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

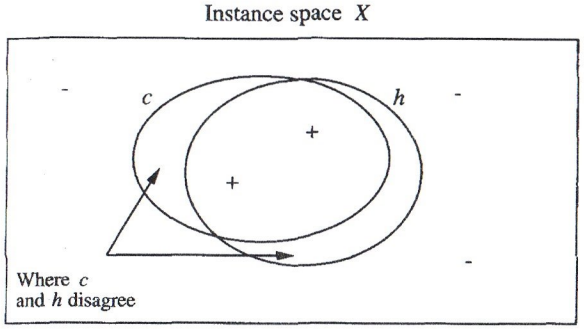

FIGURE 7.1
The error of hypothesis h with respect to target concept c. The error of h with respect to c is the probability that a randomly drawn instance will fall into the region where h and c disagree on its classification. The + and − points indicate positive and negative training examples. Note h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

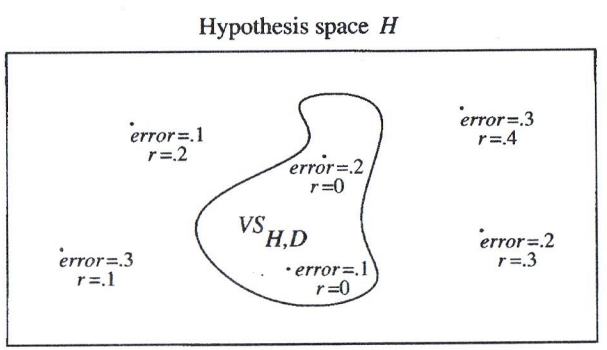

FIGURE 7.2
Exhausting the version space. The version space $VS_{H,D}$ is the subset of hypotheses $h \in H$, which have zero training error (denoted by $r = 0$ in the figure). Of course the true $error_{\mathcal{D}}(h)$ (denoted by *error* in the figure) may be nonzero, even for hypotheses that commit 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

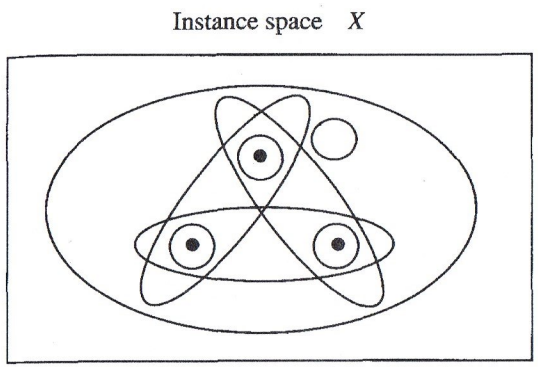

FIGURE 7.3
A set of three instances shattered by eight hypotheses. For every possible dichotomy of the instances, there exists a corresponding hypothesis.

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).