training examples are presented by a helpful teacher, or obtained by the learner performing experiments, or simply generated at random according to some process outside the learner's control. As we might expect, the answers to the above questions depend on the particular setting, or learning model, we have in mind.

The remainder of this chapter is organized as follows. Section 7.2 introduces the probably approximately correct (PAC) learning setting. Section 7.3 then analyzes the sample complexity and computational complexity for several learning

problems within this PAC setting. Section 7.4 introduces an important measure of hypothesis space complexity called the VC-dimension and extends our PAC analysis to problems in which the hypothesis space is infinite. Section 7.5 introduces the mistake-bound model and provides a bound on the number of mistakes made by several learning algorithms discussed in earlier chapters. Finally, we introduce the WEIGHTED-MAJORITY algorithm, a practical algorithm for combining the predictions of multiple competing learning algorithms, along with a theoretical mistake bound for this algorithm.

7.2 PROBABLY LEARNING AN APPROXIMATELY CORRECT HYPOTHESIS

In this section we consider a particular setting for the learning problem, called the *probably approximately correct* (PAC) learning model. We begin by specifying the problem setting that defines the PAC learning model, then consider the questions of how many training examples and how much computation are required in order to learn various classes of target functions within this PAC model. For the sake of simplicity, we restrict the discussion to the case of learning boolean-valued concepts from noise-free training data. However, many of the results can be extended to the more general scenario of learning real-valued target functions (see, for example, Natarajan 1991), and some can be extended to learning from certain types of noisy data (see, for example, Laird 1988; Kearns and Vazirani 1994).

7.2.1 The Problem Setting

As in earlier chapters, let X refer to the set of all possible instances over which target functions may be defined. For example, X might represent the set of all people, each described by the attributes *age* (e.g., *young* or *old*) and *height* (*short* or *tall*). Let C refer to some set of target concepts that our learner might be called upon to learn. Each target concept c in C corresponds to some subset of X, or equivalently to some boolean-valued function $c : X \rightarrow \{0, 1\}$. For example, one target concept c in C might be the concept "people who are skiers." If x is a positive example of c, then we will write $c(x) = 1$; if x is a negative example, $c(x) = 0$.

We assume instances are generated at random from X according to some probability distribution $\mathcal{D}$. For example, $\mathcal{D}$ might be the distribution of instances generated by observing people who walk out of the largest sports store in Switzerland. In general, $\mathcal{D}$ may be any distribution, and it will not generally be known to the learner. All that we require of $\mathcal{D}$ is that it be stationary; that is, that the distribution not change over time. Training examples are generated by drawing an instance x at random according to $\mathcal{D}$, then presenting x along with its target value, $c(x)$, to the learner.

The learner L considers some set H of possible hypotheses when attempting to learn the target concept. For example, H might be the set of all hypotheses

describable by conjunctions of the attributes *age* and *height*. After observing a sequence of training examples of the target concept c, L must output some hypothesis h from H, which is its estimate of c. To be fair, we evaluate the success of L by the performance of h over new instances drawn randomly from X according to $\mathcal{D}$, the same probability distribution used to generate the training data.

Within this setting, we are interested in characterizing the performance of various learners L using various hypothesis spaces H, when learning individual target concepts drawn from various classes C. Because we demand that L be general enough to learn any target concept from C regardless of the distribution of training examples, we will often be interested in worst-case analyses over all possible target concepts from C and all possible instance distributions $\mathcal{D}$.

7.2.2 Error of a Hypothesis

Because we are interested in how closely the learner's output hypothesis h approximates the actual target concept c, let us begin by defining the *true error* of a hypothesis h with respect to target concept c and instance distribution $\mathcal{D}$. Informally, the true error of h is just the error rate we expect when applying h to future instances drawn according to the probability distribution $\mathcal{D}$. In fact, we already defined the true error of h in Chapter 5. For convenience, we restate the definition here using c to represent the boolean target function.

> ***Definition:*** The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target concept c and distribution $\mathcal{D}$ is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.
>
> $$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[c(x) \neq h(x)]$$

Here the notation $\Pr_{x \in \mathcal{D}}$ indicates that the probability is taken over the instance distribution $\mathcal{D}$.

Figure 7.1 shows this definition of error in graphical form. The concepts c and h are depicted by the sets of instances within X that they label as positive. The error of h with respect to c is the probability that a randomly drawn instance will fall into the region where h and c disagree (i.e., their set difference). Note we have chosen to define error over the *entire distribution* of instances—not simply over the training examples—because this is the true error we expect to encounter when actually using the learned hypothesis h on subsequent instances drawn from $\mathcal{D}$.

Note that error depends strongly on the unknown probability distribution $\mathcal{D}$. For example, if $\mathcal{D}$ is a uniform probability distribution that assigns the same probability to every instance in X, then the error for the hypothesis in Figure 7.1 will be the fraction of the total instance space that falls into the region where h and c disagree. However, the same h and c will have a much higher error if $\mathcal{D}$ happens to assign very high probability to instances for which h and c disagree. In the extreme, if $\mathcal{D}$ happens to assign zero probability to the instances for which

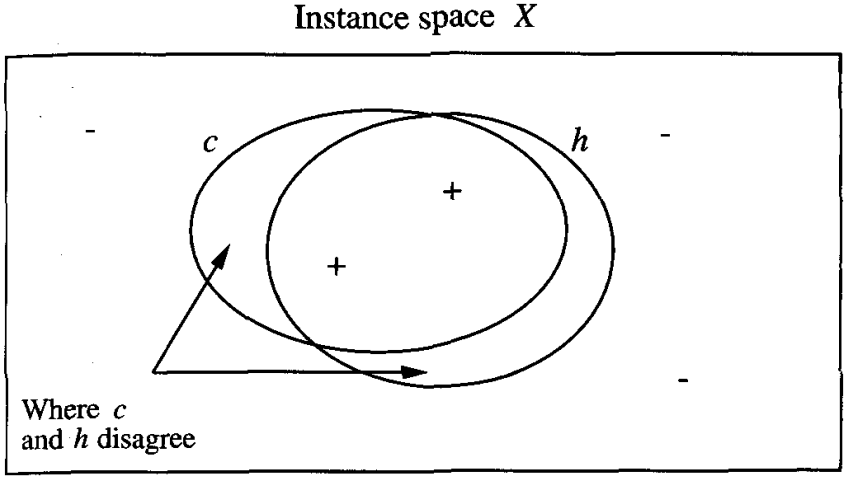


FIGURE 7.1
The error of hypothesis h with respect to target concept c. The error of h with respect to c is the probability that a randomly drawn instance will fall into the region where h and c disagree on its classification. The + and − points indicate positive and negative training examples. Note h has a nonzero error with respect to c despite the fact that h and c agree on all five training examples observed thus far.

$h(x) = c(x)$, then the error for the h in Figure 7.1 will be 1, despite the fact the h and c agree on a very large number of (zero probability) instances.

Finally, note that the error of h with respect to c is not directly observable to the learner. L can only observe the performance of h over the *training examples*, and it must choose its output hypothesis on this basis only. We will use the term *training error* to refer to the fraction of training examples misclassified by h, in contrast to the *true error* defined above. Much of our analysis of the complexity of learning centers around the question "how probable is it that the observed *training error* for h gives a misleading estimate of the *true error*$_{\mathcal{D}}(h)$?"

Notice the close relationship between this question and the questions considered in Chapter 5. Recall that in Chapter 5 we defined the *sample error* of h with respect to a set S of examples to be the fraction of S misclassified by h. The training error defined above is just the sample error when S is the set of training examples. In Chapter 5 we determined the probability that the sample error will provide a misleading estimate of the true error, under the assumption that the data sample S is drawn independent of h. However, when S is the set of training data, the learned hypothesis h depends very much on S! Therefore, in this chapter we provide an analysis that addresses this important special case.

7.2.3 PAC Learnability

Our aim is to characterize classes of target concepts that can be reliably learned from a reasonable number of randomly drawn training examples and a reasonable amount of computation.

What kinds of statements about learnability should we guess hold true? We might try to characterize the number of training examples needed to learn

a hypothesis h for which $error_{\mathcal{D}}(h) = 0$. Unfortunately, it turns out this is futile in the setting we are considering, for two reasons. First, unless we provide training examples corresponding to every possible instance in X (an unrealistic assumption), there may be multiple hypotheses consistent with the provided training examples, and the learner cannot be certain to pick the one corresponding to the target concept. Second, given that the training examples are drawn randomly, there will always be some nonzero probability that the training examples encountered by the learner will be misleading. (For example, although we might frequently see skiers of different heights, on any given day there is some small chance that all observed training examples will happen to be 2 meters tall.)

To accommodate these two difficulties, we weaken our demands on the learner in two ways. First, we will not require that the learner output a zero error hypothesis—we will require only that its error be bounded by some constant, ϵ, that can be made arbitrarily small. Second, we will not require that the learner succeed for *every* sequence of randomly drawn training examples—we will require only that its probability of failure be bounded by some constant, δ, that can be made arbitrarily small. In short, we require only that the learner *probably* learn a hypothesis that is *approximately correct*—hence the term probably approximately correct learning, or PAC learning for short.

Consider some class C of possible target concepts and a learner L using hypothesis space H. Loosely speaking, we will say that the concept class C is PAC-learnable by L using H if, for any target concept c in C, L will with probability $(1 - \delta)$ output a hypothesis h with $error_{\mathcal{D}}(h) < \epsilon$, after observing a reasonable number of training examples and performing a reasonable amount of computation. More precisely,

> ***Definition:*** Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H. C is **PAC-learnable** by L using H if for all $c \in C$, distributions $\mathcal{D}$ over X, ϵ such that $0 < \epsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least $(1 - \delta)$ output a hypothesis $h \in H$ such that $error_{\mathcal{D}}(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, n, and $size(c)$.

Our definition requires two things from L. First, L must, with arbitrarily high probability $(1 - \delta)$, output a hypothesis having arbitrarily low error (ϵ). Second, it must do so efficiently—in time that grows at most polynomially with $1/\epsilon$ and $1/\delta$, which define the strength of our demands on the output hypothesis, and with n and $size(c)$ that define the inherent complexity of the underlying instance space X and concept class C. Here, n is the size of instances in X. For example, if instances in X are conjunctions of k boolean features, then $n = k$. The second space parameter, $size(c)$, is the encoding length of c in C, assuming some representation for C. For example, if concepts in C are conjunctions of up to k boolean features, each described by listing the indices of the features in the conjunction, then $size(c)$ is the number of boolean features actually used to describe c.

Our definition of PAC learning may at first appear to be concerned only with the computational resources required for learning, whereas in practice we are

usually more concerned with the number of training examples required. However, the two are very closely related: If L requires some minimum processing time per training example, then for C to be PAC-learnable by L, L *must learn from a polynomial number of training examples.* In fact, a typical approach to showing that some class C of target concepts is PAC-learnable, is to first show that each target concept in C can be learned from a polynomial number of training examples and then show that the processing time per example is also polynomially bounded.

Before moving on, we should point out a restrictive assumption implicit in our definition of PAC-learnable. This definition implicitly assumes that the learner's hypothesis space H contains a hypothesis with arbitrarily small error for every target concept in C. This follows from the requirement in the above definition that the learner succeed when the error bound ϵ is arbitrarily close to zero. Of course this is difficult to assure if one does not know C in advance (what is C for a program that must learn to recognize faces from images?), unless H is taken to be the power set of X. As pointed out in Chapter 2, such an unbiased H will not support accurate generalization from a reasonable number of training examples. Nevertheless, the results based on the PAC learning model provide useful insights regarding the relative complexity of different learning problems and regarding the rate at which generalization accuracy improves with additional training examples. Furthermore, in Section 7.3.1 we will lift this restrictive assumption, to consider the case in which the learner makes no prior assumption about the form of the target concept.

7.3 SAMPLE COMPLEXITY FOR FINITE HYPOTHESIS SPACES

As noted above, PAC-learnability is largely determined by the number of training examples required by the learner. The growth in the number of required training examples with problem size, called the *sample complexity* of the learning problem, is the characteristic that is usually of greatest interest. The reason is that in most practical settings the factor that most limits success of the learner is the limited availability of training data.

Here we present a general bound on the sample complexity for a very broad class of learners, called *consistent learners*. A learner is *consistent* if it outputs hypotheses that perfectly fit the training data, whenever possible. It is quite reasonable to ask that a learning algorithm be consistent, given that we typically prefer a hypothesis that fits the training data over one that does not. Note that many of the learning algorithms discussed in earlier chapters, including all the learning algorithms described in Chapter 2, are consistent learners.

Can we derive a bound on the number of training examples required by *any* consistent learner, independent of the specific algorithm it uses to derive a consistent hypothesis? The answer is yes. To accomplish this, it is useful to recall the definition of version space from Chapter 2. There we defined the version space, $VS_{H,D}$, to be the set of all hypotheses $h \in H$ that correctly classify the training examples D.

$$VS_{H,D} = \{h \in H | (\forall \langle x, c(x) \rangle \in D)\ (h(x) = c(x))\}$$

The significance of the version space here is that *every consistent learner outputs a hypothesis belonging to the version space*, regardless of the instance space X, hypothesis space H, or training data D. The reason is simply that by definition the version space $VS_{H,D}$ contains every consistent hypothesis in H. Therefore, *to bound the number of examples needed by any consistent learner, we need only bound the number of examples needed to assure that the version space contains no unacceptable hypotheses.* The following definition, after Haussler (1988), states this condition precisely.

> ***Definition:*** Consider a hypothesis space H, target concept c, instance distribution $\mathcal{D}$, and set of training examples D of c. The version space $VS_{H,D}$ is said to be ϵ-**exhausted** with respect to c and $\mathcal{D}$, if every hypothesis h in $VS_{H,D}$ has error less than ϵ with respect to c and $\mathcal{D}$.
>
> $$(\forall h \in VS_{H,D})\ error_{\mathcal{D}}(h) < \epsilon$$

This definition is illustrated in Figure 7.2. The version space is ϵ-exhausted just in the case that all the hypotheses consistent with the observed training examples (i.e., those with zero training error) happen to have true error less than ϵ. Of course from the learner's viewpoint all that can be known is that these hypotheses fit the training data equally well—they all have zero training error. Only an observer who knew the identity of the target concept could determine with certainty whether the version space is ϵ-exhausted. Surprisingly, a probabilistic argument allows us to bound the probability that the version space will be ϵ-exhausted after a given number of training examples, even without knowing the identity of the target concept or the distribution from which training examples

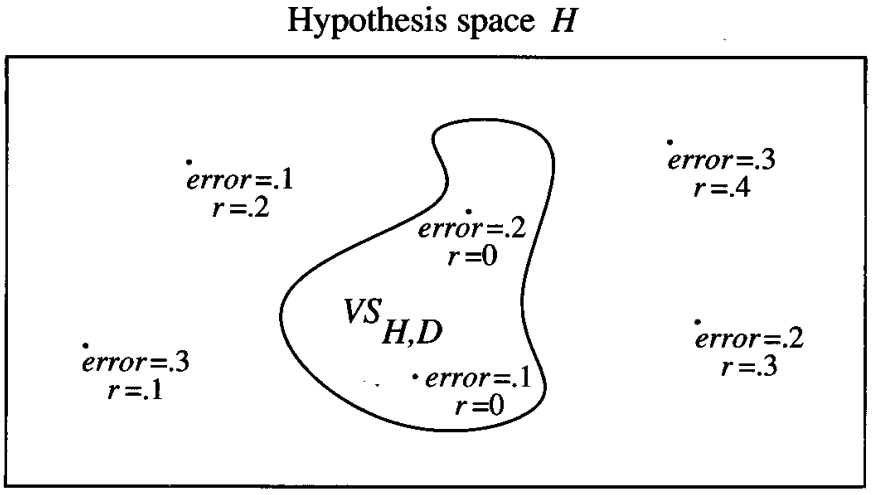


FIGURE 7.2
Exhausting the version space. The version space $VS_{H,D}$ is the subset of hypotheses $h \in H$, which have zero training error (denoted by $r = 0$ in the figure). Of course the true $error_{\mathcal{D}}(h)$ (denoted by *error* in the figure) may be nonzero, even for hypotheses that commit zero errors over the training data. The version space is said to be ϵ-exhausted when all hypotheses h remaining in $VS_{H,D}$ have $error_{\mathcal{D}}(h) < \epsilon$.

are drawn. Haussler (1988) provides such a bound, in the form of the following theorem.

Theorem 7.1. ϵ-exhausting the version space. If the hypothesis space H is finite, and D is a sequence of $m \geq 1$ independent randomly drawn examples of some target concept c, then for any $0 \leq \epsilon \leq 1$, the probability that the version space $VS_{H,D}$ is not ϵ-exhausted (with respect to c) is less than or equal to

$$|H|e^{-\epsilon m}$$

Proof. Let $h_1, h_2, \ldots h_k$ be all the hypotheses in H that have true error greater than ϵ with respect to c. We fail to ϵ-exhaust the version space if and only if at least one of these k hypotheses happens to be consistent with all m independent random training examples. The probability that any single hypothesis having true error greater than ϵ would be consistent with one randomly drawn example is at most $(1-\epsilon)$. Therefore the probability that this hypothesis will be consistent with m independently drawn examples is at most $(1-\epsilon)^m$. Given that we have k hypotheses with error greater than ϵ, the probability that at least one of these will be consistent with all m training examples is at most

$$k(1-\epsilon)^m$$

And since $k \leq |H|$, this is at most $|H|(1-\epsilon)^m$. Finally, we use a general inequality stating that if $0 \leq \epsilon \leq 1$ then $(1-\epsilon) \leq e^{-\epsilon}$. Thus,

$$k(1-\epsilon)^m \leq |H|(1-\epsilon)^m \leq |H|e^{-\epsilon m}$$

which proves the theorem. □

We have just proved an upper bound on the probability that the version space is not ϵ-exhausted, based on the number of training examples m, the allowed error ϵ, and the size of H. Put another way, this bounds the probability that m training examples will fail to eliminate all "bad" hypotheses (i.e., hypotheses with true error greater than ϵ), for any consistent learner using hypothesis space H.

Let us use this result to determine the number of training examples required to reduce this probability of failure below some desired level δ.

$$|H|e^{-\epsilon m} \leq \delta \tag{7.1}$$

Rearranging terms to solve for m, we find

$$m \geq \frac{1}{\epsilon}(\ln|H| + \ln(1/\delta)) \tag{7.2}$$

To summarize, the inequality shown in Equation (7.2) provides a general bound on the number of training examples sufficient for *any consistent learner* to successfully learn any target concept in H, for any desired values of δ and ϵ. This number m of training examples is sufficient to assure that any consistent hypothesis will be probably (with probability $(1-\delta)$) approximately (within error ϵ) correct. Notice m grows linearly in $1/\epsilon$ and logarithmically in $1/\delta$. It also grows logarithmically in the size of the hypothesis space H.

Note that the above bound can be a substantial overestimate. For example, although the probability of failing to exhaust the version space must lie in the interval [0, 1], the bound given by the theorem grows linearly with $|H|$. For sufficiently large hypothesis spaces, this bound can easily be greater than one. As a result, the bound given by the inequality in Equation (7.2) can substantially overestimate the number of training examples required. The weakness of this bound is mainly due to the $|H|$ term, which arises in the proof when summing the probability that a single hypothesis could be unacceptable, over all possible hypotheses. In fact, a much tighter bound is possible in many cases, as well as a bound that covers infinitely large hypothesis spaces. This will be the subject of Section 7.4.

7.3.1 Agnostic Learning and Inconsistent Hypotheses

Equation (7.2) is important because it tells us how many training examples suffice to ensure (with probability $(1-\delta)$) that every hypothesis in H having zero training error will have a true error of at most ϵ. Unfortunately, if H does not contain the target concept c, then a zero-error hypothesis cannot always be found. In this case, the most we might ask of our learner is to output the hypothesis from H that has the *minimum* error over the training examples. A learner that makes no assumption that the target concept is representable by H and that simply finds the hypothesis with minimum training error, is often called an *agnostic* learner, because it makes no prior commitment about whether or not $C \subseteq H$.

Although Equation (7.2) is based on the assumption that the learner outputs a zero-error hypothesis, a similar bound can be found for this more general case in which the learner entertains hypotheses with nonzero training error. To state this precisely, let D denote the particular set of training examples available to the learner, in contrast to $\mathcal{D}$, which denotes the probability distribution over the entire set of instances. Let $error_D(h)$ denote the training error of hypothesis h. In particular, $error_D(h)$ is defined as the fraction of the training examples in D that are misclassified by h. Note the $error_D(h)$ over the particular sample of training data D may differ from the true error $error_{\mathcal{D}}(h)$ over the entire probability distribution $\mathcal{D}$. Now let h_{best} denote the hypothesis from H having lowest training error over the training examples. How many training examples suffice to ensure (with high probability) that its true error $error_{\mathcal{D}}(h_{best})$ will be no more than $\epsilon + error_D(h_{best})$? Notice the question considered in the previous section is just a special case of this question, when $error_D(h_{best})$ happens to be zero.

This question can be answered (see Exercise 7.3) using an argument analogous to the proof of Theorem 7.1. It is useful here to invoke the general Hoeffding bounds (sometimes called the additive Chernoff bounds). The Hoeffding bounds characterize the deviation between the true probability of some event and its observed frequency over m independent trials. More precisely, these bounds apply to experiments involving m distinct Bernoulli trials (e.g., m independent flips of a coin with some probability of turning up heads). This is exactly analogous to the setting we consider when estimating the error of a hypothesis in Chapter 5: The

probability of the coin being heads corresponds to the probability that the hypothesis will misclassify a randomly drawn instance. The m independent coin flips correspond to the m independently drawn instances. The frequency of heads over the m examples corresponds to the frequency of misclassifications over the m instances.

The Hoeffding bounds state that if the training error $error_D(h)$ is measured over the set D containing m randomly drawn examples, then

$$\Pr[error_{\mathcal{D}}(h) > error_D(h) + \epsilon] \leq e^{-2m\epsilon^2}$$

This gives us a bound on the probability that an arbitrarily chosen single hypothesis has a very misleading training error. To assure that the *best* hypothesis found by L has an error bounded in this way, we must consider the probability that any one of the $|H|$ hypotheses could have a large error

$$\Pr[(\exists h \in H)(error_{\mathcal{D}}(h) > error_D(h) + \epsilon)] \leq |H|e^{-2m\epsilon^2}$$

If we call this probability δ, and ask how many examples m suffice to hold δ to some desired value, we now obtain

$$m \geq \frac{1}{2\epsilon^2}(\ln|H| + \ln(1/\delta)) \tag{7.3}$$

This is the generalization of Equation (7.2) to the case in which the learner still picks the best hypothesis $h \in H$, but where the best hypothesis may have nonzero training error. Notice that m depends logarithmically on H and on $1/\delta$, as it did in the more restrictive case of Equation (7.2). However, in this less restrictive situation m now grows as the square of $1/\epsilon$, rather than linearly with $1/\epsilon$.

7.3.2 Conjunctions of Boolean Literals Are PAC-Learnable

Now that we have a bound indicating the number of training examples sufficient to probably approximately learn the target concept, we can use it to determine the sample complexity and PAC-learnability of some specific concept classes.

Consider the class C of target concepts described by conjunctions of boolean literals. A boolean *literal* is any boolean variable (e.g., Old), or its negation (e.g., $\neg Old$). Thus, conjunctions of boolean literals include target concepts such as "$Old \wedge \neg Tall$". Is C PAC-learnable? We can show that the answer is yes by first showing that any consistent learner will require only a polynomial number of training examples to learn any c in C, and then suggesting a specific algorithm that uses polynomial time per training example.

Consider any consistent learner L using a hypothesis space H identical to C. We can use Equation (7.2) to compute the number m of random training examples sufficient to ensure that L will, with probability $(1 - \delta)$, output a hypothesis with maximum error ϵ. To accomplish this, we need only determine the size $|H|$ of the hypothesis space.

Now consider the hypothesis space H defined by conjunctions of literals based on n boolean variables. The size $|H|$ of this hypothesis space is 3^n. To see this, consider the fact that there are only three possibilities for each variable in

any given hypothesis: Include the variable as a literal in the hypothesis, include its negation as a literal, or ignore it. Given n such variables, there are 3^n distinct hypotheses.

Substituting $|H| = 3^n$ into Equation (7.2) gives the following bound for the sample complexity of learning conjunctions of up to n boolean literals.

$$m \geq \frac{1}{\epsilon}(n \ln 3 + \ln(1/\delta)) \tag{7.4}$$

For example, if a consistent learner attempts to learn a target concept described by conjunctions of up to 10 boolean literals, and we desire a 95% probability that it will learn a hypothesis with error less than .1, then it suffices to present m randomly drawn training examples, where $m = \frac{1}{.1}(10 \ln 3 + \ln(1/.05)) = 140$.

Notice that m grows linearly in the number of literals n, linearly in $1/\epsilon$, and logarithmically in $1/\delta$. What about the overall computational effort? That will depend, of course, on the specific learning algorithm. However, as long as our learning algorithm requires no more than polynomial computation per training example, and no more than a polynomial number of training examples, then the total computation required will be polynomial as well.

In the case of learning conjunctions of boolean literals, one algorithm that meets this requirement has already been presented in Chapter 2. It is the FIND-S algorithm, which incrementally computes the most specific hypothesis consistent with the training examples. For each new positive training example, this algorithm computes the intersection of the literals shared by the current hypothesis and the new training example, using time linear in n. Therefore, the FIND-S algorithm PAC-learns the concept class of conjunctions of n boolean literals with negations.

Theorem 7.2. PAC-learnability of boolean conjunctions. The class C of conjunctions of boolean literals is PAC-learnable by the FIND-S algorithm using $H = C$.

Proof. Equation (7.4) shows that the sample complexity for this concept class is polynomial in n, $1/\delta$, and $1/\epsilon$, and independent of $size(c)$. To incrementally process each training example, the FIND-S algorithm requires effort linear in n and independent of $1/\delta$, $1/\epsilon$, and $size(c)$. Therefore, this concept class is PAC-learnable by the FIND-S algorithm. □

7.3.3 PAC-Learnability of Other Concept Classes

As we just saw, Equation (7.2) provides a general basis for bounding the sample complexity for learning target concepts in some given class C. Above we applied it to the class of conjunctions of boolean literals. It can also be used to show that many other concept classes have polynomial sample complexity (e.g., see Exercise 7.2).

7.3.3.1 UNBIASED LEARNERS

Not all concept classes have polynomially bounded sample complexity according to the bound of Equation (7.2). For example, consider the *unbiased* concept class

C that contains every teachable concept relative to X. The set C of all definable target concepts corresponds to the power set of X—the set of all subsets of X—which contains $|C| = 2^{|X|}$ concepts. Suppose that instances in X are defined by n boolean features. In this case, there will be $|X| = 2^n$ distinct instances, and therefore $|C| = 2^{|X|} = 2^{2^n}$ distinct concepts. Of course to learn such an unbiased concept class, the learner must itself use an unbiased hypothesis space $H = C$. Substituting $|H| = 2^{2^n}$ into Equation (7.2) gives the sample complexity for learning the unbiased concept class relative to X.

$$m \geq \frac{1}{\epsilon}(2^n \ln 2 + \ln(1/\delta)) \tag{7.5}$$

Thus, this unbiased class of target concepts has exponential sample complexity under the PAC model, according to Equation (7.2). Although Equations (7.2) and (7.5) are not tight upper bounds, it can in fact be proven that the sample complexity for the unbiased concept class is exponential in n.

7.3.3.2 K-TERM DNF AND K-CNF CONCEPTS

It is also possible to find concept classes that have polynomial sample complexity, but nevertheless cannot be learned in polynomial time. One interesting example is the concept class C of k-term disjunctive normal form (k-term DNF) expressions. k-term DNF expressions are of the form $T_1 \vee T_2 \vee \cdots \vee T_k$, where each term T_i is a conjunction of n boolean attributes and their negations. Assuming $H = C$, it is easy to show that $|H|$ is at most 3^{nk} (because there are k terms, each of which may take on 3^n possible values). Note 3^{nk} is an overestimate of H, because it is double counting the cases where $T_i = T_j$ and where T_i is *more_general_than* T_j. Still, we can use this upper bound on $|H|$ to obtain an upper bound on the sample complexity, substituting this into Equation (7.2).

$$m \geq \frac{1}{\epsilon}(nk \ln 3 + \ln(1/\delta)) \tag{7.6}$$

which indicates that the sample complexity of k-term DNF is polynomial in $1/\epsilon$, $1/\delta$, n, and k. Despite having polynomial sample complexity, the computational complexity is not polynomial, because this learning problem can be shown to be equivalent to other problems that are known to be unsolvable in polynomial time (unless $RP = NP$). Thus, although k-term DNF has polynomial sample complexity, it does not have polynomial computational complexity for a learner using $H = C$.

The surprising fact about k-term DNF is that although it is not PAC-learnable, there is a strictly larger concept class that is! This is possible because the larger concept class has polynomial computation complexity per example and still has polynomial sample complexity. This larger class is the class of k-CNF expressions: conjunctions of arbitrary length of the form $T_1 \wedge T_2 \wedge \cdots \wedge T_j$, where each T_i is a disjunction of up to k boolean attributes. It is straightforward to show that k-CNF subsumes k-DNF, because any k-term DNF expression can easily be

rewritten as a k-CNF expression (but not vice versa). Although k-CNF is more expressive than k-term DNF, it has both polynomial sample complexity and polynomial time complexity. Hence, the concept class k-term DNF is PAC learnable by an efficient algorithm using $H = k$-CNF. See Kearns and Vazirani (1994) for a more detailed discussion.

7.4 SAMPLE COMPLEXITY FOR INFINITE HYPOTHESIS SPACES

In the above section we showed that sample complexity for PAC learning grows as the logarithm of the size of the hypothesis space. While Equation (7.2) is quite useful, there are two drawbacks to characterizing sample complexity in terms of $|H|$. First, it can lead to quite weak bounds (recall that the bound on δ can be significantly greater than 1 for large $|H|$). Second, in the case of infinite hypothesis spaces we cannot apply Equation (7.2) at all!

Here we consider a second measure of the complexity of H, called the Vapnik-Chervonenkis dimension of H (VC dimension, or $VC(H)$, for short). As we shall see, we can state bounds on sample complexity that use $VC(H)$ rather than $|H|$. In many cases, the sample complexity bounds based on $VC(H)$ will be tighter than those from Equation (7.2). In addition, these bounds allow us to characterize the sample complexity of many infinite hypothesis spaces, and can be shown to be fairly tight.

7.4.1 Shattering a Set of Instances

The VC dimension measures the complexity of the hypothesis space H, not by the number of distinct hypotheses $|H|$, but instead by the number of distinct instances from X that can be completely discriminated using H.

To make this notion more precise, we first define the notion of *shattering* a set of instances. Consider some subset of instances $S \subseteq X$. For example, Figure 7.3 shows a subset of three instances from X. Each hypothesis h from H imposes some dichotomy on S; that is, h partitions S into the two subsets $\{x \in S | h(x) = 1\}$ and $\{x \in S | h(x) = 0\}$. Given some instance set S, there are $2^{|S|}$ possible dichotomies, though H may be unable to represent some of these. We say that H shatters S if every possible dichotomy of S can be represented by some hypothesis from H.

> ***Definition:*** A set of instances S is **shattered** by hypothesis space H if and only if for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.

Figure 7.3 illustrates a set S of three instances that is shattered by the hypothesis space. Notice that each of the 2^3 dichotomies of these three instances is covered by some hypothesis.

Note that if a set of instances is not shattered by a hypothesis space, then there must be some concept (dichotomy) that can be defined over the instances, but that cannot be represented by the hypothesis space. The ability of H to shatter

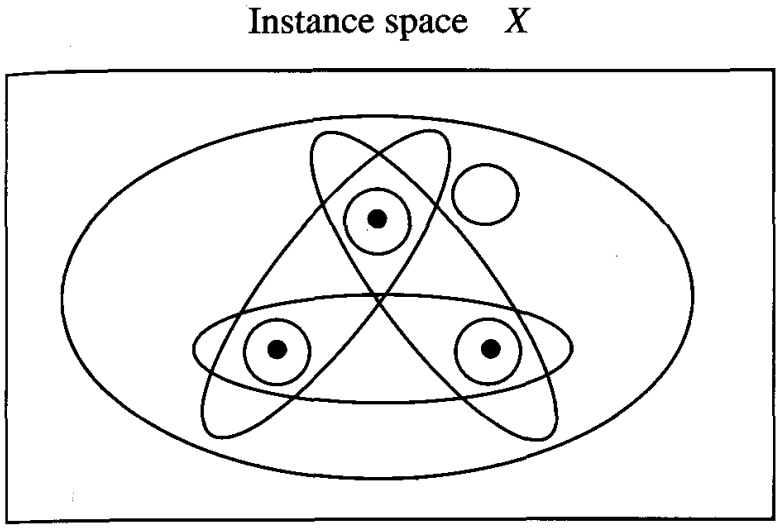


FIGURE 7.3
A set of three instances shattered by eight hypotheses. For every possible dichotomy of the instances, there exists a corresponding hypothesis.

a set of instances is thus a measure of its capacity to represent target concepts defined over these instances.

7.4.2 The Vapnik-Chervonenkis Dimension

The ability to shatter a set of instances is closely related to the inductive bias of a hypothesis space. Recall from Chapter 2 that an unbiased hypothesis space is one capable of representing every possible concept (dichotomy) definable over the instance space X. Put briefly, an unbiased hypothesis space H is one that shatters the instance space X. What if H cannot shatter X, but can shatter some large subset S of X? Intuitively, it seems reasonable to say that the larger the subset of X that can be shattered, the more expressive H. The VC dimension of H is precisely this measure.

> ***Definition:*** The **Vapnik-Chervonenkis dimension**, $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then $VC(H) \equiv \infty$.

Note that for any finite H, $VC(H) \leq \log_2 |H|$. To see this, suppose that $VC(H) = d$. Then H will require 2^d distinct hypotheses to shatter d instances. Hence, $2^d \leq |H|$, and $d = VC(H) \leq \log_2 |H|$.

7.4.2.1 ILLUSTRATIVE EXAMPLES

In order to develop an intuitive feeling for $VC(H)$, consider a few example hypothesis spaces. To get started, suppose the instance space X is the set of real numbers $X = \Re$ (e.g., describing the *height* of people), and H the set of intervals on the real number line. In other words, H is the set of hypotheses of the

form $a < x < b$, where a and b may be any real constants. What is $VC(H)$? To answer this question, we must find the largest subset of X that can be shattered by H. Consider a particular subset containing two distinct instances, say $S = \{3.1, 5.7\}$. Can S be shattered by H? Yes. For example, the four hypotheses $(1 < x < 2)$, $(1 < x < 4)$, $(4 < x < 7)$, and $(1 < x < 7)$ will do. Together, they represent each of the four dichotomies over S, covering neither instance, either one of the instances, and both of the instances, respectively. Since we have found a set of size two that can be shattered by H, we know the VC dimension of H is at least two. Is there a set of size three that can be shattered? Consider a set $S = \{x_0, x_1, x_2\}$ containing three arbitrary instances. Without loss of generality, assume $x_0 < x_1 < x_2$. Clearly this set cannot be shattered, because the dichotomy that includes x_0 and x_2, but not x_1, cannot be represented by a single closed interval. Therefore, *no* subset S of size three can be shattered, and $VC(H) = 2$. Note here that H is infinite, but $VC(H)$ finite.

Next consider the set X of instances corresponding to points on the x, y plane (see Figure 7.4). Let H be the set of all linear decision surfaces in the plane. In other words, H is the hypothesis space corresponding to a single perceptron unit with two inputs (see Chapter 4 for a general discussion of perceptrons). What is the VC dimension of this H? It is easy to see that any two distinct points in the plane can be shattered by H, because we can find four linear surfaces that include neither, either, or both points. What about sets of three points? As long as the points are not colinear, we will be able to find 2^3 linear surfaces that shatter them. Of course three colinear points cannot be shattered (for the same reason that the three points on the real line could not be shattered in the previous example). What is $VC(H)$ in this case—two or three? It is at least three. The definition of VC dimension indicates that if we find *any* set of instances of size d that can be shattered, then $VC(H) \geq d$. To show that $VC(H) < d$, we must show that no set of size d can be shattered. In this example, no sets of size four can be shattered, so $VC(H) = 3$. More generally, it can be shown that the VC dimension of linear decision surfaces in an r dimensional space (i.e., the VC dimension of a perceptron with r inputs) is $r + 1$.

As one final example, suppose each instance in X is described by the conjunction of exactly three boolean literals, and suppose that each hypothesis in H is described by the conjunction of up to three boolean literals. What is $VC(H)$? We

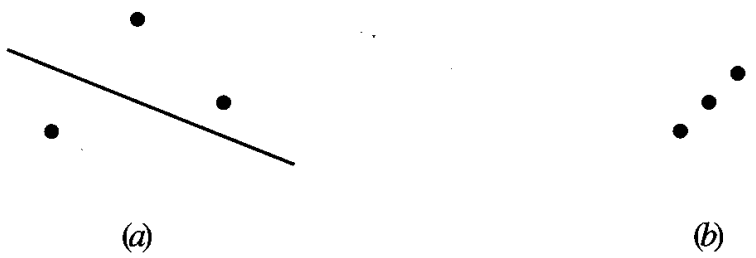


FIGURE 7.4
The VC dimension for linear decision surfaces in the x, y plane is 3. (*a*) A set of three points that can be shattered using linear decision surfaces. (*b*) A set of three that cannot be shattered.

can show that it is at least 3, as follows. Represent each instance by a 3-bit string corresponding to the values of each of its three literals l_1, l_2, and l_3. Consider the following set of three instances:

$instance_1$: 100
$instance_2$: 010
$instance_3$: 001

This set of three instances can be shattered by H, because a hypothesis can be constructed for any desired dichotomy as follows: If the dichotomy is to exclude $instance_i$, add the literal $\neg l_i$ to the hypothesis. For example, suppose we wish to include $instance_2$, but exclude $instance_1$ and $instance_3$. Then we use the hypothesis $\neg l_1 \wedge \neg l_3$. This argument easily extends from three features to n. Thus, the VC dimension for conjunctions of n boolean literals is at least n. In fact, it is exactly n, though showing this is more difficult, because it requires demonstrating that no set of $n + 1$ instances can be shattered.

7.4.3 Sample Complexity and the VC Dimension

Earlier we considered the question "How many randomly drawn training examples suffice to probably approximately learn any target concept in C?" (i.e., how many examples suffice to ϵ-exhaust the version space with probability $(1 - \delta)$?). Using $VC(H)$ as a measure for the complexity of H, it is possible to derive an alternative answer to this question, analogous to the earlier bound of Equation (7.2). This new bound (see Blumer et al. 1989) is

$$m \geq \frac{1}{\epsilon}(4 \log_2(2/\delta) + 8VC(H) \log_2(13/\epsilon)) \tag{7.7}$$

Note that just as in the bound from Equation (7.2), the number of required training examples m grows logarithmically in $1/\delta$. It now grows log times linear in $1/\epsilon$, rather than linearly. Significantly, the $\ln|H|$ term in the earlier bound has now been replaced by the alternative measure of hypothesis space complexity, $VC(H)$ (recall $VC(H) \leq \log_2|H|$).

Equation (7.7) provides an upper bound on the number of training examples sufficient to probably approximately learn any target concept in C, for any desired ϵ and δ. It is also possible to obtain a lower bound, as summarized in the following theorem (see Ehrenfeucht et al. 1989).

Theorem 7.3. Lower bound on sample complexity. Consider any concept class C such that $VC(C) \geq 2$, any learner L, and any $0 < \epsilon < \frac{1}{8}$, and $0 < \delta < \frac{1}{100}$. Then there exists a distribution $\mathcal{D}$ and target concept in C such that if L observes fewer examples than

$$\max\left[\frac{1}{\epsilon}\log(1/\delta), \frac{VC(C) - 1}{32\epsilon}\right]$$

then with probability at least δ, L outputs a hypothesis h having $error_{\mathcal{D}}(h) > \epsilon$.

This theorem states that if the number of training examples is too few, then no learner can PAC-learn every target concept in any nontrivial C. Thus, this theorem provides a lower bound on the number of training examples *necessary* for successful learning, complementing the earlier upper bound that gives a *sufficient* number. Notice this lower bound is determined by the complexity of the concept class C, whereas our earlier upper bounds were determined by H. (Why?)†

This lower bound shows that the upper bound of the inequality in Equation (7.7) is fairly tight. Both bounds are logarithmic in $1/\delta$ and linear in $VC(H)$. The only difference in the order of these two bounds is the extra $\log(1/\epsilon)$ dependence in the upper bound.

7.4.4 VC Dimension for Neural Networks

Given the discussion of artificial neural network learning in Chapter 4, it is interesting to consider how we might calculate the VC dimension of a network of interconnected units such as the feedforward networks trained by the BACKPROPAGATION procedure. This section presents a general result that allows computing the VC dimension of layered acyclic networks, based on the structure of the network and the VC dimension of its individual units. This VC dimension can then be used to bound the number of training examples sufficient to probably approximately correctly learn a feedforward network to desired values of ϵ and δ. This section may be skipped on a first reading without loss of continuity.

Consider a network, G, of units, which forms a layered directed acyclic graph. A *directed acyclic* graph is one for which the edges have a direction (e.g., the units have inputs and outputs), and in which there are no directed cycles. A *layered* graph is one whose nodes can be partitioned into layers such that all directed edges from nodes at layer l go to nodes at layer $l + 1$. The layered feedforward neural networks discussed throughout Chapter 4 are examples of such layered directed acyclic graphs.

It turns out that we can bound the VC dimension of such networks based on their graph structure and the VC dimension of the primitive units from which they are constructed. To formalize this, we must first define a few more terms. Let n be the number of inputs to the network G, and let us assume that there is just one output node. Let each internal unit N_i of G (i.e., each node that is not an input) have at most r inputs and implement a boolean-valued function $c_i : \Re^r \rightarrow \{0, 1\}$ from some function class C. For example, if the internal nodes are perceptrons, then C will be the class of linear threshold functions defined over $\Re^r$.

We can now define the *G-composition of C* to be the class of all functions that can be implemented by the network G assuming individual units in G take on functions from the class C. In brief, the G-composition of C is the hypothesis space representable by the network G.

---

†Hint: If we were to substitute H for C in the lower bound, this would result in a tighter bound on m in the case $H \supset C$.

The following theorem bounds the VC dimension of the G-composition of C, based on the VC dimension of C and the structure of G.

Theorem 7.4. VC-dimension of directed acyclic layered networks. (See Kearns and Vazirani 1994.) Let G be a layered directed acyclic graph with n input nodes and $s \geq 2$ internal nodes, each having at most r inputs. Let C be a concept class over $\Re^r$ of VC dimension d, corresponding to the set of functions that can be described by each of the s internal nodes. Let C_G be the G-composition of C, corresponding to the set of functions that can be represented by G. Then $VC(C_G) \leq 2ds \log(es)$, where e is the base of the natural logarithm.

Note this bound on the VC dimension of the network G grows linearly with the VC dimension d of its individual units and log times linear in s, the number of threshold units in the network.

Suppose we consider acyclic layered networks whose individual nodes are perceptrons. Recall from Chapter 4 that an r input perceptron uses linear decision surfaces to represent boolean functions over $\Re^r$. As noted in Section 7.4.2.1, the VC dimension of linear decision surfaces over $\Re^r$ is $r + 1$. Therefore, a single perceptron with r inputs has VC dimension $r + 1$. We can use this fact, together with the above theorem, to bound the VC dimension of acyclic layered networks containing s perceptrons, each with r inputs, as

$$\text{VC}(C_G^{perceptrons}) \leq 2(r + 1)s \log(es)$$

We can now bound the number m of training examples sufficient to learn (with probability at least $(1 - \delta)$) any target concept from $C_G^{perceptrons}$ to within error ϵ. Substituting the above expression for the network VC dimension into Equation (7.7), we have

$$\begin{aligned} m &\geq \frac{1}{\epsilon}(4 \log(2/\delta) + 8VC(H) \log(13/\epsilon)) \\ &\geq \frac{1}{\epsilon}(4 \log(2/\delta) + 16(r + 1)s \log(es) \log(13/\epsilon)) \end{aligned} \tag{7.8}$$

As illustrated by this perceptron network example, the above theorem is interesting because it provides a general method for bounding the VC dimension of layered, acyclic networks of units, based on the network structure and the VC dimension of the individual units. Unfortunately the above result does not directly apply to networks trained using BACKPROPAGATION, for two reasons. First, this result applies to networks of perceptrons rather than networks of *sigmoid units* to which the BACKPROPAGATION algorithm applies. Nevertheless, notice that the VC dimension of sigmoid units will be at least as great as that of perceptrons, because a sigmoid unit can approximate a perceptron to arbitrary accuracy by using sufficiently large weights. Therefore, the above bound on m will be at least as large for acyclic layered networks of sigmoid units. The second shortcoming of the above result is that it fails to account for the fact that BACKPROPAGATION

trains a network by beginning with near-zero weights, then iteratively modifying these weights until an acceptable hypothesis is found. Thus, BACKPROPAGATION with a cross-validation stopping criterion exhibits an inductive bias in favor of networks with small weights. This inductive bias, which reduces the effective VC dimension, is not captured by the above analysis.

7.5 THE MISTAKE BOUND MODEL OF LEARNING

While we have focused thus far on the PAC learning model, computational learning theory considers a variety of different settings and questions. Different learning settings that have been studied vary by how the training examples are generated (e.g., passive observation of random examples, active querying by the learner), noise in the data (e.g., noisy or error-free), the definition of success (e.g., the target concept must be learned exactly, or only probably and approximately), assumptions made by the learner (e.g., regarding the distribution of instances and whether $C \subseteq H$), and the measure according to which the learner is evaluated (e.g., number of training examples, number of mistakes, total time).

In this section we consider the *mistake bound* model of learning, in which the learner is evaluated by the total number of mistakes it makes before it converges to the correct hypothesis. As in the PAC setting, we assume the learner receives a sequence of training examples. However, here we demand that upon receiving each example x, the learner must predict the target value $c(x)$, before it is shown the correct target value by the trainer. The question considered is "How many *mistakes* will the learner make in its predictions before it learns the target concept?" This question is significant in practical settings where learning must be done while the system is in actual use, rather than during some off-line training stage. For example, if the system is to learn to predict which credit card purchases should be approved and which are fraudulent, based on data collected during use, then we are interested in minimizing the total number of mistakes it will make before converging to the correct target function. Here the total number of mistakes can be even more important than the total number of training examples.

This mistake bound learning problem may be studied in various specific settings. For example, we might count the number of mistakes made before PAC learning the target concept. In the examples below, we consider instead the number of mistakes made before learning the target concept *exactly*. Learning the target concept exactly means converging to a hypothesis such that $(\forall x)h(x) = c(x)$.

7.5.1 Mistake Bound for the FIND-S Algorithm

To illustrate, consider again the hypothesis space H consisting of conjunctions of up to n boolean literals $l_1 \ldots l_n$ and their negations (e.g., $Rich \wedge \neg Handsome$). Recall the FIND-S algorithm from Chapter 2, which incrementally computes the maximally specific hypothesis consistent with the training examples. A straightforward implementation of FIND-S for the hypothesis space H is as follows:

FIND-S:

- Initialize h to the most specific hypothesis $l_1 \wedge \neg l_1 \wedge l_2 \wedge \neg l_2 \ldots l_n \wedge \neg l_n$
- For each positive training instance x
  - Remove from h any literal that is not satisfied by x
- Output hypothesis h.

FIND-S converges in the limit to a hypothesis that makes no errors, provided $C \subseteq H$ and provided the training data is noise-free. FIND-S begins with the most specific hypothesis (which classifies every instance a negative example), then incrementally generalizes this hypothesis as needed to cover observed positive training examples. For the hypothesis representation used here, this generalization step consists of deleting unsatisfied literals.

Can we prove a bound on the total number of mistakes that FIND-S will make before exactly learning the target concept c? The answer is yes. To see this, note first that if $c \in H$, then FIND-S can never mistakenly classify a negative example as positive. The reason is that its current hypothesis h is always at least as specific as the target concept c. Therefore, to calculate the number of mistakes it will make, we need only count the number of mistakes it will make misclassifying truly positive examples as negative. How many such mistakes can occur before FIND-S learns c exactly? Consider the first positive example encountered by FIND-S. The learner will certainly make a mistake classifying this example, because its initial hypothesis labels every instance negative. However, the result will be that half of the $2n$ terms in its initial hypothesis will be eliminated, leaving only n terms. For each subsequent positive example that is mistakenly classified by the current hypothesis, at least one more of the remaining n terms must be eliminated from the hypothesis. Therefore, the total number of mistakes can be at most $n+1$. This number of mistakes will be required in the worst case, corresponding to learning the most general possible target concept $(\forall x)c(x) = 1$ and corresponding to a worst case sequence of instances that removes only one literal per mistake.

7.5.2 Mistake Bound for the HALVING Algorithm

As a second example, consider an algorithm that learns by maintaining a description of the version space, incrementally refining the version space as each new training example is encountered. The CANDIDATE-ELIMINATION algorithm and the LIST-THEN-ELIMINATE algorithm from Chapter 2 are examples of such algorithms. In this section we derive a worst-case bound on the number of mistakes that will be made by such a learner, for any finite hypothesis space H, assuming again that the target concept must be learned exactly.

To analyze the number of mistakes made while learning we must first specify precisely how the learner will make predictions given a new instance x. Let us assume this prediction is made by taking a majority vote among the hypotheses in the current version space. If the majority of version space hypotheses classify the new instance as positive, then this prediction is output by the learner. Otherwise a negative prediction is output.

This combination of learning the version space, together with using a majority vote to make subsequent predictions, is often called the HALVING algorithm. What is the maximum number of mistakes that can be made by the HALVING algorithm, for an arbitrary finite H, before it exactly learns the target concept? Notice that learning the target concept "exactly" corresponds to reaching a state where the version space contains only a single hypothesis (as usual, we assume the target concept c is in H).

To derive the mistake bound, note that the only time the HALVING algorithm can make a mistake is when the majority of hypotheses in its current version space incorrectly classify the new example. In this case, once the correct classification is revealed to the learner, the version space will be reduced to at most half its current size (i.e., only those hypotheses that voted with the minority will be retained). Given that each mistake reduces the size of the version space by at least half, and given that the initial version space contains only $|H|$ members, the maximum number of mistakes possible before the version space contains just one member is $\log_2 |H|$. In fact one can show the bound is $\lfloor \log_2 |H| \rfloor$. Consider, for example, the case in which $|H| = 7$. The first mistake must reduce $|H|$ to at most 3, and the second mistake will then reduce it to 1.

Note that $\lfloor \log_2 |H| \rfloor$ is a worst-case bound, and that it is possible for the HALVING algorithm to learn the target concept exactly without making any mistakes at all! This can occur because even when the majority vote is correct, the algorithm will remove the incorrect, minority hypotheses. If this occurs over the entire training sequence, then the version space may be reduced to a single member while making no mistakes along the way.

One interesting extension to the HALVING algorithm is to allow the hypotheses to vote with different weights. Chapter 6 describes the Bayes optimal classifier, which takes such a weighted vote among hypotheses. In the Bayes optimal classifier, the weight assigned to each hypothesis is the estimated posterior probability that it describes the target concept, given the training data. Later in this section we describe a different algorithm based on weighted voting, called the WEIGHTED-MAJORITY algorithm.

7.5.3 Optimal Mistake Bounds

The above analyses give worst-case mistake bounds for two specific algorithms: FIND-S and CANDIDATE-ELIMINATION. It is interesting to ask what is the optimal mistake bound for an arbitrary concept class C, assuming $H = C$. By optimal mistake bound we mean the lowest worst-case mistake bound over all possible learning algorithms. To be more precise, for any learning algorithm A and any target concept c, let $M_A(c)$ denote the maximum over all possible sequences of training examples of the number of mistakes made by A to exactly learn c. Now for any nonempty concept class C, let $M_A(C) \equiv \max_{c \in C} M_A(c)$. Note that above we showed $M_{Find-S}(C) = n + 1$ when C is the concept class described by up to n boolean literals. We also showed $M_{Halving}(C) \leq log_2(|C|)$ for any concept class C.

We define the optimal mistake bound for a concept class C below.

Definition: Let C be an arbitrary nonempty concept class. The **optimal mistake bound** for C, denoted $Opt(C)$, is the minimum over all possible learning algorithms A of $M_A(C)$.

$$Opt(C) \equiv \min_{A \in learning\ algorithms} M_A(C)$$

Speaking informally, this definition states that $Opt(C)$ is the number of mistakes made for the hardest target concept in C, using the hardest training sequence, by the best algorithm. Littlestone (1987) shows that for any concept class C, there is an interesting relationship among the optimal mistake bound for C, the bound of the HALVING algorithm, and the VC dimension of C, namely

$$VC(C) \leq Opt(C) \leq M_{Halving}(C) \leq log_2(|C|)$$

Furthermore, there exist concept classes for which the four quantities above are exactly equal. One such concept class is the powerset C_P of any finite set of instances X. In this case, $VC(C_P) = |X| = \log_2(|C_P|)$, so all four quantities must be equal. Littlestone (1987) provides examples of other concept classes for which $VC(C)$ is strictly less than $Opt(C)$ and for which $Opt(C)$ is strictly less than $M_{Halving}(C)$.

7.5.4 WEIGHTED-MAJORITY Algorithm

In this section we consider a generalization of the HALVING algorithm called the WEIGHTED-MAJORITY algorithm. The WEIGHTED-MAJORITY algorithm makes predictions by taking a weighted vote among a pool of prediction algorithms and learns by altering the weight associated with each prediction algorithm. These prediction algorithms can be taken to be the alternative hypotheses in H, or they can be taken to be alternative learning algorithms that themselves vary over time. All that we require of a prediction algorithm is that it predict the value of the target concept, given an instance. One interesting property of the WEIGHTED-MAJORITY algorithm is that it is able to accommodate inconsistent training data. This is because it does not eliminate a hypothesis that is found to be inconsistent with some training example, but rather reduces its weight. A second interesting property is that we can bound the number of mistakes made by WEIGHTED-MAJORITY in terms of the number of mistakes committed by the best of the pool of prediction algorithms.

The WEIGHTED-MAJORITY algorithm begins by assigning a weight of 1 to each prediction algorithm, then considers the training examples. Whenever a prediction algorithm misclassifies a new training example its weight is decreased by multiplying it by some number β, where $0 \leq \beta < 1$. The exact definition of the WEIGHTED-MAJORITY algorithm is given in Table 7.1.

Notice if $\beta = 0$ then WEIGHTED-MAJORITY is identical to the HALVING algorithm. On the other hand, if we choose some other value for β, no prediction

a_i denotes the i^{th} prediction algorithm in the pool A of algorithms. w_i denotes the weight associated with a_i.

- For all i initialize $w_i \leftarrow 1$
- For each training example $\langle x, c(x) \rangle$
  - Initialize q_0 and q_1 to 0
  - For each prediction algorithm a_i
    - If $a_i(x) = 0$ then $q_0 \leftarrow q_0 + w_i$
      If $a_i(x) = 1$ then $q_1 \leftarrow q_1 + w_i$
- If $q_1 > q_0$ then predict $c(x) = 1$
  If $q_0 > q_1$ then predict $c(x) = 0$
  If $q_1 = q_0$ then predict 0 or 1 at random for $c(x)$
- For each prediction algorithm a_i in A do
  If $a_i(x) \neq c(x)$ then $w_i \leftarrow \beta w_i$

TABLE 7.1
WEIGHTED-MAJORITY algorithm.

algorithm will ever be eliminated completely. If an algorithm misclassifies a training example, it will simply receive a smaller vote in the future.

We now show that the number of mistakes committed by the WEIGHTED-MAJORITY algorithm can be bounded in terms of the number of mistakes made by the best prediction algorithm in the voting pool.

Theorem 7.5. Relative mistake bound for WEIGHTED-MAJORITY. Let D be any sequence of training examples, let A be any set of n prediction algorithms, and let k be the minimum number of mistakes made by any algorithm in A for the training sequence D. Then the number of mistakes over D made by the WEIGHTED-MAJORITY algorithm using $\beta = \frac{1}{2}$ is at most

$$2.4(k + \log_2 n)$$

Proof. We prove the theorem by comparing the final weight of the best prediction algorithm to the sum of weights over all algorithms. Let a_j denote an algorithm from A that commits the optimal number k of mistakes. The final weight w_j associated with a_j will be $(\frac{1}{2})^k$, because its initial weight is 1 and it is multiplied by $\frac{1}{2}$ for each mistake. Now consider the sum $W = \sum_{i=1}^{n} w_i$ of the weights associated with all n algorithms in A. W is initially n. For each mistake made by WEIGHTED-MAJORITY, W is reduced to at most $\frac{3}{4}W$. This is the case because the algorithms voting in the weighted majority must hold at least half of the total weight W, and this portion of W will be reduced by a factor of $\frac{1}{2}$. Let M denote the total number of mistakes committed by WEIGHTED-MAJORITY for the training sequence D. Then the final total weight W is at most $n(\frac{3}{4})^M$. Because the final weight w_j cannot be greater than the final total weight, we have

$$\left(\frac{1}{2}\right)^k \leq n\left(\frac{3}{4}\right)^M$$

Rearranging terms yields

$$M \leq \frac{(k + \log_2 n)}{-\log_2\left(\frac{3}{4}\right)} \leq 2.4(k + \log_2 n)$$

which proves the theorem. □

To summarize, the above theorem states that the number of mistakes made by the WEIGHTED-MAJORITY algorithm will never be greater than a constant factor times the number of mistakes made by the best member of the pool, plus a term that grows only logarithmically in the size of the pool.

This theorem is generalized by Littlestone and Warmuth (1991), who show that for an arbitrary $0 \leq \beta < 1$ the above bound is

$$\frac{k \log_2 \frac{1}{\beta} + \log_2 n}{\log_2 \frac{2}{1+\beta}}$$

7.6 SUMMARY AND FURTHER READING

The main points of this chapter include:

- The probably approximately correct (PAC) model considers algorithms that learn target concepts from some concept class C, using training examples drawn at random according to an unknown, but fixed, probability distribution. It requires that the learner probably (with probability at least $[1 - \delta]$) learn a hypothesis that is approximately (within error ϵ) correct, given computational effort and training examples that grow only polynomially with $1/\epsilon$, $1/\delta$, the size of the instances, and the size of the target concept.
- Within the setting of the PAC learning model, any consistent learner using a finite hypothesis space H where $C \subseteq H$ will, with probability $(1 - \delta)$, output a hypothesis within error ϵ of the target concept, after observing m randomly drawn training examples, as long as

  $$m \geq \frac{1}{\epsilon}(\ln(1/\delta) + \ln|H|)$$

  This gives a bound on the number of training examples sufficient for successful learning under the PAC model.
- One constraining assumption of the PAC learning model is that the learner knows in advance some restricted concept class C that contains the target concept to be learned. In contrast, the *agnostic learning* model considers the more general setting in which the learner makes no assumption about the class from which the target concept is drawn. Instead, the learner outputs the hypothesis from H that has the least error (possibly nonzero) over the training data. Under this less restrictive agnostic learning model, the learner is assured with probability $(1-\delta)$ to output a hypothesis within error ϵ of the

best possible hypothesis in H, after observing m randomly drawn training examples, provided

$$m \geq \frac{1}{2\epsilon^2}(\ln(1/\delta) + \ln|H|)$$

- The number of training examples required for successful learning is strongly influenced by the complexity of the hypothesis space considered by the learner. One useful measure of the complexity of a hypothesis space H is its Vapnik-Chervonenkis dimension, $VC(H)$. $VC(H)$ is the size of the largest subset of instances that can be shattered (split in all possible ways) by H.
- An alternative upper bound on the number of training examples sufficient for successful learning under the PAC model, stated in terms of $VC(H)$ is

$$m \geq \frac{1}{\epsilon}(4\log_2(2/\delta) + 8VC(H)\log_2(13/\epsilon))$$

A lower bound is

$$m \geq \max\left[\frac{1}{\epsilon}\log(1/\delta), \frac{VC(C)-1}{32\epsilon}\right]$$

- An alternative learning model, called the *mistake bound model*, is used to analyze the number of training examples a learner will misclassify before it exactly learns the target concept. For example, the HALVING algorithm will make at most $\lfloor \log_2 |H| \rfloor$ mistakes before exactly learning any target concept drawn from H. For an arbitrary concept class C, the best worst-case algorithm will make $Opt(C)$ mistakes, where

$$VC(C) \leq Opt(C) \leq \log_2(|C|)$$

- The WEIGHTED-MAJORITY algorithm combines the weighted votes of multiple prediction algorithms to classify new instances. It learns weights for each of these prediction algorithms based on errors made over a sequence of examples. Interestingly, the number of mistakes made by WEIGHTED-MAJORITY can be bounded in terms of the number of mistakes made by the best prediction algorithm in the pool.

Much early work on computational learning theory dealt with the question of whether the learner could identify the target concept in the limit, given an indefinitely long sequence of training examples. The identification in the limit model was introduced by Gold (1967). A good overview of results in this area is (Angluin 1992). Vapnik (1982) examines in detail the problem of uniform convergence, and the closely related PAC-learning model was introduced by Valiant (1984). The discussion in this chapter of ϵ-exhausting the version space is based on Haussler's (1988) exposition. A useful collection of results under the PAC model can be found in Blumer et al. (1989). Kearns and Vazirani (1994) provide an excellent exposition of many results from computational learning theory. Earlier texts in this area include Anthony and Biggs (1992) and Natarajan (1991).

Current research on computational learning theory covers a broad range of learning models and learning algorithms. Much of this research can be found in the proceedings of the annual conference on Computational Learning Theory (COLT). Several special issues of the journal *Machine Learning* have also been devoted to this topic.

EXERCISES

7.1. Consider training a two-input perceptron. Give an upper bound on the number of training examples sufficient to assure with 90% confidence that the learned perceptron will have true error of at most 5%. Does this bound seem realistic?

7.2. Consider the class C of concepts of the form $(a \leq x \leq b) \wedge (c \leq y \leq d)$, where a, b, c, and d are integers in the interval (0, 99). Note each concept in this class corresponds to a rectangle with integer-valued boundaries on a portion of the x, y plane. Hint: Given a region in the plane bounded by the points $(0, 0)$ and $(n-1, n-1)$, the number of distinct rectangles with integer-valued boundaries within this region is $(\frac{n(n+1)}{2})^2$.

(*a*) Give an upper bound on the number of randomly drawn training examples sufficient to assure that for any target concept c in C, any consistent learner using $H = C$ will, with probability 95%, output a hypothesis with error at most .15.

(*b*) Now suppose the rectangle boundaries a, b, c, and d take on *real* values instead of integer values. Update your answer to the first part of this question.

7.3. In this chapter we derived an expression for the number of training examples sufficient to ensure that every hypothesis will have true error no worse than ϵ plus its observed training error $error_D(h)$. In particular, we used Hoeffding bounds to derive Equation (7.3). Derive an alternative expression for the number of training examples sufficient to ensure that every hypothesis will have true error no worse than $(1+\gamma)error_D(h)$. You can use the general Chernoff bounds to derive such a result.

Chernoff bounds: Suppose $X_1, \ldots, X_m$ are the outcomes of m independent coin flips (Bernoulli trials), where the probability of heads on any single trial is $\Pr[X_i = 1] = p$ and the probability of tails is $\Pr[X_i = 0] = 1 - p$. Define $S = X_1 + X_2 + \cdots + X_m$ to be the sum of the outcomes of these m trials. The expected value of S/m is $E[S/m] = p$. The Chernoff bounds govern the probability that S/m will differ from p by some factor $0 \leq \gamma \leq 1$.

$$\Pr[S/m > (1+\gamma)p] \leq e^{-mp\gamma^2/3}$$

$$\Pr[S/m < (1-\gamma)p] \leq e^{-mp\gamma^2/2}$$

7.4. Consider a learning problem in which $X = \Re$ is the set of real numbers, and $C = H$ is the set of intervals over the reals, $H = \{(a < x < b) \mid a, b \in \Re\}$. What is the probability that a hypothesis consistent with m examples of this target concept will have error at least ϵ? Solve this using the VC dimension. Can you find a second way to solve this, based on first principles and ignoring the VC dimension?

7.5. Consider the space of instances X corresponding to all points in the x, y plane. Give the VC dimension of the following hypothesis spaces:

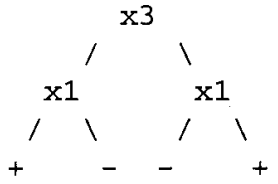
(a) H_r = the set of all rectangles in the x, y plane. That is, $H = \{((a < x < b) \wedge (c < y < d)) | a, b, c, d \in \Re\}$.

(b) $H_c =$ circles in the x, y plane. Points inside the circle are classified as positive examples

(c) $H_t =$triangles in the x, y plane. Points inside the triangle are classified as positive examples

7.6. Write a consistent learner for H_r from Exercise 7.5. Generate a variety of target concept rectangles at random, corresponding to different rectangles in the plane. Generate random examples of each of these target concepts, based on a uniform distribution of instances within the rectangle from $\langle 0, 0 \rangle$ to $\langle 100, 100 \rangle$. Plot the generalization error as a function of the number of training examples, m. On the same graph, plot the theoretical relationship between ϵ and m, for $\delta = .95$. Does theory fit experiment?

7.7. Consider the hypothesis class H_{rd2} of "regular, depth-2 decision trees" over n Boolean variables. A "regular, depth-2 decision tree" is a depth-2 decision tree (a tree with four leaves, all distance 2 from the root) in which the left and right child of the root are *required to contain the same variable.* For instance, the following tree is in H_{rd2}.



(a) As a function of n, how many syntactically distinct trees are there in H_{rd2}?

(b) Give an upper bound for the number of examples needed in the PAC model to learn H_{rd2} with error ϵ and confidence δ.

(c) Consider the following WEIGHTED-MAJORITY algorithm, for the class H_{rd2}. You begin with all hypotheses in H_{rd2} assigned an initial weight equal to 1. Every time you see a new example, you predict based on a weighted majority vote over all hypotheses in H_{rd2}. Then, instead of eliminating the inconsistent trees, you cut down their weight by a factor of 2. How many mistakes will this procedure make at most, as a function of n and the number of mistakes of the best tree in H_{rd2}?

7.8. This question considers the relationship between the PAC analysis considered in this chapter and the evaluation of hypotheses discussed in Chapter 5. Consider a learning task in which instances are described by n boolean variables (e.g., $x_1 \wedge \bar{x_2} \wedge x_3 \ldots \bar{x_n}$) and are drawn according to a fixed but unknown probability distribution $\mathcal{D}$. The target concept is known to be describable by a conjunction of boolean attributes and their negations (e.g., $x_2 \wedge \bar{x_5}$), and the learning algorithm uses this concept class as its hypothesis space H. A consistent learner is provided a set of 100 training examples drawn according to $\mathcal{D}$. It outputs a hypothesis h from H that is consistent with all 100 examples (i.e., the error of h over these training examples is zero).

(a) We are interested in the true error of h, that is, the probability that it will misclassify future instances drawn randomly according to $\mathcal{D}$. Based on the above information, can you give an interval into which this true error will fall with at least 95% probability? If so, state it and justify it briefly. If not, explain the difficulty.

(*b*) You now draw a new set of 100 instances, drawn independently according to the same distribution $\mathcal{D}$. You find that h misclassifies 30 of these 100 new examples. Can you give an interval into which this true error will fall with approximately 95% probability? (Ignore the performance over the earlier training data for this part.) If so, state it and justify it briefly. If not, explain the difficulty.

(*c*) It may seem a bit odd that h misclassifies 30% of the new examples even though it perfectly classified the training examples. Is this event more likely for large n or small n? Justify your answer in a sentence.

REFERENCES

Angluin, D. (1992). Computational learning theory: Survey and selected bibliography. *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing* (pp. 351–369). ACM Press.

Angluin, D., Frazier, M., & Pitt, L. (1992). Learning conjunctions of horn clauses. *Machine Learning*, 9, 147–164.

Anthony, M., & Biggs, N. (1992). *Computational learning theory: An introduction.* Cambridge, England: Cambridge University Press.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4) (October), 929–965.

Ehrenfeucht, A., Haussler, D., Kearns, M., & Valiant, L. (1989). A general lower bound on the number of examples needed for learning. *Information and Computation*, 82, 247–261.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10, 447–474.

Goldman, S. (Ed.). (1995). Special issue on computational learning theory. *Machine Learning*, 18(2/3), February.

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36, 177–221.

Kearns, M. J., & Vazirani, U. V. (1994). *An introduction to computational learning theory.* Cambridge, MA: MIT Press.

Laird, P. (1988). *Learning from good and bad data.* Dordrecht: Kluwer Academic Publishers.

Li, M., & Valiant, L. G. (Eds.). (1994). Special issue on computational learning theory. *Machine Learning*, 14(1).

Littlestone, N. (1987). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2, 285–318.

Littlestone, N., & Warmuth, M. (1991). *The weighted majority algorithm* (Technical report UCSC-CRL-91-28). Univ. of California Santa Cruz, Computer Engineering and Information Sciences Dept., Santa Cruz, CA.

Littlestone, N., & Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation* (108), 212–261.

Pitt, L. (Ed.). (1990). Special issue on computational learning theory. *Machine Learning*, 5(2).

Natarajan, B. K. (1991). *Machine learning: A theoretical approach.* San Mateo, CA: Morgan Kaufmann.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27(11), 1134–1142.

Vapnik, V. N. (1982). *Estimation of dependences based on empirical data.* New York: Springer-Verlag.

Vapnik, V. N., & Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16, 264–280.

CHAPTER 8

INSTANCE-BASED LEARNING

In contrast to learning methods that construct a general, explicit description of the target function when training examples are provided, instance-based learning methods simply store the training examples. Generalizing beyond these examples is postponed until a new instance must be classified. Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value for the new instance. Instance-based learning includes nearest neighbor and locally weighted regression methods that assume instances can be represented as points in a Euclidean space. It also includes case-based reasoning methods that use more complex, symbolic representations for instances. Instance-based methods are sometimes referred to as "lazy" learning methods because they delay processing until a new instance must be classified. A key advantage of this kind of delayed, or lazy, learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.

8.1 INTRODUCTION

Instance-based learning methods such as nearest neighbor and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions. Learning in these algorithms consists of simply storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the

new query instance. One key difference between these approaches and the methods discussed in other chapters is that instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified. In fact, many techniques construct only a local approximation to the target function that applies in the neighborhood of the new query instance, and never construct an approximation designed to perform well over the entire instance space. This has significant advantages when the target function is very complex, but can still be described by a collection of less complex local approximations.

Instance-based methods can also use more complex, symbolic representations for instances. In case-based learning, instances are represented in this fashion and the process for identifying "neighboring" instances is elaborated accordingly. Case-based reasoning has been applied to tasks such as storing and reusing past experience at a help desk, reasoning about legal cases by referring to previous cases, and solving complex scheduling problems by reusing relevant portions of previously solved problems.

One disadvantage of instance-based approaches is that the cost of classifying new instances can be high. This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered. Therefore, techniques for efficiently indexing training examples are a significant practical issue in reducing the computation required at query time. A second disadvantage to many instance-based approaches, especially nearest-neighbor approaches, is that they typically consider *all* attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

In the next section we introduce the *k*-NEAREST NEIGHBOR learning algorithm, including several variants of this widely-used approach. The subsequent section discusses locally weighted regression, a learning method that constructs local approximations to the target function and that can be viewed as a generalization of *k*-NEAREST NEIGHBOR algorithms. We then describe radial basis function networks, which provide an interesting bridge between instance-based and neural network learning algorithms. The next section discusses case-based reasoning, an instance-based approach that employs symbolic representations and knowledge-based inference. This section includes an example application of case-based reasoning to a problem in engineering design. Finally, we discuss the fundamental differences in capabilities that distinguish lazy learning methods discussed in this chapter from eager learning methods discussed in the other chapters of this book.

8.2 *k*-NEAREST NEIGHBOR LEARNING

The most basic instance-based method is the *k*-NEAREST NEIGHBOR algorithm. This algorithm assumes all instances correspond to points in the n-dimensional space $\Re^n$. The nearest neighbors of an instance are defined in terms of the standard

Euclidean distance. More precisely, let an arbitrary instance x be described by the feature vector

$$\langle a_1(x), a_2(x), \ldots a_n(x) \rangle$$

where $a_r(x)$ denotes the value of the rth attribute of instance x. Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$, where

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

In nearest-neighbor learning the target function may be either discrete-valued or real-valued. Let us first consider learning discrete-valued target functions of the form $f : \Re^n \rightarrow V$, where V is the finite set $\{v_1, \ldots v_s\}$. The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued target function is given in Table 8.1. As shown there, the value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of f among the k training examples nearest to x_q. If we choose $k = 1$, then the 1-NEAREST NEIGHBOR algorithm assigns to $\hat{f}(x_q)$ the value $f(x_i)$ where x_i is the training instance nearest to x_q. For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

Figure 8.1 illustrates the operation of the k-NEAREST NEIGHBOR algorithm for the case where the instances are points in a two-dimensional space and where the target function is boolean valued. The positive and negative training examples are shown by "+" and "−" respectively. A query point x_q is shown as well. Note the 1-NEAREST NEIGHBOR algorithm classifies x_q as a positive example in this figure, whereas the 5-NEAREST NEIGHBOR algorithm classifies it as a negative example.

What is the nature of the hypothesis space H implicitly considered by the k-NEAREST NEIGHBOR algorithm? Note the k-NEAREST NEIGHBOR algorithm never forms an explicit general hypothesis $\hat{f}$ regarding the target function f. It simply computes the classification of each new query instance as needed. Nevertheless,

---

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

TABLE 8.1
The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

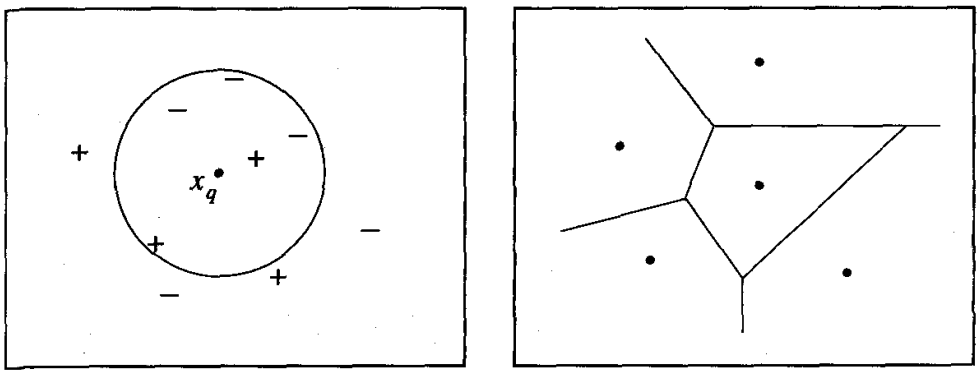


FIGURE 8.1
k-NEAREST NEIGHBOR. A set of positive and negative training examples is shown on the left, along with a query instance x_q to be classified. The 1-NEAREST NEIGHBOR algorithm classifies x_q positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEAREST NEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point (i.e., the instances for which the 1-NEAREST NEIGHBOR algorithm will assign the classification belonging to that training example).

we can still ask what the implicit general function is, or what classifications would be assigned if we were to hold the training examples constant and query the algorithm with every possible instance in X. The diagram on the right side of Figure 8.1 shows the shape of this decision surface induced by 1-NEAREST NEIGHBOR over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples. For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the *Voronoi diagram* of the set of training examples.

The *k*-NEAREST NEIGHBOR algorithm is easily adapted to approximating continuous-valued target functions. To accomplish this, we have the algorithm calculate the mean value of the k nearest training examples rather than calculate their most common value. More precisely, to approximate a real-valued target function $f : \Re^n \rightarrow \Re$ we replace the final line of the above algorithm by the line

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k} \tag{8.1}$$

8.2.1 Distance-Weighted NEAREST NEIGHBOR Algorithm

One obvious refinement to the *k*-NEAREST NEIGHBOR algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point x_q, giving greater weight to closer neighbors. For example, in the algorithm of Table 8.1, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from x_q.

This can be accomplished by replacing the final line of the algorithm by

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i)) \tag{8.2}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2} \tag{8.3}$$

To accommodate the case where the query point x_q exactly matches one of the training instances x_i and the denominator $d(x_q, x_i)^2$ is therefore zero, we assign $\hat{f}(x_q)$ to be $f(x_i)$ in this case. If there are several such training examples, we assign the majority classification among them.

We can distance-weight the instances for real-valued target functions in a similar fashion, replacing the final line of the algorithm in this case by

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \tag{8.4}$$

where w_i is as defined in Equation (8.3). Note the denominator in Equation (8.4) is a constant that normalizes the contributions of the various weights (e.g., it assures that if $f(x_i) = c$ for all training examples, then $\hat{f}(x_q) \leftarrow c$ as well).

Note all of the above variants of the k-NEAREST NEIGHBOR algorithm consider only the k nearest neighbors to classify the query point. Once we add distance weighting, there is really no harm in allowing all training examples to have an influence on the classification of the x_q, because very distant examples will have very little effect on $\hat{f}(x_q)$. The only disadvantage of considering all examples is that our classifier will run more slowly. If all training examples are considered when classifying a new query instance, we call the algorithm a *global* method. If only the nearest training examples are considered, we call it a *local* method. When the rule in Equation (8.4) is applied as a global method, using all training examples, it is known as Shepard's method (Shepard 1968).

8.2.2 Remarks on k-NEAREST NEIGHBOR Algorithm

The distance-weighted k-NEAREST NEIGHBOR algorithm is a highly effective inductive inference method for many practical problems. It is robust to noisy training data and quite effective when it is provided a sufficiently large set of training data. Note that by taking the weighted average of the k neighbors nearest to the query point, it can smooth out the impact of isolated noisy training examples.

What is the inductive bias of k-NEAREST NEIGHBOR? The basis for classifying new query points is easily understood based on the diagrams in Figure 8.1. The inductive bias corresponds to an assumption that the classification of an instance x_q will be most similar to the classification of other instances that are nearby in Euclidean distance.

One practical issue in applying k-NEAREST NEIGHBOR algorithms is that the distance between instances is calculated based on *all* attributes of the instance

(i.e., on all axes in the Euclidean space containing the instances). This lies in contrast to methods such as rule and decision tree learning systems that select only a subset of the instance attributes when forming the hypothesis. To see the effect of this policy, consider applying k-NEAREST NEIGHBOR to a problem in which each instance is described by 20 attributes, but where only 2 of these attributes are relevant to determining the classification for the particular target function. In this case, instances that have identical values for the 2 relevant attributes may nevertheless be distant from one another in the 20-dimensional instance space. As a result, the similarity metric used by k-NEAREST NEIGHBOR—depending on all 20 attributes—will be misleading. The distance between neighbors will be dominated by the large number of irrelevant attributes. This difficulty, which arises when many irrelevant attributes are present, is sometimes referred to as the *curse of dimensionality*. Nearest-neighbor approaches are especially sensitive to this problem.

One interesting approach to overcoming this problem is to weight each attribute differently when calculating the distance between two instances. This corresponds to stretching the axes in the Euclidean space, shortening the axes that correspond to less relevant attributes, and lengthening the axes that correspond to more relevant attributes. The amount by which each axis should be stretched can be determined automatically using a cross-validation approach. To see how, first note that we wish to stretch (multiply) the jth axis by some factor z_j, where the values $z_1 \ldots z_n$ are chosen to minimize the true classification error of the learning algorithm. Second, note that this true error can be estimated using cross-validation. Hence, one algorithm is to select a random subset of the available data to use as training examples, then determine the values of $z_1 \ldots z_n$ that lead to the minimum error in classifying the remaining examples. By repeating this process multiple times the estimate for these weighting factors can be made more accurate. This process of stretching the axes in order to optimize the performance of k-NEAREST NEIGHBOR provides a mechanism for suppressing the impact of irrelevant attributes.

An even more drastic alternative is to completely eliminate the least relevant attributes from the instance space. This is equivalent to setting some of the z_i scaling factors to zero. Moore and Lee (1994) discuss efficient cross-validation methods for selecting relevant subsets of the attributes for k-NEAREST NEIGHBOR algorithms. In particular, they explore methods based on leave-one-out cross-validation, in which the set of m training instances is repeatedly divided into a training set of size $m-1$ and test set of size 1, in all possible ways. This leave-one-out approach is easily implemented in k-NEAREST NEIGHBOR algorithms because no additional training effort is required each time the training set is redefined. Note both of the above approaches can be seen as stretching each axis by some constant factor. Alternatively, we could stretch each axis by a value that varies over the instance space. However, as we increase the number of degrees of freedom available to the algorithm for redefining its distance metric in such a fashion, we also increase the risk of overfitting. Therefore, the approach of locally stretching the axes is much less common.

One additional practical issue in applying k-NEAREST NEIGHBOR is efficient memory indexing. Because this algorithm delays all processing until a new query is received, significant computation can be required to process each new query. Various methods have been developed for indexing the stored training examples so that the nearest neighbors can be identified more efficiently at some additional cost in memory. One such indexing method is the *kd*-tree (Bentley 1975; Friedman et al. 1977), in which instances are stored at the leaves of a tree, with nearby instances stored at the same or nearby nodes. The internal nodes of the tree sort the new query x_q to the relevant leaf by testing selected attributes of x_q.

8.2.3 A Note on Terminology

Much of the literature on nearest-neighbor methods and weighted local regression uses a terminology that has arisen from the field of statistical pattern recognition. In reading that literature, it is useful to know the following terms:

- *Regression* means approximating a real-valued target function.
- *Residual* is the error $\hat{f}(x) - f(x)$ in approximating the target function.
- *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$.

8.3 LOCALLY WEIGHTED REGRESSION

The nearest-neighbor approaches described in the previous section can be thought of as approximating the target function $f(x)$ at the single query point $x = x_q$. Locally weighted regression is a generalization of this approach. It constructs an explicit approximation to f over a local region surrounding x_q. Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to f. For example, we might approximate the target function in the neighborhood surrounding x_q using a linear function, a quadratic function, a multilayer neural network, or some other functional form. The phrase "locally weighted regression" is called *local* because the function is approximated based only on data near the query point, *weighted* because the contribution of each training example is weighted by its distance from the query point, and *regression* because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Given a new query instance x_q, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding x_q. This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance. The description of $\hat{f}$ may then be deleted, because a different local approximation will be calculated for each distinct query instance.

8.3.1 Locally Weighted Linear Regression

Let us consider the case of locally weighted regression in which the target function f is approximated near x_q using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

As before, $a_i(x)$ denotes the value of the ith attribute of the instance x.

Recall that in Chapter 4 we discussed methods such as gradient descent to find the coefficients $w_0 \ldots w_n$ to minimize the error in fitting such linear functions to a given set of training examples. In that chapter we were interested in a global approximation to the target function. Therefore, we derived methods to choose weights that minimize the squared error summed over the set D of training examples

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \tag{8.5}$$

which led us to the gradient descent training rule

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x) \tag{8.6}$$

where η is a constant learning rate, and where the training rule has been re-expressed from the notation of Chapter 4 to fit our current notation (i.e., $t \rightarrow f(x)$, $o \rightarrow \hat{f}(x)$, and $x_j \rightarrow a_j(x)$).

How shall we modify this procedure to derive a local approximation rather than a global one? The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below. Note we write the error $E(x_q)$ to emphasize the fact that now the error is being defined as a function of the query point x_q.

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

Criterion two is perhaps the most esthetically pleasing because it allows every training example to have an impact on the classification of x_q. However,

this approach requires computation that grows linearly with the number of training examples. Criterion three is a good approximation to criterion two and has the advantage that computational cost is independent of the total number of training examples; its cost depends only on the number k of neighbors considered.

If we choose criterion three above and rederive the gradient descent rule using the same style of argument as in Chapter 4, we obtain the following training rule (see Exercise 8.1):

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x))\ (f(x) - \hat{f}(x))\ a_j(x) \qquad (8.7)$$

Notice the only differences between this new rule and the rule given by Equation (8.6) are that the contribution of instance x to the weight update is now multiplied by the distance penalty $K(d(x_q, x))$, and that the error is summed over only the k nearest training examples. In fact, if we are fitting a linear function to a fixed set of training examples, then methods much more efficient than gradient descent are available to directly solve for the desired coefficients $w_0 \ldots w_n$. Atkeson et al. (1997a) and Bishop (1995) survey several such methods.

8.3.2 Remarks on Locally Weighted Regression

Above we considered using a linear function to approximate f in the neighborhood of the query instance x_q. The literature on locally weighted regression contains a broad range of alternative methods for distance weighting the training examples, and a range of methods for locally approximating the target function. In most cases, the target function is approximated by a constant, linear, or quadratic function. More complex functional forms are not often found because (1) the cost of fitting more complex functions for each query instance is prohibitively high, and (2) these simple approximations model the target function quite well over a sufficiently small subregion of the instance space.

8.4 RADIAL BASIS FUNCTIONS

One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions (Powell 1987; Broomhead and Lowe 1988; Moody and Darken 1989). In this approach, the learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x)) \qquad (8.8)$$

where each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases. Here k is a user-provided constant that specifies the number of kernel functions to be included. Even though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u. It is common