3 Using AdaBoost to Minimize Training Error

In this chapter, we study how AdaBoost can be used to minimize the training error, that is, the number of mistakes on the training set itself. As discussed in chapter 1, we will prove that AdaBoost drives the training error down very fast as a function of the error rates of the weak classifiers, even if they all have error rates that are close (but not too close) to the trivial error rate of 50% achievable by simple random guessing. This is AdaBoost's most basic theoretical property.

Note that our approach is deliberately vague with regard to the weak learning algorithm, that is, the source of the weak hypotheses. As discussed in section 2.3.3, our analysis depends only on an assumption of empirical weak learnability. Such an agnostic approach has the important advantage of generality and flexibility: By leaving the weak learner unspecified, we are able to derive a boosting algorithm and an analysis that are immediately applicable to any choice of weak learning algorithm. However, in practice, we must at some point choose or design an appropriate algorithm for this purpose, one that achieves better-than-guessing accuracy on any given distribution over the training set. This chapter therefore includes a discussion of some of the practical approaches that can be used here.

We also look at general conditions that guarantee weak learnability. And as an aside, we dwell briefly on the close relationship between the simple proof technique used to analyze AdaBoost's training error and those commonly used to prove Chernoff bounds, such as theorem 2.1.

One may wonder why it is worthwhile to study the training error at all since our prime interest is in the generalization error. However, as was seen in chapter 2, fitting the training data, typically by minimizing the training error, is one of the main conditions for successful learning. Of course, for now, we are ignoring the other main condition for learning, that of simplicity, an issue that will be addressed in later chapters in our upcoming analysis of the generalization error. Moreover, for that analysis, we will see that the present study of the training error will turn out to be very helpful.

3.1 A Bound on AdaBoost's Training Error

We begin by proving a fundamental bound on AdaBoost's training error. In proving this main theorem, we make no assumptions about the training set and how it was generated, nor about the weak learner. The theorem simply gives a bound on the training error in terms of the error rates of the weak hypotheses.

In the simple version of AdaBoost shown as algorithm 1.1 (p. 5), D_1 is initialized to the uniform distribution over the training set. Here, however, we give a slightly more general proof applicable to an arbitrary initialization of D_1. The resulting proof provides an upper bound on the *weighted* fraction of examples misclassified by H, where each example i is weighted by $D_1(i)$. A bound on the ordinary, unweighted training error, when D_1 is initialized as in algorithm 1.1, follows immediately as a special case.

Theorem 3.1 Given the notation of algorithm 1.1, let $\gamma_t \doteq \frac{1}{2} - \epsilon_t$, and let D_1 be an arbitrary initial distribution over the training set. Then the weighted training error of the combined classifier H with respect to D_1 is bounded as

$$\mathbf{Pr}_{i \sim D_1}[H(x_i) \neq y_i] \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \exp\left(-2 \sum_{t=1}^{T} \gamma_t^2\right).$$

Note that because $\epsilon_t = \frac{1}{2} - \gamma_t$, the *edge* γ_t measures how much better than the random-guessing error rate of $\frac{1}{2}$ is the error rate of the t-th weak classifier h_t. As an illustration of the theorem, suppose all of the γ_t's are at least 10% so that no h_t has error rate above 40%. Then the theorem implies that the training error of the combined classifier is at most

$$\left(\sqrt{1 - 4(0.1)^2}\right)^T \approx (0.98)^T.$$

In other words, the training error drops *exponentially fast* as a function of the number of base classifiers combined. More discussion of this property follows below.

Here is the informal idea behind the theorem: On every round, AdaBoost increases the weights (under distribution D_t) of the misclassified examples. Moreover, because the final classifier H is a (weighted) majority vote of the weak classifiers, if some example is misclassified by H, then it must have been misclassified by most of the weak classifiers as well. This means that it must have had its weight increased on many rounds, so that its weight under the final distribution D_{T+1} must be large. However, because D_{T+1} is a distribution (with weights that sum to 1), there can be only a few examples with large weights, that is, where H makes an incorrect prediction. Therefore, the training error of H must be small.

We now give a formal argument.

Proof Let

$$F(x) \doteq \sum_{t=1}^{T} \alpha_t h_t(x). \tag{3.1}$$

Unraveling the recurrence in algorithm 1.1 that defines D_{t+1} in terms of D_t gives

$$\begin{aligned} D_{T+1}(i) &= D_1(i) \times \frac{e^{-y_i \alpha_1 h_1(x_i)}}{Z_1} \times \cdots \times \frac{e^{-y_i \alpha_T h_T(x_i)}}{Z_T} \\ &= \frac{D_1(i) \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)\right)}{\prod_{t=1}^{T} Z_t} \\ &= \frac{D_1(i) \exp\left(-y_i F(x_i)\right)}{\prod_{t=1}^{T} Z_t}. \end{aligned} \tag{3.2}$$

Since $H(x) = \text{sign}(F(x))$, if $H(x) \neq y$, then $yF(x) \leq 0$, which implies that $e^{-yF(x)} \geq 1$. That is, $\mathbf{1}\{H(x) \neq y\} \leq e^{-yF(x)}$. Therefore, the (weighted) training error is

$$\begin{aligned} \mathbf{Pr}_{i \sim D_1}[H(x_i) \neq y_i] &= \sum_{i=1}^{m} D_1(i)\, \mathbf{1}\{H(x_i) \neq y_i\} \\ &\leq \sum_{i=1}^{m} D_1(i) \exp\left(-y_i F(x_i)\right) &(3.3) \\ &= \sum_{i=1}^{m} D_{T+1}(i) \prod_{t=1}^{T} Z_t &(3.4) \\ &= \prod_{t=1}^{T} Z_t &(3.5) \end{aligned}$$

where equation (3.4) uses equation (3.2), and equation (3.5) uses the fact that D_{T+1} is a distribution (which sums to 1). Finally, by our choice of α_t, we have that

$$\begin{aligned} Z_t &= \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i h_t(x_i)} \\ &= \sum_{i: y_i = h_t(x_i)} D_t(i) e^{-\alpha_t} + \sum_{i: y_i \neq h_t(x_i)} D_t(i) e^{\alpha_t} &(3.6) \\ &= e^{-\alpha_t}\, (1 - \epsilon_t) + e^{\alpha_t}\, \epsilon_t &(3.7) \\ &= e^{-\alpha_t} \left(\frac{1}{2} + \gamma_t\right) + e^{\alpha_t} \left(\frac{1}{2} - \gamma_t\right) &(3.8) \\ &= \sqrt{1 - 4\gamma_t^2}. &(3.9) \end{aligned}$$

Here, equation (3.6) uses the fact that both y_i and $h_t(x_i)$ are $\{-1, +1\}$-valued; equation (3.7) follows from the definition of ϵ_t; and equation (3.9) uses the definition of α_t, which, as we will discuss shortly, was chosen specifically to minimize equation (3.7).

Plugging into equation (3.5) gives the first bound of the theorem. For the second bound, we simply apply the approximation $1 + x \leq e^x$ for all real x. ■

From the proof, it is apparent where AdaBoost's choice of α_t comes from: The proof shows that the training error is upper bounded by $\prod_t Z_t$. To minimize this expression, we can minimize each Z_t separately. Expanding Z_t gives equation (3.7), which can be minimized over choices of α_t using simple calculus giving the choice of α_t used in algorithm 1.1. Note that α_t is being chosen *greedily* on each round t without consideration of how that choice will affect future rounds.

As discussed above, theorem 3.1 assures a rapid drop in training error when each weak classifier is assumed to have error bounded away from $\frac{1}{2}$. This assumption, that $\epsilon_t \leq \frac{1}{2} - \gamma$ for some $\gamma > 0$ on every round t, is a slight relaxation of the empirical γ-weak learning assumption, as discussed in section 2.3.3. When this condition holds, theorem 3.1 implies that the combined classifier will have training error at most

$$\left(\sqrt{1-4\gamma^2}\right)^T \leq e^{-2\gamma^2 T},$$

an exponentially decreasing function of T for any $\gamma > 0$. Although the bound on training error is easier to understand in light of the weak-learnability condition, it is important to remember that AdaBoost and its analysis do *not* require this condition. AdaBoost, being adaptive, does not need to assume an a priori lower bound on the γ_t's, and the analysis takes into account all of the γ_t's. If some γ_t's are large, then the progress (in terms of reducing the bound on the training error) will be that much greater.

Although the bound implies an exponential drop in training error, the bound itself is nevertheless rather loose. For instance, figure 3.1 shows a plot of the training error of the combined classifier compared to the theoretical upper bound as a function of the number of rounds of boosting for the heart-disease dataset described in section 1.2.3. The figure also shows the training errors ϵ_t of the base classifiers h_t with respect to the distributions D_t on which they were trained.

3.2 A Sufficient Condition for Weak Learnability

The assumption of empirical γ-weak learnability is fundamental to the study of boosting, and theorem 3.1 proves that this assumption is sufficient to ensure that AdaBoost will drive down the training error very quickly. But when does this assumption actually hold? Is it possible that this assumption is actually vacuous, in other words, that there are no natural situations in which it holds? What's more, our formulation of weak learnability is somewhat cumbersome, depending as it does on the weighted training error of base hypotheses with respect to virtually any distribution over the training set.

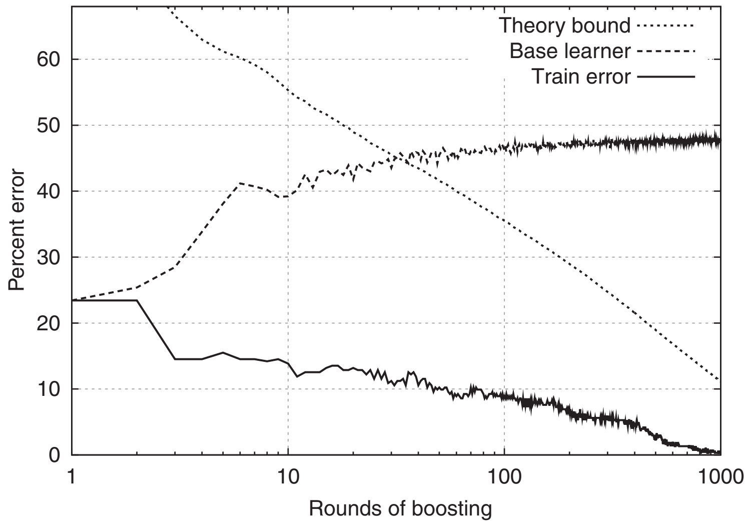

Figure 3.1
The training percent error rate and theoretical upper bound on the training error rate of the combined classifier obtained by using boosting on the entire heart-disease dataset from section 1.2.3. The error rates ϵ_t of the base classifiers on their respective weighted training sets are also plotted.

In this section, we provide a simple condition that itself implies the assumption of empirical weak learnability. As we will see, this condition is only in terms of the functional relationship between the instances and their labels, and does not involve distributions over examples. Although we show only the sufficiency of the condition, later, in section 5.4.3, we will discuss the necessity of the condition as well, thus providing a fairly complete characterization of weak learnability (but one that ignores issues of computational efficiency).

Let all the weak hypotheses belong to some class of hypotheses $\mathcal{H}$. Since we are ignoring computational issues, we simply seek a sufficient condition for there always to exist a weak hypothesis in $\mathcal{H}$ that is significantly better than random for any distribution.

Suppose our training sample S is such that for some weak hypotheses $g_1, \ldots, g_k$ from the given space $\mathcal{H}$, and for some nonnegative coefficients $a_1, \ldots, a_k$ with $\sum_{j=1}^k a_j = 1$, and for some $\theta > 0$, it holds that

$$y_i \sum_{j=1}^{k} a_j g_j(x_i) \geq \theta \tag{3.10}$$

for each example (x_i, y_i) in S. This condition implies that y_i can be computed by a weighted majority vote of the weak hypotheses since equation (3.10) implies that

$$y_i = \operatorname{sign}\left(\sum_{j=1}^{k} a_j g_j(x_i)\right). \tag{3.11}$$

However, the condition in equation (3.10) is a bit stronger; whereas equation (3.11) specifies that barely a weighted majority of the predictions be correct on each example, equation (3.10) demands that *significantly more* than a bare weighted majority be correct. When the condition in equation (3.10) holds for all i, we say that the sample S is *linearly separable with margin θ*. (Margins will be discussed in far more detail in chapter 5.)

In fact, when this condition holds, the assumption of empirical weak learnability holds as well. For suppose that D is any distribution over S. Then, taking expectations of both sides of equation (3.10) and applying linearity of expectations gives

$$\sum_{j=1}^{k} a_j \mathbf{E}_{i\sim D}\left[y_i g_j(x_i)\right] = \mathbf{E}_{i\sim D}\left[y_i \sum_{j=1}^{k} a_j g_j(x_i) \right] \geq \theta.$$

Since the a_j's constitute a distribution, this means that there exists j (and thus a corresponding weak hypothesis $g_j \in \mathcal{H}$) such that

$$\mathbf{E}_{i\sim D}\left[y_i g_j(x_i)\right] \geq \theta.$$

In general, we have that

$$\begin{aligned}\mathbf{E}_{i\sim D}\left[y_i g_j(x_i)\right] &= 1 \cdot \mathbf{Pr}_{i\sim D}\left[y_i = g_j(x_i)\right] + (-1)\cdot \mathbf{Pr}_{i\sim D}\left[y_i \neq g_j(x_i)\right] \\ &= 1 - 2\mathbf{Pr}_{i\sim D}\left[y_i \neq g_j(x_i)\right],\end{aligned}$$

so the weighted error of g_j is

$$\begin{aligned}\mathbf{Pr}_{i\sim D}\left[y_i \neq g_j(x_i)\right] &= \frac{1 - \mathbf{E}_{i\sim D}\left[y_i g_j(x_i)\right]}{2} \\ &\leq \frac{1}{2} - \frac{\theta}{2}.\end{aligned}$$

Thus, this argument shows that if the sample is linearly separable with margin 2γ, then for every distribution over the sample, there exists a base hypothesis in the space $\mathcal{H}$ with weighted error at most $\frac{1}{2} - \gamma$. Such a hypothesis would surely be found by an *exhaustive weak learning algorithm*, meaning a (possibly prohibitively inefficient) base learning algorithm that conducts a brute-force search for the *best* (that is, minimum weighted training error) weak hypothesis in $\mathcal{H}$. This means that when computational costs are not an issue, linear separability with positive margin 2γ is a sufficient condition that guarantees γ-weak learnability.

This assumption of linear separability can be shown to hold in various natural settings. As a simple example, suppose each instance $\mathbf{x}_i$ is a vector in $\mathbb{R}^n$, and that the label y_i is $+1$ for points $\mathbf{x}_i$ falling inside some hyper-rectangle

$$[a_1, b_1] \times \cdots \times [a_n, b_n],$$

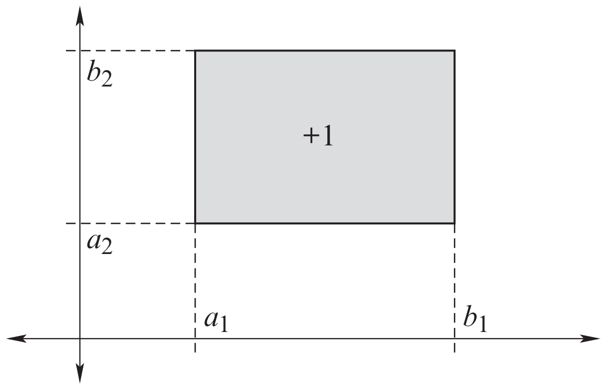

Figure 3.2
A sample target that is $+1$ inside the two-dimensional rectangle $[a_1, b_1] \times [a_2, b_2]$, and -1 outside.

and -1 otherwise. (See figure 3.2 for an example in $n = 2$ dimensions.) Let

$$f(\mathbf{x}) \doteq \frac{1}{4n-1}\left[\sum_{j=1}^{n}\left(\mathbf{1}^*\{x_j \geq a_j\} + \mathbf{1}^*\{x_j \leq b_j\}\right) - (2n-1)\right] \tag{3.12}$$

where $\mathbf{1}^*\{\cdot\}$ is $+1$ if its argument holds true, and -1 otherwise. Then it can be argued that

$$y_i f(\mathbf{x}_i) \geq \frac{1}{4n-1}$$

for all i since the inner sum of equation (3.12) will be equal to $2n$ if $\mathbf{x}_i$ is in the defining hyper-rectangle, and will be at most $2n-2$ otherwise. Noting that f has been written as a convex combination (or average) of decision stumps (over features matching the dimensions of the instances, and including the "constant" stump that always predicts $+1$ or always -1), this shows that our linear separability assumption holds, and thus that the weak learning assumption holds as well when using decision stumps. (See section 3.4.2 for more detail on decision stumps.)

Theorem 3.1 already provides us with the beginnings of a converse to what was shown above. As noted earlier, if the γ-weak learning assumption holds for some $\gamma > 0$, then the number of mistakes of the combined classifier is at most $e^{-2\gamma^2 T}$ (taking D_1 to be uniform). Thus, if

$$T > \frac{\ln m}{2\gamma^2}$$

so that $e^{-2\gamma^2 T} < 1/m$, then the training error of the combined classifier, which is always an integer multiple of $1/m$, must in fact be zero. Moreover, this final classifier has the form of a weighted majority vote. This means that, under the weak learning assumption, theorem 3.1 implies that equation (3.11) must hold for some choice of base classifiers and corresponding coefficients as witnessed by AdaBoost's own combined classifier. This is clearly weaker than equation (3.10), as noted earlier. Nevertheless, it is a start, and in section 5.4.2 we will have the necessary tools to prove the full converse.

3.3 Relation to Chernoff Bounds

As remarked in section 2.2.1, the proof technique used to prove theorem 3.1 is closely related to a standard technique for proving Chernoff bounds, such as Hoeffding's inequality (theorem 2.1). To bring out this connection, we show, as a brief, somewhat whimsical digression, how a special case of Hoeffding's inequality can be derived as an immediate corollary of theorem 3.1. Let $X_1, \ldots, X_n$ be independent, identically distributed random variables such that

$$X_t = \begin{cases} 1 & \text{with probability } \frac{1}{2} + \gamma \\ 0 & \text{with probability } \frac{1}{2} - \gamma \end{cases}$$

for some $\gamma > 0$. What is the probability that at most half of the X_t's are 1? That is, we seek the probability that

$$\frac{1}{n} \sum_{t=1}^{n} X_t \leq \frac{1}{2}. \tag{3.13}$$

According to theorem 2.1, this probability is at most $e^{-2n\gamma^2}$ since, in the notation of that theorem, $\mathbf{E}[A_n] = \frac{1}{2} + \gamma$.

This same result can be derived using our analysis of AdaBoost's training error by contriving an artificially defined training set. In particular, let the instances in the "training set" S be $\{0, 1\}^n$, that is, all n-bit sequences $\mathbf{x}$ corresponding to outcomes of $X_1, \ldots, X_n$. Each example in S is defined to have label $y = +1$. Let the initial distribution D_1 be defined to match the process generating these random variables so that

$$D_1(\mathbf{x}) = \mathbf{Pr}[X_1 = x_1, \ldots, X_n = x_n] = \prod_{t=1}^{n} \left[\left(\tfrac{1}{2} + \gamma\right)^{x_t} \left(\tfrac{1}{2} - \gamma\right)^{1-x_t} \right].$$

(Here, we abuse notation slightly so that the distributions D_t are defined directly over instances in S rather than over the *indices* of those examples.) Now let the number of rounds T be equal to n, and define the t-th "weak hypothesis" h_t to be

$$h_t(\mathbf{x}) = \begin{cases} +1 & \text{if } x_t = 1 \\ -1 & \text{if } x_t = 0. \end{cases}$$

With these definitions, it can be shown (exercise 3.4) that

$$\epsilon_t = \mathbf{Pr}_{\mathbf{x}\sim D_t}[h_t(\mathbf{x}) \neq +1] = \tfrac{1}{2} - \gamma. \tag{3.14}$$

This follows from the independence of the predictions of the h_t's under distribution D_1, as well as from the multiplicative nature of the updates to the distributions created by AdaBoost. This means that all the α_t's are equal to the same positive constant

$$\alpha_t = \alpha = \frac{1}{2}\ln\left(\frac{\frac{1}{2}+\gamma}{\frac{1}{2}-\gamma}\right),$$

so the combined classifier $H(\mathbf{x})$ is a simple (unweighted) majority vote of the h_t's, which is $+1$ if and only if

$$\sum_{t=1}^{n} h_t(\mathbf{x}) > 0,$$

or, equivalently,

$$\frac{1}{n}\sum_{t=1}^{n} x_t > \frac{1}{2}.$$

Thus, applying theorem 3.1, we have that the probability of equation (3.13) is equal to

$$\begin{aligned}\mathbf{Pr}_{\mathbf{x}\sim D_1}\left[\frac{1}{n}\sum_{t=1}^{n} x_t \leq \frac{1}{2}\right] &= \mathbf{Pr}_{\mathbf{x}\sim D_1}[H(\mathbf{x}) \neq +1] \\ &\leq \left(1-4\gamma^2\right)^{n/2} \leq e^{-2n\gamma^2}.\end{aligned}$$

Again, the fact that we get the identical bound as when we apply Hoeffding's inequality directly is not a coincidence, but a consequence of the similar proof techniques used. Moreover, direct generalizations of AdaBoost and theorem 3.1, such as those discussed in section 5.4.2, can be used to prove theorem 2.1 in full generality (see exercise 5.4), as well as some of its extensions, such as Azuma's lemma for non-independent random variables called martingales. That our analysis of AdaBoost applies even when the weak hypotheses are not independent (or martingales) suggests that AdaBoost's mechanism is somehow forcing them to behave as if they actually were independent.

Thus, AdaBoost is a kind of analogue of Hoeffding's inequality for the boosting setting. Hoeffding's inequality is an approximation of the tail of the binomial distribution (equation (2.4)). So what is the analogous boosting algorithm corresponding to the *exact* binomial tail? There is no apparent reason why this strange question should have a meaningful answer. But it does, in the form of the boost-by-majority algorithm presented in chapter 13, which provides an “exact” form of boosting, and one whose corresponding bounds involve exact tails of the binomial distribution, rather than Chernoff-style approximations to it.

3.4 Using and Designing Base Learning Algorithms

AdaBoost, like all boosting algorithms, is an inherently incomplete learning method since it is, by its nature, a “meta-algorithm,” one which is meant to be built on top of, or in combination with, some unspecified base learning algorithm. In this section, we explore some general approaches in the use and choice of the base learning algorithm.

The job of the base learner is to produce base hypotheses h_t. As input, the algorithm accepts a training set

$$S = \langle (x_1, y_1), \ldots, (x_m, y_m) \rangle \tag{3.15}$$

and a set of weights D_t. Its criterion for measuring the goodness of any candidate hypothesis h is the weighted training error

$$\epsilon_t \doteq \mathbf{Pr}_{i \sim D_t}[h(x_i) \neq y_i]. \tag{3.16}$$

In other words, it seeks a base hypothesis h_t that minimizes ϵ_t, or at least one for which ϵ_t is somewhat smaller than $\frac{1}{2}$. Theorem 3.1 shows that this is sufficient to drive down AdaBoost’s training error very quickly. Moreover, in later chapters we will see that the weighted training errors ϵ_t of the base classifiers are also directly related to AdaBoost’s generalization error.

In what follows, we simplify notation by dropping subscripts so that $D = D_t$, $h = h_t$, and so on.

3.4.1 Using the Example Weights

The objective that the base learner seeks to minimize is nearly identical to the ordinary training error, except that training examples now have varying weights. So the first question we need to address is how these weights should be used. Here, there are two main approaches. The first is to use the given distribution D to generate an ordinary, unweighted training sample simply by randomly selecting a sequence of examples S' according to D. In other words,

$$S' = \langle (x_{i_1}, y_{i_1}), \ldots, (x_{i_{m'}}, y_{i_{m'}}) \rangle,$$

where each i_j is selected[1] independently at random according to D. This unweighted sample can then be fed to a base learning algorithm but one that need not be concerned with

1. To select a point i from a distribution D, given access to a standard (pseudo)random number generator, we first precompute in linear time the cumulative distribution $0 = C_0 \leq C_1 \leq \ldots \leq C_m = 1$ where

$$C_i = C_{i-1} + D(i) = \sum_{j=1}^{i} D(j).$$

weighted samples. Thus, this approach, called *boosting by resampling*, is often useful when the chosen base learner cannot easily be modified to handle the given weights directly.

If m', the size of S', is sufficiently large, then the unweighted training error with respect to S' will be a reasonable estimate of the weighted training error on S with respect to D—modulo the issues discussed at length in chapter 2 regarding the complexity of base classifiers and how that complexity relates to the tendency of the error on a sample to diverge from its true error when the sampled error is minimized. Typically, m' is chosen to be equal to m, though sometimes there are reasons to choose larger or smaller values. For instance, using a sample size m' that is significantly smaller than m can sometimes afford a computational speedup. And although boosting by resampling introduces an additional layer of indirection away from the goal of error minimization, this injection of randomness can sometimes be beneficial to the learning process by providing a smoothing effect that can counter the part of the error due to the base learner's variable behavior. A related method called "bagging," which we discuss in section 5.5, works on essentially this same principle.

Of course, in boosting by resampling, the base learner only minimizes an approximation of the weighted training error. An alternative approach is to modify the base learning algorithm to directly utilize the given example weights so that the weighted training error is minimized explicitly. We will see examples shortly. This approach, called *boosting by reweighting*, has the advantage of being direct and exact, and of avoiding all issues of imprecision in estimating the best base hypothesis.

3.4.2 Designing an Algorithm

In the design of the base learner itself, there are again, broadly speaking, two general approaches. The first is to select an existing, off-the-shelf learning algorithm. Boosting is designed for use with any learning algorithm, so there is no reason not to use boosting to improve the performance of an algorithm that may already be pretty good. Thus, for the base learner, we can use standard and well-studied algorithms such as decision trees (see section 1.3) or neural networks. Some of these may expect an unweighted sample, but this is not a problem if boosting by resampling is used. Even if boosting by reweighting is used, many of these algorithms can be modified to handle example weights. For instance, a standard decision tree algorithm may select a node for placement at the root of the tree that maximizes some measure of "purity," such as entropy or the Gini index. Such measures can usually be modified sensibly to take example weights into consideration (this is very natural for something like entropy which is defined for any distribution).

The other broad approach is to design a base learner that finds very simple base hypotheses, ones that, in the spirit of boosting, are expected to be only a bit better than random

Next, we select r uniformly at random from $[0, 1)$ and let i be the unique integer in $\{1, \ldots, m\}$ for which $r \in [C_{i-1}, C_i)$. This i can be found using binary search in $O(\log m)$ time. Moreover, it can be verified that such a random i is distributed exactly according to D.

guessing. A typical choice would be to use *decision stumps* for this purpose. These are single-level decision trees (hence the name), exactly like the ones used in section 1.2.3, where numerous examples are given. Finding the best decision stump for a given weighted training set—the one that minimizes equation (3.16)—can generally be computed very fast. We illustrate this here as a concrete example of a frequently used base learner.

As in the example in section 1.2.3, we assume that our instances are described by a given set of *features* or *attributes* $f_1, \ldots, f_n$. For instance, if each instance x is a person, then a feature $f_k(x)$ might encode the person x's height, or the person's gender, or the person's eye color, and so on. There may be a variety of types of features, such as *binary features* with values in $\{0, 1\}$; *discrete* (or *categorical*) *features* with values taken from an unordered, finite set; and *continuous features*, taking values in $\mathbb{R}$. A particular decision stump is associated with a single feature, but the exact form will depend on the type of the feature.

Given a dataset S as in equation (3.15), and given a distribution D over S, our goal now in designing a decision-stump base learner is to find the best decision stump with respect to S and D. We do this by efficiently searching, eventually considering all possible decision stumps. The "outer loop" of this search considers each of the features f_k in turn, finding the best decision stump associated with that feature, and finally selecting the best stump overall. Since this is straightforward, let us fix a particular feature f_k, and focus on the "inner loop," that is, the problem of finding the best stump associated with this one feature.

If f_k is binary, then the decision stump can vary only in the predictions made for each branch of the split. Thus, it will have the form

$$h(x) = \begin{cases} c_0 & \text{if } f_k(x) = 0 \\ c_1 & \text{if } f_k(x) = 1, \end{cases} \tag{3.17}$$

and we only need to choose the best values of c_0 and c_1 from $\{-1, +1\}$. This could be done by trying all four possibilities, but there is a more generalizable approach. For $j \in \{0, 1\}$ and $b \in \{-1, +1\}$, let

$$W_b^j \doteq \sum_{i: f_k(x_i)=j \wedge y_i=b} D(i) \;=\; \mathbf{Pr}_{i\sim D}[f_k(x_i) = j \wedge y_i = b] \tag{3.18}$$

be the weighted fraction of examples with label b and for which the feature f_k is equal to j. We also use the shorthand W_+^j and W_-^j for W_{+1}^j and W_{-1}^j, respectively. Then the weighted error of h as in equation (3.17) can be computed to be

$$W_{-c_0}^0 + W_{-c_1}^1. \tag{3.19}$$

This is because if $f_k(x) = 0$, then $h(x) = c_0$, so the weight of examples with labels different from $h(x)$ is the first term of equation (3.19), and similarly for the case $f_k(x) = 1$. Equation (3.19) is minimized over c_0 and c_1 if

$$c_j = \begin{cases} +1 & \text{if } W_-^j < W_+^j \\ -1 & \text{if } W_-^j > W_+^j \end{cases} \tag{3.20}$$

(and with c_j chosen arbitrarily if $W_-^j = W_+^j$). Plugging into equation (3.19) gives that the weighted error of h for this optimal setting is

$$\min\{W_-^0, W_+^0\} + \min\{W_-^1, W_+^1\}.$$

Now suppose f_k is discrete with values in some finite set, say $\{1, \ldots, J\}$. We might consider stumps making a J-way split, that is, of the form

$$h(x) = \begin{cases} c_1 & \text{if } f_k(x) = 1 \\ \vdots & \\ c_J & \text{if } f_k(x) = J. \end{cases} \tag{3.21}$$

Directly generalizing the argument above, we let W_b^j be defined as in equation (3.18) for $j = 1, \ldots, J$. Note that all of these can be computed in a single pass through the data in $O(m)$ time. Then the optimal setting of c_j is still exactly as in equation (3.20), giving a stump whose weighted error is

$$\sum_{j=1}^{J} \min\{W_-^j, W_+^j\}. \tag{3.22}$$

Alternatively, we might wish to use simpler stumps making a binary split of the form

$$h(x) = \begin{cases} c_0 & \text{if } f_k(x) = r \\ c_1 & \text{else} \end{cases}$$

for some choice of c_0 and $c_1 \in \{-1, +1\}$, and $r \in \{1, \ldots, J\}$. Again, all $4J$ choices could be tried exhaustively, but a more efficient approach would be first to compute the W_b^j's as above in linear time, as well as

$$W_b \doteq \sum_{j=1}^{J} W_b^j.$$

Then, by arguments similar to those above, the best choices of c_0 and c_1 for a particular choice of r will have a weighted training error of

$$\min\{W_-^r, W_+^r\} + \min\{W_- - W_-^r\ ,\ W_+ - W_+^r\}.$$

Thus, the best choice of r can be found quickly in $O(J)$ time to be the one minimizing this expression, and then the best choice of c_0 and c_1 can be determined using an expression analogous to equation (3.20).

The case where f_k is continuous is the most challenging. Here, we consider decision stumps of the form

$$h(x) = \begin{cases} c_0 & \text{if } f_k(x) \le \nu \\ c_1 & \text{if } f_k(x) > \nu \end{cases} \tag{3.23}$$

for some real-valued threshold ν. For a fixed choice of ν, we are essentially in the binary case from above. We can compute

$$W_b^0 \doteq \mathbf{Pr}_{i\sim D}[f_k(x_i) \le \nu \wedge y_i = b]$$

$$W_b^1 \doteq \mathbf{Pr}_{i\sim D}[f_k(x_i) > \nu \wedge y_i = b]$$

and then set c_0 and c_1 as before. However, it appears that in addition to the four possible settings of c_0 and c_1, we also need to consider an infinite number of settings of ν. Nevertheless, similar to arguments used in section 2.2.3, we can take advantage of the fact that any finite set S of m examples will divide the space of possible thresholds $\nu \in \mathbb{R}$ into just $m+1$ equivalence classes, so that the behavior of a stump of the form above on the sample S will be the same for any two choices of ν selected from the same equivalence class. More concretely, suppose S has been sorted by f_k so that

$$f_k(x_1) \le f_k(x_2) \le \cdots \le f_k(x_m).$$

Then, in searching for the best stump of the form above, it suffices to consider just one threshold value ν from each of the intervals $[f_k(x_i), f_k(x_{i+1}))$ for $i = 1, \ldots, m-1$, as well as $[-\infty, f_k(x_1))$ and $[f_k(x_m), +\infty]$.

So there are essentially $4(m+1)$ choices of c_0, c_1, and ν to consider. Exhaustively computing the weighted training error of each would take $O(m^2)$ time. However, if the examples have been presorted by f_k, then the best decision stump can be found in only $O(m)$ time. This can be done by scanning through the examples, considering each threshold equivalence class in turn, and incrementally updating the W_b^j's as each example is passed. Pseudocode is shown as algorithm 3.1, for the special case in which no two examples have the same value of f_k. To see why this algorithm works, the main point to observe is that, following each iteration $i = 1, \ldots, m-1$, the W_b^j's are set correctly for a decision stump defined by any $\nu \in [f_k(x_i), f_k(x_{i+1}))$, from which the correctness of the other computations follows directly. (If the f_k values are not all distinct, the algorithm can be modified so that, on rounds in which $f_k(x_i) = f_k(x_{i+1})$, only the W_b^j's are updated and all other computations are skipped.)

Note that with sufficient memory, for each feature, the examples need to be presorted only once (*not* on every round), requiring time $O(m \log m)$.

3.4.3 An Application to Face Detection

The methods outlined above are intended to be general-purpose. Sometimes, however, the base learning algorithm can be specially tailored to the application at hand, often to great

Algorithm 3.1
Finding a decision stump for a single continuous feature

```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$
        real-valued feature $f_k$ with $f_k(x_1) < \cdots < f_k(x_m)$
        distribution $D$ over $\{1, \ldots, m\}$.
Goal: find stump for $f_k$ with minimum weighted training error.
Initialize:
  • $W_b^0 \leftarrow 0$, $W_b^1 \leftarrow \sum_{i:y_i=b} D(i)$ for $b \in \{-1, +1\}$.
  • $\epsilon_{best} \leftarrow \min\{W_-^1, W_+^1\}$.
  • Pick $\nu \in [-\infty, f_k(x_1))$, and compute $c_0$ and $c_1$ as in equation (3.20).
For $i = 1, \ldots, m$:
  • $W_{y_i}^0 \leftarrow W_{y_i}^0 + D(i)$.
  • $W_{y_i}^1 \leftarrow W_{y_i}^1 - D(i)$.
  • $\epsilon \leftarrow \min\{W_-^0, W_+^0\} + \min\{W_-^1, W_+^1\}$.
  • If $\epsilon < \epsilon_{best}$:
      ◦ $\epsilon_{best} \leftarrow \epsilon$.
      ◦ Pick $\nu \in \begin{cases} [f_k(x_i), f_k(x_{i+1})) & \text{if } i < m \\ [f_k(x_m), +\infty) & \text{if } i = m. \end{cases}$
      ◦ Recompute $c_0$ and $c_1$ as in equation (3.20).
Output: $h$ as in equation (3.23) for the current final values of $c_0$, $c_1$, and $\nu$.
```

benefit. Indeed, the choice of base learning algorithm affords our greatest opportunity for incorporating prior expert knowledge about a specific problem into the boosting process. A beautiful example of this is given in the application of boosting to face detection. Visually detecting all instances of some object type, such as human faces, in photographs, movies, and other digital images is a fundamental problem in computer vision.

As a first step in applying boosting to this challenge, we need to transform what is really a search task (looking for faces) into a classification problem. To do so, we can regard our instances as small subimages of size, say, 24×24 pixels, each of which would be considered positive if and only if it captures a full frontal shot of a face at a standard scale. An example

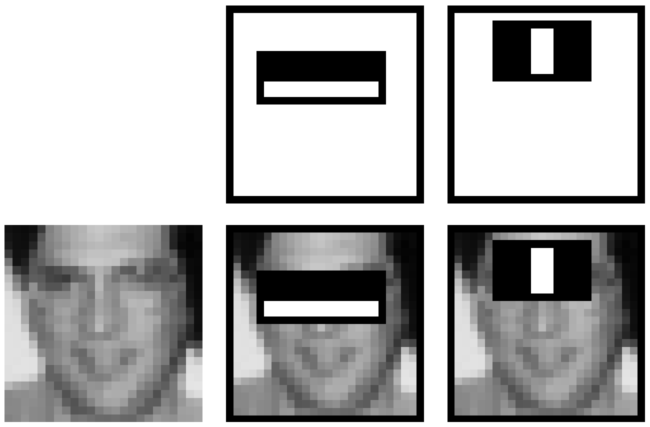

Figure 3.3
The features selected on the first two rounds of boosting, shown in isolation in the top row and as overlays on the sample face at left in the bottom row. (Copyright ©2001 IEEE. Reprinted, with permission, from [227].)

is shown on the left of figure 3.3. Clearly, an accurate classifier for such subimages can be used to detect all faces in an image simply by scanning the entire image and reporting the presence of a face anywhere that it registers a positive instance. Faces of varying sizes can be found by repeating this process at various scales. Needless to say, such an exhaustive process demands that a very fast classifier be used in the innermost loop.

The next major design decision is the choice of weak classifier and weak learning algorithm. The boosting paradigm allows us to choose weak classifiers that are very simple, even if they are individually rather inaccurate; potentially, such simple classifiers can have the additional advantage of being very fast to evaluate. At somewhat of an extreme, we can use weak classifiers which merely detect rectangular patterns of relative light and darkness in the image. Examples are shown in the top row of figure 3.3. The one on the left is sensitive to a dark region over a light region at the specified location of the image; the one on the right is similarly sensitive to dark regions surrounding a light region. In more precise terms, such a pattern defines a real-valued feature that is equal to the sum of the intensities of all the pixels in the black rectangle(s) minus the sum of the intensities of all the pixels in the white rectangle(s). Such a feature can be used to define a decision stump, as described in section 3.4.2, that makes its predictions based on whether the feature value for a particular image is above or below some threshold.

During training, we can consider features defined by all possible patterns of a small number of types, such as the four given in figure 3.4. Each one of these types defines a large number of patterns, each of which is identified with a feature. For instance, the one on the left defines all possible patterns consisting of a white rectangle directly above a black rectangle of equal size. In 24×24 pixel images, the four types of figure 3.4 define some 45,396 features.

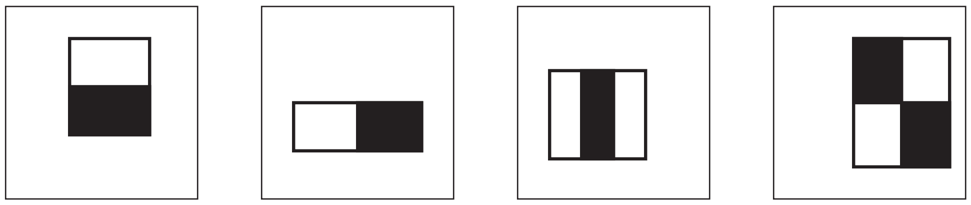

Figure 3.4
The four pattern types used to define features.

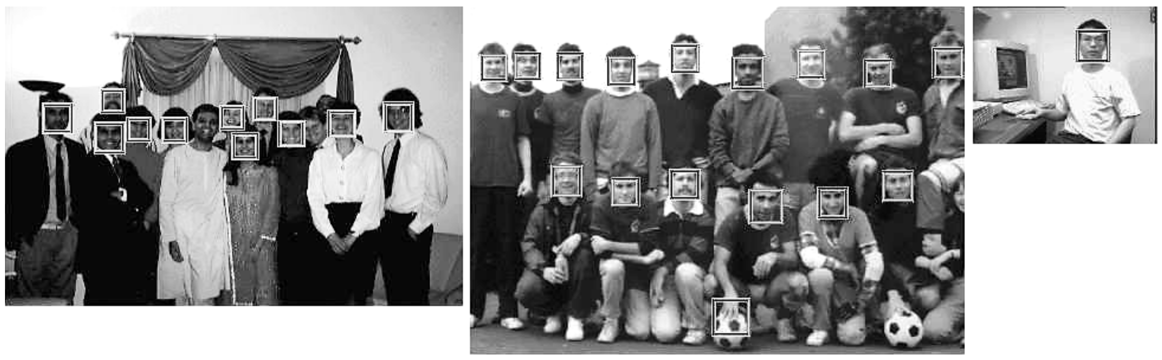

Figure 3.5
The faces detected by the final classifier obtained using boosting on some sample test images. (Copyright ©2001 IEEE. Reprinted, with permission, from [227].)

Having defined this large set of real-valued features, we can apply AdaBoost using the weak learning algorithm given in section 3.4.2 to find the best decision stump. Figure 3.3 in fact shows the two features found on the first two rounds of boosting. The first apparently exploits the tendency of the eyes to appear darker than the upper cheeks, while the second exploits a similar tendency for the eyes to appear darker than the bridge of the nose. Clearly, such weak detectors will individually do a very poor job of identifying faces.

However, when combined with boosting in this fashion, the performance of AdaBoost's final classifier is extremely good. For instance, after 200 rounds of boosting, on one test dataset, the final classifier was able to detect 95% of the faces while reporting false positives at a rate of only 1 in 14,084. Detection results of the complete system are shown on some sample test images in figure 3.5.

In addition to its high accuracy, this approach to face detection can be made extremely fast. Naively, the features we have described require time proportional to the size of the rectangles involved. However, given a bit of precomputation, it becomes possible to evaluate any feature in *constant* time. To see this, we can first define the *integral image* $I(x, y)$ to be the sum of the intensities of all pixels above and to the left of position (x, y). This can be computed for all pixels (x, y) in a single pass over the image, beginning in the

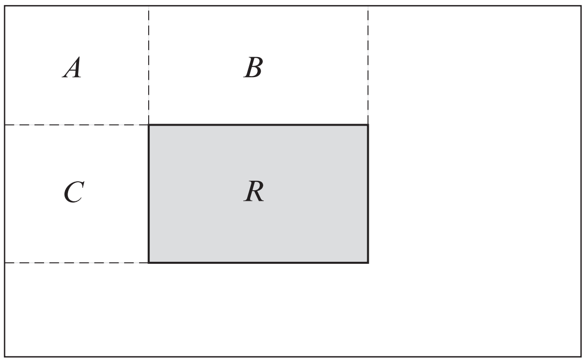

Figure 3.6
The sum of the pixels in any rectangle R can be computed in just four references to the integral image.

upper left corner. Once computed, the sum of all pixels of *any* rectangle can be computed in just four references to the integral image: For suppose we want the sum of pixels in the rectangle R in figure 3.6. This sum can be computed as

$$R = (A + B + C + R) - (A + C) - (A + B) + (A), \tag{3.24}$$

where, with slight abuse of notation, we use A, B, C, and R to stand both for the rectangles in the figure and for the sum of the pixels in each. Note that each of the four parenthesized terms in equation (3.24) can be looked up in the integral image by referencing, respectively, the bottom right, bottom left, top right, and top left corners of rectangle R. Thus, the sum of the pixels in any rectangle, and therefore also any feature, can be evaluated in a small and constant number of references to the integral image. This means that these features can be evaluated very quickly, dramatically speeding up both training and evaluation on test examples.

Evaluation can be made even faster by using a cascading technique in which relatively small and rough classifiers are trained which are good enough to quickly eliminate the vast majority of background images as non-faces. The entire system is so fast that it can be used, for instance, to find all faces in video in real time at 15 frames per second.

Summary

In summary, in this chapter we have proved a bound on the training error obtained by AdaBoost, and we have seen that this error drops exponentially fast as a function of the number of rounds of boosting, given the weak learning assumption. We have also proved a general sufficient condition for the weak learning assumption to hold, and have looked at how to choose or design a weak learning algorithm. Next, we turn to the central issue of generalization beyond the training data.