Chapter 9

Using Confidence-rated Weak Predictions

Having studied AdaBoost and boosting theoretically from a variety of perspectives, we turn next to a number of techniques for extending AdaBoost beyond vanilla binary classification using binary base classifiers. Our emphasis now is on algorithm design beginning, in this chapter, with techniques involving real-valued base hypotheses.

We have so far taken for granted that the base hypotheses used by AdaBoost always produce predictions that are themselves classifications, either -1 or $+1$. The aim of the base learner in such a setting is to find a base classifier with low weighted classification error, that is, a small number of mistakes on the weighted training set. This set-up is simple and natural, and admits the use of off-the-shelf classification learning algorithms as base learners.

However, in some situations, the rigid use of such "hard" predictions can lead to difficulties and significant inefficiencies. For instance, consider the data in Figure 9.1, and a simple base classifier whose predictions only depend on which side of the line L a given point falls on. Suppose for simplicity that these training examples are equally weighted, as will be the case on the first round of boosting. Here, it would certainly seem natural to classify everything above line L as positive, and this fact should be enough for us to construct a prediction rule that is substantially better than random guessing. But how should this classifier predict on the points below L? In the set-up as we have described it up until now, there are only two options: either predict all points below L are positive, or predict they are all negative. Because of the nearly perfect predictions on points above L, either option will yield a classifier that, overall, is significantly better than random, which should be "good enough" for boosting. On the other hand, both options will lead to a very

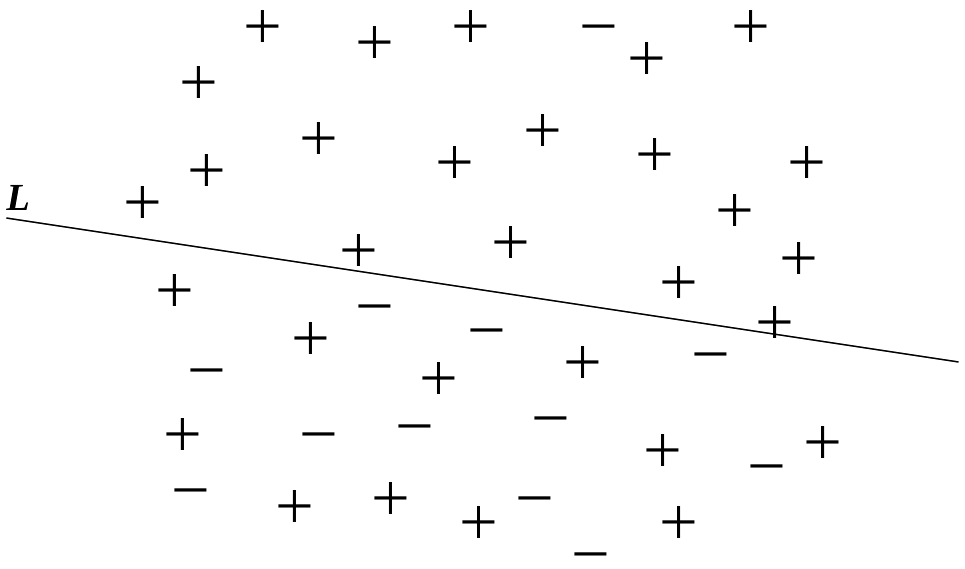

Figure 9.1: A sample dataset.

substantial number of mistakes on the points below L. This is a serious problem because in the process of boosting, all of these bad predictions will eventually need to be corrected or "cleaned up" in later rounds of boosting, a process that can add tremendously to the training time.

The problem here is that a "hard" classifier cannot express varying degrees of confidence. Intuitively, the data suggests that we can be highly confident in predicting that points above L are positive. On the other hand, the even split between positives and negatives on examples below L suggests that the best prediction on these points is in fact no prediction at all, but rather an abstention from prediction expressing a sentiment of absolute uncertainty about the correct label.

Such situations arise naturally with real data as well. For instance, when classifying email as spam or ham, one can easily find somewhat accurate patterns, such as: "if *viagra* occurs in the message, then it is spam." However, it is unclear how such a rule should predict when *viagra* does not occur in the message, and indeed, whatever prediction is made here should probably have low confidence.

In this chapter, we describe an extension of the boosting framework in which each weak hypothesis generates not only predicted classifications, but also *self-rated confidence scores* that estimate the reliability of each of its predictions. Although informally a natural and simple idea, there are two essential questions which arise in its implementation. First, how do we modify AdaBoost, which was designed to handle only simple $\{-1, +1\}$ predictions, to instead use confidence-rated predictions in the most effective manner? And second, how should we design weak learners whose predictions are confidence-rated in the manner described above? In this chapter, we give answers to both of these questions. The result is a powerful set of boosting methods for handling more expressive weak hypotheses, as well as an advanced methodology for designing weak learners appropriate for use with boosting algorithms.

As specific examples, we study techniques for using weak hypotheses that abstain from making predictions on much of the space, as well as for weak hypotheses which partition the instance space into a relatively small number of equivalent prediction regions, such as decision trees and decision stumps. As an application, we demonstrate how this framework can be used as a basis for effectively learning sets of intuitively understandable rules, such as the one given above for detecting spam. We also apply our methodology to derive an algorithm for learning a variant of standard decision trees; the classifiers produced by this algorithm are often both accurate and compact.

In short, this chapter is about techniques for making boosting better. In some cases, the algorithmic improvements we present lead to substantial speed-ups in training time, while in other cases, we see improvement in accuracy or in the comprehensibility of the learned hypothesis.

9.1 The framework

The foundation for a framework with predictions rated for their confidence was laid down already in preceding chapters. In Chapter 5, we used the real-valued margin as a measure of the confidence of the predictions made by the *combined* classifier. In the same fashion, we can bundle the predictions and confidences of a *base* classifier into a single real number. In other words, a base hypothesis can now be formulated as a function of the form $h : \mathcal{X} \to \mathbb{R}$ whose range is all of $\mathbb{R}$ rather than just $\{-1, +1\}$ as was the case up to this point. We interpret the sign of $h(x)$ as the predicted label, -1 or $+1$, to be assigned to instance x, and the magnitude $|h(x)|$ as the *confidence* in this prediction. Thus, if $h(x)$ is close to or far from zero, it is interpreted as a respectively low or high confidence prediction. Although the range of h may generally include all real numbers, we will sometimes restrict this range.

How does this change affect AdaBoost as depicted in Algorithm 1.1? In fact, the required modifications are quite minimal. There is no need at all to modify either the exponential rule for updating D_t:

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

or the computation of the combined classifier:

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right). \tag{9.1}$$

These are already consistent with our interpretation of the confidence-rated predictions: Predictions with high confidence, where $h_t(x)$ is large in absolute value, will cause a dramatic change in the distribution D_t, and will have a major influence on the outcome of the final classifier. Conversely, predictions with low confidence, with $h_t(x)$ near zero, will have a correspondingly low effect.

Indeed, the only necessary modification is in the choice of α_t which earlier depended on the weighted training error

$$\epsilon_t \doteq \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right],$$

a quantity that no longer makes sense when the range of the h_t's extends beyond $\{-1, +1\}$. For the moment, we leave the choice of α_t unspecified, but return to this issue shortly. The resulting generalized version of AdaBoost is given as Algorithm 9.1.

Although α_t is not specified, we can give a bound on the training error of this version of AdaBoost. In fact, the entire first part of our earlier proof of Theorem 3.1 remains valid even when α_t is unspecified, and h_t is real-valued. Only the final part of that proof where Z_t was computed in terms of ϵ_t is no longer valid. Thus, by the first part of that proof, up through Eq. (3.5), we have:

Theorem 9.1 *Given the notation of Algorithm 9.1, the training error of the combined classifier H is at most*

$$\prod_{t=1}^{T} Z_t.$$

Theorem 9.1 suggests that, in order to minimize overall training error, a reasonable approach might be to greedily minimize the bound given in the theorem by minimizing Z_t on each round of boosting. In other words, the theorem suggests that the boosting algorithm and weak learning algorithm should work in concert to choose α_t and h_t on each round so as to minimalize the normalization factor

$$Z_t \doteq \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i)), \tag{9.2}$$

```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:
    • Train weak learner using distribution $D_t$.
    • Get weak hypothesis $h_t : \mathcal{X} \to \mathbb{R}$.
    • Choose $\alpha_t \in \mathbb{R}$.
    • Aim: select $h_t$ and $\alpha_t$ to minimalize the normalization factor
          $Z_t \doteq \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$.
    • Update, for $i = 1, \ldots, m$:
          $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$.
Output the final hypothesis:
    $H(x) = \mathrm{sign}\left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$.
```

Algorithm 9.1: A generalized version of AdaBoost with confidence-rated predictions.

as shown in the pseudocode. From the boosting algorithm's perspective, this provides a general principle for choosing α_t. From the weak learner's point of view, we obtain a general criterion for constructing confidence-rated weak hypotheses that replaces the previous goal of minimizing the weighted training error. We will soon see examples of the consequences of both of these.

Although we focus on minimizing Z_t through the choice of α_t and h_t, it should be noted that this approach is entirely equivalent to the greedy method of minimizing exponential loss (Eq. (7.3)) via coordinate descent as discussed in Sections 7.1 and 7.2. This is because Z_t measures exactly the ratio between the new and old values of the exponential loss so that $\prod_t Z_t$ is its final value. Thus, greedy minimization of the exponential loss on each round is equivalent to minimization of Z_t.

So the approach presented in this chapter is founded on minimization of training error, or alternatively, on minimization of exponential loss on the training set.

We do not take up the important question of the impact of such methods on generalization error, although some of the techniques presented in previous chapters can surely be adapted.

9.2 General methods for algorithm design

Next, we develop some general techniques for working in the framework outlined above, especially for choosing α_t, and for designing weak learning algorithms, with particular consideration of efficiency issues. Specific applications of these general methods are described later in Sections 9.3 and 9.4.

9.2.1 Choosing α_t in general

As just discussed, given h_t, the boosting algorithm should seek to choose α_t to minimize Z_t. We begin by considering this problem in general.

To simplify notation, when clear from context, we take t to be fixed and omit it as a subscript so that $Z = Z_t$, $D = D_t$, $h = h_t$, $\alpha = \alpha_t$, etc. Also, let $u_i = y_i h_t(x_i)$. In the following discussion, we assume without loss of generality that $D(i) \neq 0$ for all i. Our goal is to find α which minimizes Z as a function of α:

$$Z(\alpha) = Z = \sum_{i=1}^{m} D_t(i) e^{-\alpha u_i}.$$

In general, this quantity can be numerically minimized as a function of α. The first derivative of Z is

$$\begin{aligned} Z'(\alpha) = \frac{dZ}{d\alpha} &= -\sum_{i=1}^{m} D_t(i) u_i e^{-\alpha u_i} \\ &= -Z \sum_{i=1}^{m} D_{t+1}(i) u_i \end{aligned}$$

by definition of D_{t+1}. Thus, if D_{t+1} is formed using the value of α_t which minimizes Z (so that $Z'(\alpha) = 0$), then we will have that

$$\sum_{i=1}^{m} D_{t+1}(i) u_i = \mathrm{E}_{i \sim D_{t+1}} \left[y_i h_t(x_i) \right] = 0.$$

In words, this means that, with respect to distribution D_{t+1}, the weak hypothesis h_t will be exactly uncorrelated with the labels y_i.

Moreover,

$$Z''(\alpha) = \frac{d^2 Z}{d\alpha^2} = \sum_{i=1}^{m} D_t(i) u_i^2 e^{-\alpha u_i}$$

is strictly positive for all $\alpha \in \mathbb{R}$ (ignoring the trivial case that $u_i = 0$ for all i), meaning that $Z(\alpha)$ is strictly convex in α. Therefore, $Z'(\alpha)$ can have at most one zero. In addition, if there exists i such that $u_i < 0$ then $Z'(\alpha) \to \infty$ as $\alpha \to \infty$. Similarly, $Z'(\alpha) \to -\infty$ as $\alpha \to -\infty$ if $u_i > 0$ for some i. This means that $Z'(\alpha)$ has at least one root, except in the degenerate case that all nonzero u_i's are of the same sign. Furthermore, because $Z'(\alpha)$ is strictly increasing, we can numerically find the unique minimum of $Z(\alpha)$ by a simple binary search, or more sophisticated numerical methods.

Summarizing, we have argued the following:

Theorem 9.2 *Assume the set $\{y_i h_t(x_i) : i = 1, \ldots, m\}$ includes both positive and negative values. Then there exists a unique choice of α_t which minimizes Z_t. Furthermore, for this choice of α_t, we have that*

$$\mathrm{E}_{i \sim D_{t+1}} \left[y_i h_t(x_i) \right] = 0 \,. \tag{9.3}$$

Note that, in the language of Chapter 8, the condition in Eq. (9.3), which is essentially the same as Eq. (8.9), is equivalent to D_{t+1} belonging to the hyperplane associated with the selected weak hypothesis h_t. It can also be shown that D_{t+1} is in fact the projection onto that hyperplane, just as in Section 8.1.3.

9.2.2 Binary predictions

In the very special case of binary predictions in which all predictions $h(x_i)$ are in $\{-1, +1\}$, we let ϵ be the usual weighted error:

$$\epsilon \doteq \sum_{i : y_i \neq h(x_i)} D(i).$$

Then we can rewrite Z as

$$Z = \epsilon e^{\alpha} + (1 - \epsilon) e^{-\alpha}$$

which is minimized when

$$\alpha = \tfrac{1}{2} \ln \left(\frac{1 - \epsilon}{\epsilon} \right)$$

giving

$$Z = 2\sqrt{\epsilon(1 - \epsilon)}. \tag{9.4}$$

Thus, we immediately recover the original version of AdaBoost for $\{-1,+1\}$-valued base classifiers, as well as the analysis of its training error given in Section 3.1 via Theorem 9.1. Moreover, Z, as in Eq. (9.4), is minimized when ϵ is as far from $1/2$ as possible. If $-h$ can be chosen whenever h can be, then we can assume without loss of generality that $\epsilon < 1/2$ so that minimizing Z in Eq. (9.4) is equivalent to minimizing the weighted training error ϵ. Thus, in this case, we also recover the usual criterion for selecting binary weak hypotheses.

9.2.3 Predictions with bounded range

When the predictions of the weak hypotheses lie in some bounded range, say $[-1,+1]$, we cannot in general give analytic expressions for the minimizing choice of α and resulting value of Z. Nevertheless, we can give useful analytic approximations of these. Since the predictions $h(x_i)$ are in $[-1,+1]$, the u_i's are as well, where, as before, $u_i = y_i h(x_i)$. Thus, we can use the convexity of $e^{-\alpha u}$ (as a function of u) to upper-bound Z as follows:

$$\begin{aligned} Z &= \sum_{i=1}^{m} D(i) e^{-\alpha u_i} && (9.5) \\ &\leq \sum_{i=1}^{m} D(i) \left[\left(\frac{1+u_i}{2} \right) e^{-\alpha} + \left(\frac{1-u_i}{2} \right) e^{\alpha} \right] \\ &= \frac{e^{\alpha}+e^{-\alpha}}{2} - \frac{e^{\alpha}-e^{-\alpha}}{2} r. && (9.6) \end{aligned}$$

Here,

$$\begin{aligned} r = r_t &\doteq \sum_{i=1}^{m} D_t(i) y_i h_t(x_i) \\ &= \mathrm{E}_{i \sim D_t} \left[y_i h_t(x_i) \right] \end{aligned}$$

is a measure of the correlation between the y_i's and the predictions $h_t(x_i)$ with respect to the distribution D_t. The upper bound given in Eq. (9.6) is minimized when we set

$$\alpha = \frac{1}{2} \ln \left(\frac{1+r}{1-r} \right), \qquad (9.7)$$

which, plugging into Eq. (9.6), gives

$$Z \leq \sqrt{1-r^2}. \qquad (9.8)$$

Thus, in this case, α_t can be chosen analytically as in Eq. (9.7), and, to minimize Eq. (9.8), weak hypotheses can be chosen so as to maximize r_t (or $|r_t|$). Theorem 9.1 and Eq. (9.8) immediately give a bound of

$$\prod_{t=1}^{T} \sqrt{1 - r_t^2}$$

on the training error of the combined classifier. Of course, this approach is approximate, and better results might be possible with more exact calculations.

9.2.4 Weak hypotheses that abstain

We next consider a natural special case in which the range of each weak hypothesis h_t is restricted to $\{-1, 0, +1\}$. In other words, a weak hypothesis can make a definitive prediction that the label is -1 or $+1$, or it can "abstain" by predicting 0, effectively saying "I don't know." No other levels of confidence are allowed.

For fixed t, let U_0, U_{-1} and U_{+1} be defined by

$$U_b \doteq \sum_{i:u_i=b} D(i) = \Pr_{i\sim D}\left[u_i = b\right]$$

for $b \in \{-1, 0, +1\}$. Also, for readability of notation, we often abbreviate subscripts $+1$ and -1 by the symbols $+$ and $-$ so that U_{+1} is written U_+, and U_{-1} is written U_-. We can calculate Z as:

$$\begin{aligned} Z &= \sum_{i=1}^{m} D(i) e^{-\alpha u_i} \\ &= \sum_{b\in\{-1,0,+1\}} \sum_{i:u_i=b} D(i) e^{-\alpha b} \\ &= U_0 + U_- e^{\alpha} + U_+ e^{-\alpha}. \end{aligned}$$

Then Z is minimized when

$$\alpha = \tfrac{1}{2} \ln \left(\frac{U_+}{U_-} \right). \tag{9.9}$$

For this setting of α, we have

$$\begin{aligned} Z &= U_0 + 2\sqrt{U_- U_+} \\ &= 1 - \left(\sqrt{U_+} - \sqrt{U_-}\right)^2 \end{aligned} \tag{9.10}$$

where we have used the fact that $U_0+U_++U_- = 1$. If $U_0 = 0$ (so that h effectively has range $\{-1, +1\}$), then the choices of α and resulting values of Z are identical to what was derived in Section 9.2.2.

Using abstaining weak hypotheses can sometimes admit a significantly faster implementation, both of the weak learner and the boosting algorithm. This is especially true when using weak hypotheses that are *sparse* in the sense that they are nonzero on only a relatively small fraction of the training examples. This is because the main operations described above can often be implemented in a way that only involves examples for which a given hypothesis is nonzero. For instance, computing U_+ and U_- clearly only involves such examples, and so as well is this the case for Z as in Eq. (9.10) and α as in Eq. (9.9).

Moreover, updating the distribution D_t can also be speeded up by working instead with a set of *unnormalized* weights d_t that are proportional to D_t. In particular, we initialize $d_1(i) = 1$ for $i = 1, \ldots, m$, and then use an unnormalized version of AdaBoost's update rule, namely,

$$d_{t+1}(i) = d_t(i)e^{-\alpha_t y_i h_t(x_i)}. \tag{9.11}$$

It can immediately be seen that $d_t(i)$ will always be off by a fixed multiplicative constant from $D_t(i)$. This constant does not affect the computation of α-values as in Eq. (9.9) since the constant simply cancels with itself. It also does not affect the choice of the weak hypothesis with the smallest Z-value, or equivalently, from Eq. (9.10), the largest value of

$$\left|\sqrt{U_+} - \sqrt{U_-}\right|$$

since a computation of this quantity using the unnormalized weights d_t will be off by the same multiplicative constant for every weak hypothesis. Thus, each weak hypothesis can still be evaluated against our criterion for choosing the best in time proportional to the number of examples for which it is nonzero. The key advantage of this technique is that according to the update rule in Eq. (9.11), only the weights of examples for which the selected weak hypothesis h_t is nonzero need be updated since the weights of all other examples, where h_t is equal to zero, are unchanged.[1]

These ideas are brought together explicitly in Algorithm 9.2. For simplicity, we have here assumed a given space $\mathcal{H}$ of n weak hypotheses $\hbar_1, \ldots, \hbar_n$ that is large, but still small enough to search over. As a preprocessing step, this implementation begins by computing for each weak hypothesis $\hbar_j$ lists A_+^j and A_-^j of all examples (x_i, y_i) on which $y_i\hbar_j(x_i)$ is $+1$ or -1, respectively. The algorithm also maintains unnormalized weights $d(i)$ as above where we have dropped the t subscript

[1]On an actual computer, the unnormalized weights d_t may become so small or so large as to cause numerical difficulties. This can be avoided by occasionally renormalizing all of the weights.

```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$
       weak hypotheses $\hbar_1, \ldots, \hbar_n$ with range $\{-1, 0, +1\}$.
Initialize:
  • $A_b^j = \{1 \le i \le m : y_i \hbar_j(x_i) = b\}$ for $j = 1, \ldots, n$ and for $b \in \{-1, +1\}$
  • $d(i) \leftarrow 1$ for $i = 1, \ldots, m$
For $t = 1, \ldots, T$:
  • For $j = 1, \ldots, n$:
      – $U_b^j \leftarrow \sum_{i \in A_b^j} d(i)$ for $b \in \{-1, +1\}$.
      – $G_j \leftarrow \left| \sqrt{U_+^j} - \sqrt{U_-^j} \right|$.
  • $j_t = \arg\max_{1 \le j \le n} G_j$.
  • $\alpha_t = \frac{1}{2} \ln \left( \frac{U_+^{j_t}}{U_-^{j_t}} \right)$.
  • for $u \in \{-1, +1\}$, for $i \in A_u^{j_t}$: $d(i) \leftarrow d(i) e^{-\alpha_t u}$.
Output the final hypothesis:
  $$H(x) = \text{sign}\left( \sum_{t=1}^{T} \alpha_t \hbar_{j_t}(x) \right).$$
```

Algorithm 9.2: An efficient version of confidence-rated AdaBoost with abstaining weak hypotheses.

emphasizing the fact that only some values change on each iteration. On every round t, for every $\hbar_j$, the values U_-^j and U_+^j are computed corresponding to U_- and U_+ above, though off by a multiplicative constant since unnormalized weights $d(i)$ have been used. Next, G_j, our measure of goodness for each $\hbar_j$ is computed and the best j_t selected along with α_t. Finally, the weights $d(i)$ for which $\hbar_{j_t}(x_i)$ is nonzero are updated. Note that all operations involve only the examples on which the weak hypotheses are nonzero, a substantial savings when these are sparse.

This idea can be carried even further. Rather than recomputing on every round the entire sum defining U_b^j, only some of whose terms may have changed, we can instead update the variables U_b^j just when individual terms change. In other words, whenever some particular $d(i)$ is updated, we can also update all those variables

```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$
       weak hypotheses $\hbar_1, \ldots, \hbar_n$ with range $\{-1, 0, +1\}$.
For $j = 1, \ldots, n$:
  • $A_b^j = \{1 \le i \le m : y_i \hbar_j(x_i) = b\}$ for $b \in \{-1, +1\}$.
  • $U_b^j \leftarrow \sum_{i \in A_b^j} d(i)$ for $b \in \{-1, +1\}$.
  • $G_j \leftarrow \left| \sqrt{U_+^j} - \sqrt{U_-^j} \right|$.
For $i = 1, \ldots, m$:
  • $d(i) \leftarrow 1$.
  • $B_b^i = \{1 \le j \le n : y_i \hbar_j(x_i) = b\}$ for $b \in \{-1, +1\}$.
For $t = 1, \ldots, T$:
  • $j_t = \arg \max_{1 \le j \le n} G_j$.
  • $\alpha_t = \frac{1}{2} \ln \left( \frac{U_+^{j_t}}{U_-^{j_t}} \right)$.
  • For $u \in \{-1, +1\}$, for $i \in A_u^{j_t}$:
      – $\Delta \leftarrow d(i) \, (e^{-\alpha_t u} - 1)$.
      – $d(i) \leftarrow d(i) e^{-\alpha_t u}$.
      – $U_b^j \leftarrow U_b^j + \Delta$ for $b \in \{-1, +1\}$ and for $j \in B_b^i$.
  • Recompute $G_j$ for all $j$ for which $U_+^j$ or $U_-^j$ have changed.
Output the final hypothesis:
      $H(x) = \text{sign}\left( \sum_{t=1}^{T} \alpha_t \hbar_{j_t}(x) \right)$.
```

Algorithm 9.3: An even more efficient version of Algorithm 9.2.

U_b^j whose defining sum includes $d(i)$ as a term; this can be done simply by adding the new value of $d(i)$ and subtracting its old value. To do this efficiently only requires precomputing for each training example (x_i, y_i), additional "reverse-index" lists B_+^i and B_-^i of all weak hypotheses $\hbar_j$ for which $y_i\hbar_j(x_i)$ is $+1$ or -1, respectively, thus making it easy to find all sums affected by an update to $d(i)$. The revised algorithm is shown as Algorithm 9.3 with all the required bookkeeping made explicit. Note that the G variables can also be updated only when necessary, and the best chosen efficiently, for instance, using a priority queue. A simple induction argument shows that the values of U_b^j and G_j are the same at the beginning of each round as in Algorithm 9.2. This version of the algorithm can be especially fast when the weak hypotheses are sparse in the additional "reverse" sense that only a few are nonzero on a given example x_i so that the sets B_-^i and B_+^i are relatively small.

In Section 11.5.1, we give an example of an application in which this technique results in an improvement in computational efficiency by more than three orders of magnitude.

9.2.5 A simplified criterion

As discussed in Section 9.1, in our framework, the weak learner should attempt to find a weak hypothesis h that minimizes Eq. (9.2). Before continuing, we make the small observation that when using confidence-rated weak hypotheses, this expression can be simplified by folding α_t into h_t, in other words, by assuming without loss of generality that the weak learner can freely scale any weak hypothesis h by any constant factor $\alpha \in \mathbb{R}$. Then (dropping t subscripts) the weak learner's goal becomes that of minimizing

$$Z = \sum_{i=1}^{m} D(i) \exp(-y_i h(x_i)). \tag{9.12}$$

The technique presented in the next section makes use of this simplified criterion. In addition, for some algorithms, it may be possible to make appropriate modifications to handle such a loss function directly. For instance, gradient-based algorithms, such as those used for training neural networks, can easily be modified to minimize Eq. (9.12) rather than the more traditional mean squared error.

9.2.6 Domain-partitioning weak hypotheses

We focus next on weak hypotheses that make their predictions based on a *partitioning* of the domain $\mathcal{X}$. To be more specific, each such weak hypothesis h is associated with a partition of $\mathcal{X}$ into disjoint blocks $X_1, \ldots, X_N$ which cover all of $\mathcal{X}$ and for which $h(x) = h(x')$ for all $x, x' \in X_j$. In other words, h's prediction depends only on which block X_j a given instance falls into. A prime example of such a hypothesis is a decision tree (or stump) whose leaves define a partition of the domain.

Suppose that we have already found a partition $X_1, \ldots, X_N$ of the space. What predictions should be made for each block of the partition? In other words, how do we find a function $h : \mathcal{X} \to \mathbb{R}$ which respects the given partition and which minimizes Eq. (9.12) for the given distribution $D = D_t$?

For all x within each block X_j, $h(x)$ will be equal to some fixed value c_j, so our goal is simply to find appropriate choices for c_j. For each j and for $b \in \{-1, +1\}$, let

$$W_b^j \doteq \sum_{i : x_i \in X_j \wedge y_i = b} D(i) = \Pr_{i \sim D}\left[x_i \in X_j \wedge y_i = b\right]$$

be the weighted fraction of examples which fall in block j and which are labeled b. Then Eq. (9.12) can be rewritten as

$$\begin{aligned} Z &= \sum_{j=1}^{N} \sum_{i : x_i \in X_j} D(i) \exp(-y_i c_j) \\ &= \sum_{j=1}^{N} \left(W_+^j e^{-c_j} + W_-^j e^{c_j} \right). \end{aligned} \tag{9.13}$$

Using standard calculus, we see that this is minimized when

$$c_j = \tfrac{1}{2} \ln \left(\frac{W_+^j}{W_-^j} \right). \tag{9.14}$$

Plugging into Eq. (9.13), this choice gives

$$Z = 2 \sum_{j=1}^{N} \sqrt{W_+^j W_-^j}. \tag{9.15}$$

Eq. (9.14) provides the best choice of c_j according to our criterion. Note that the sign of c_j is equal to the (weighted) majority class within block j. Moreover, c_j will be close to zero (a low confidence prediction) if there is a roughly equal split of positive and negative examples in block j; likewise, c_j will be far from zero if one label strongly predominates.

Further, Eq. (9.15) provides a criterion for selecting among domain-partitioning base classifiers: the base learning algorithm should seek a base classifier from a given family that minimizes this quantity. Once found, the real-valued predictions for each block are given by Eq. (9.14).

For example, if using decision stumps as base classifiers, we can search through the space of all possible splits of the data based on the given features or attributes in a manner nearly identical to that given in Section 3.4.2. In fact, the only necessary change is in the criterion for choosing the best split, and in the values at the leaves of the stump. For instance, when considering an ℓ-way split as in Eq. (3.21), rather than using the weighted error computed in Eq. (3.22) as the selection criterion, we would instead use the corresponding Z-value, which in this case, by the arguments above, would be

$$2\sum_{j=1}^{\ell}\sqrt{W_+^j W_-^j}.$$

Likewise, c_j, rather than being set as in Eq. (3.20), would instead be set as in Eq. (9.14).

In general, each split candidate creates a partition of the domain from which W_+^j and W_-^j, and thus also Z as in Eq. (9.15), can be computed. Once the split with the smallest Z has been determined, the actual real-valued prediction for each branch of the split can be computed using Eq. (9.14).

The criterion given by Eq. (9.15) can also be used as a splitting criterion in growing a decision tree for use as a weak hypothesis, rather than the more traditional Gini index or entropic function. In other words, the decision tree could be built by greedily choosing at each decision node the split which causes the greatest drop in the value of the function given in Eq. (9.15). In this fashion, during boosting, each tree can be built using the splitting criterion given by Eq. (9.15) while the predictions at the leaves of the boosted trees are given by Eq. (9.14). An alternative approach for combining boosting with decision trees is given in Section 9.4.

The scheme presented above requires that we predict as in Eq. (9.14) on block j. It may well happen that W_-^j or W_+^j is very small or even zero, in which case c_j will be very large or infinite in magnitude. In practice, such large predictions may cause numerical problems. In addition, there may be theoretical reasons to suspect that large, overly confident predictions will increase the tendency to overfit.

To limit the magnitudes of the predictions, we can instead use the "smoothed" values

$$c_j = \tfrac{1}{2}\ln\left(\frac{W_+^j + \varepsilon}{W_-^j + \varepsilon}\right) \tag{9.16}$$

in lieu of Eq. (9.14) for some appropriately small positive value of ε. Because W_-^j

and W_+^j are both bounded between 0 and 1, this has the effect of bounding $|c_j|$ by

$$\tfrac{1}{2}\ln\left(\frac{1+\varepsilon}{\varepsilon}\right) \approx \tfrac{1}{2}\ln(1/\varepsilon).$$

Moreover, this smoothing only slightly weakens the value of Z since, plugging into Eq. (9.13) gives

$$\begin{aligned} Z &= \sum_{j=1}^{N}\left(W_+^j\sqrt{\frac{W_-^j+\varepsilon}{W_+^j+\varepsilon}} + W_-^j\sqrt{\frac{W_+^j+\varepsilon}{W_-^j+\varepsilon}}\right) \\ &\le \sum_{j=1}^{N}\left(\sqrt{(W_-^j+\varepsilon)W_+^j} + \sqrt{(W_+^j+\varepsilon)W_-^j}\right) \\ &\le \sum_{j=1}^{N}\left(2\sqrt{W_-^jW_+^j} + \sqrt{\varepsilon W_+^j} + \sqrt{\varepsilon W_-^j}\right) \qquad (9.17) \\ &\le 2\sum_{j=1}^{N}\sqrt{W_-^jW_+^j} + \sqrt{2N\varepsilon}\,. \qquad (9.18) \end{aligned}$$

In Eq. (9.17), we used the inequality $\sqrt{x+y} \le \sqrt{x} + \sqrt{y}$ for nonnegative x and y. In Eq. (9.18), we used the fact that

$$\sum_{j=1}^{N}(W_-^j + W_+^j) = 1,$$

which implies

$$\sum_{j=1}^{N}\left(\sqrt{W_-^j} + \sqrt{W_+^j}\right) \le \sqrt{2N}\,.$$

(Recall that N is the number of blocks in the partition.) Thus, comparing Eqs. (9.18) and (9.15), we see that Z will not be greatly degraded by smoothing if we choose $\varepsilon \ll 1/(2N)$. In practice, ε is typically chosen to be on the order of $1/m$ where m is the number of training examples.

Practically, the use of confidence-rated predictions can lead to very dramatic improvements in performance. For instance, Figure 9.2 shows the results of one experiment demonstrating this effect. Here, the problem is to classify titles of newspaper articles by their broad topic as in Section 7.7.1. The base classifiers are decision stumps which test for the presence or absence of a word or short phrase, and predict accordingly. When AdaBoost is run with $\{-1,+1\}$-valued base classifiers (that is, without confidence-rated predictions), the slow convergence described

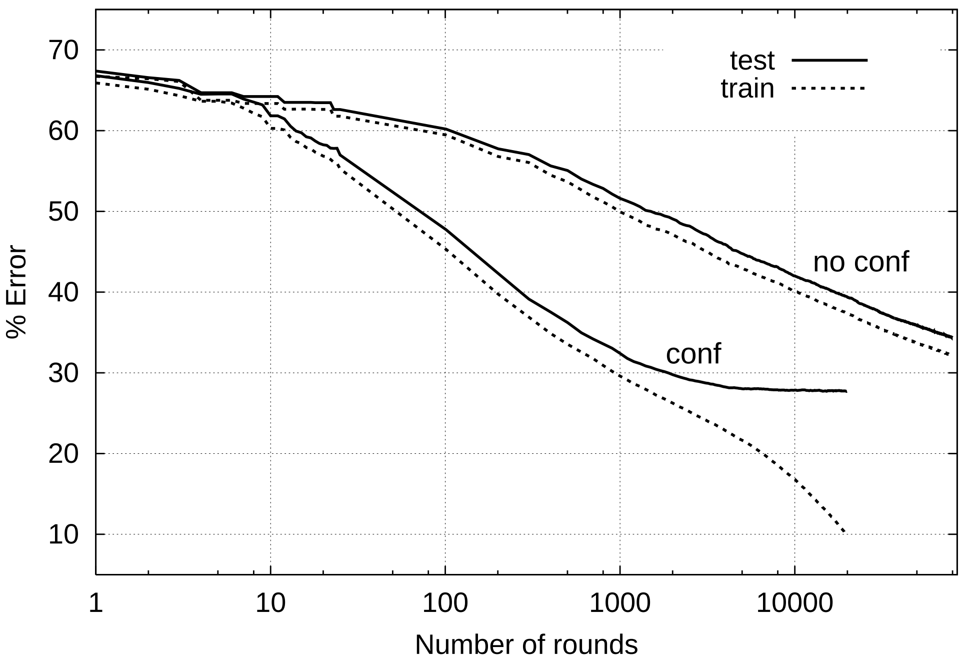

Figure 9.2: Train and test error curves for boosting on decision stumps on a text-categorization task with (bottom curves) or without (top curves) confidence-rated base hypotheses.

at the beginning of this chapter is observed, and for the very reason that was earlier given.

When confidence-rated predictions are employed (using the method above for constructing domain-partitioning base classifiers), the improvement in efficiency is spectacular. Table 9.1 shows the number of iterations needed to achieve various test error rates. For instance, in this case, the number of iterations to achieve a test error of 35% has been slashed by a factor of more than a hundred.

9.3 Learning rule-sets

We look next at two learning algorithms based on the general framework and methods developed above. The first of these is for learning sets of *rules*, simple "if-then" statements for formulating a prediction. For instance, in classifying email as spam or ham, one could easily imagine many plausible rules:

If *viagra* occurs in the message, then predict *spam*.

	round first reached		
% error	conf.	no conf.	speedup
40	268	16,938	63.2
35	598	65,292	109.2
30	1,888	>80,000	–

Table 9.1: A comparison of the number of rounds needed to achieve various test accuracies both with and without confidence-rated predictions for the same learning task as in Figure 9.2. The speedup column shows how many times faster boosting is with confidence-rated predictions, rather than without.

If message is from my wife,	then predict *ham*.
If message contains corrupted html links,	then predict *spam*.
⋮	

Such rules are generally considered to be intuitive and easy for people to understand. Indeed, some early spam-filtering systems asked users to formulate their *own* rules for identifying spam, and such rule-sets have also been utilized in the past, for instance, in so-called expert systems for medical diagnosis. A variety of learning algorithms, such as RIPPER, have been devised for inferring a good rule-set from data.

Rules are in fact a special form of abstaining hypotheses. For instance, in our formalism, the first rule above could be reformulated as the following:

$$h(x) = \begin{cases} +1 & \text{if } \textit{viagra} \text{ occurs in message } x \\ 0 & \text{else.} \end{cases}$$

In general, rules output ± 1 when some condition holds (in this case, *viagra* occurring in the message), and 0 otherwise. Examples which satisfy the condition are said to be *covered* by the rule.

Viewed in isolation, the problem of finding a good rule-set presents many challenges: How do we balance the natural tension that exists between the competing goals of selecting rules that cover as many examples as possible, versus choosing rules that are as accurate as possible on the examples that they do cover? (These goals are typically in competition because usually it is easier for more specialized rules to be more accurate.) What do we do if two rules in the set contradict each other in their predictions (one predicting positive and the other negative on an example covered by both)? How much overlap should there be between rules in terms of the examples that they cover? And how do we construct a concise set of rules that will be as accurate as possible in its overall predictions?

In fact, we can provide a boosting-based answer to all these challenges by directly applying confidence-rated boosting techniques using rules as (abstaining) weak hypotheses. Doing so will result in a combined hypothesis with the following form as a set of weighted rules:

$$\begin{array}{l} \text{If } C_1 \text{ then predict } s_1 \text{ with confidence } \alpha_1. \\ \quad\vdots \\ \text{If } C_T \text{ then predict } s_T \text{ with confidence } \alpha_T. \end{array} \tag{9.19}$$

Here, each rule (weak hypothesis) has an associated condition C_t, prediction $s_t \in \{-1, +1\}$, and confidence α_t. To evaluate such a rule-set (combined hypothesis) on a new example x, we simply add up, for each rule covering x, the predictions s_t weighted by α_t, and then take the sign of the computed sum. That is,

$$H(x) = \text{sign}\left(\sum_{t:C_t \text{ holds on } x} \alpha_t s_t\right). \tag{9.20}$$

This description is nothing more than an equivalent reformulation of Eq. (9.1) for the present setting. Note that contradictory rules are handled simply by assigning each a weight or confidence, and evaluating the prediction of the entire rule-set by taking a weighted sum of the predictions of all covering rules.

The rule-set itself can be constructed using the usual boosting mechanism of repeatedly assigning weights to examples and then searching through some space of conditions for the rule (weak hypothesis) that optimizes some criterion. This mechanism automatically focuses the construction of each subsequent rule on parts of the domain where accuracy or coverage is poor. We can directly apply the results of Section 9.2.4 to set the value of each α_t, and to provide a criterion for choosing the best rule on each round. Note that this criterion, as given in Eq. (9.10), provides a concrete and principled means of balancing the trade-off between the competing goals of finding a rule both with high coverage and high accuracy.

In addition to their intuitive interpretability, we note that rules, like abstaining weak hypotheses in general, can sometimes admit significant efficiency improvements since, using the techniques described in Section 9.2.4, operations which might naively require time proportional to the total number of training examples can usually be done instead in time proportional just to the number of examples actually covered by the selected rule, which might be much smaller.

So far, we have left unspecified the form of the condition used in defining each rule. Here, as is so often the case, there exist myriad possibilities, of which we discuss just a couple.

For concreteness, let us assume that instances are described by features or attributes as in Section 3.4.2. Along the lines of the decision stumps explored in that section, we might consider using the same sorts of simple conditions that can naturally be defined using such features. For instance, this leads to rules such as these:

If (eye-color = blue) then predict $+1$.
If (sex = female) then predict -1.
If (height ≥ 60) then predict -1.
If (age ≤ 30) then predict $+1$.
$\vdots$

Finding rules with conditions of these forms and optimizing the criterion in Eq. (9.10) can be done efficiently using a search technique very similar to what was described in Section 3.4.2 for finding decision stumps, but using this modified criterion. Such rules are really just a one-sided, confidence-rated version of decision stumps, and thus are rather weak.

In some settings, it may be advantageous to use rules whose conditions are more expressive, leading to rules that are more specialized, but potentially more accurate on the examples that they do cover. Conditions which are *conjunctions* of other base conditions are often considered natural for this purpose. For instance, this leads to rules like this one:

If (sex = male) $\wedge$ (age ≥ 40) $\wedge$ (blood-pressure ≥ 135)
then predict $+1$.

In general, these conditions have the form $B_1 \wedge \cdots \wedge B_\ell$ where each B_j is a base condition chosen from some set that is presumably easy to search, for instance, conditions of the forms given above.

Finding the optimal conjunctive condition will often be computationally infeasible. Nevertheless, there are natural greedy search techniques that can be used. Specifically, starting with an empty conjunction, we can iteratively add one conjunct at a time, on each iteration, choosing the conjunct that causes the greatest improvement in our search criterion given in Eq. (9.10).

These ideas form the core of a program called SLIPPER for learning rule-sets, although SLIPPER also incorporates the following variations: First, the greedy approach just described tends to find overly specialized rules which also tend to overfit. To prevent this, on each round of boosting, the training data is randomly split into a growing set and a pruning set. A conjunction is grown as above using the growing set, but then this conjunction is pruned back by choosing the pruning (truncation of the conjunction) that optimizes our usual criterion on the pruning set. Second, for enhanced interpretability, SLIPPER only uses conjunctive rules which predict $+1$, that is, for the positive class; in other words, in the notation of

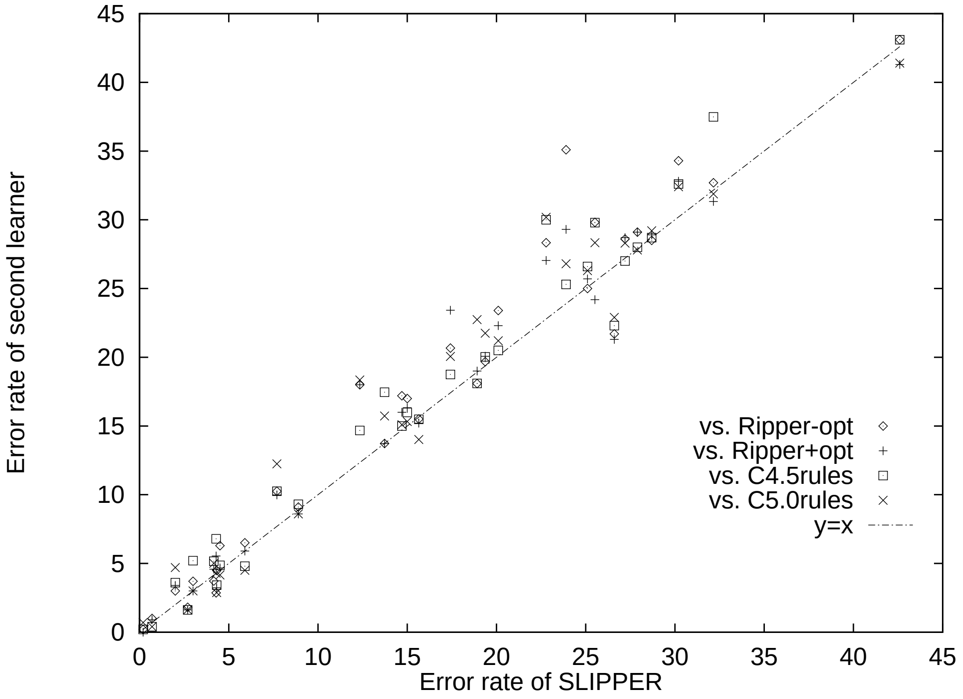

Figure 9.3: Summary of experimental results comparing SLIPPER with some other methods for learning rule-sets. Each point in the plot represents a comparison of SLIPPER's test error (x-coordinate) versus a competing algorithm's test error (y-coordinate) on a single benchmark dataset. [Copyright ©1999 Association for the Advancement of Artificial Intelligence. Reprinted, with permission, from [50].]

Eq. (9.19), s_t is equal to $+1$ for all rules. The exceptions are the constant-value rules which can predict $+1$ on all instances or -1 on all instances (such rules are equivalent to the condition C_t always being equal to **true**). Third, SLIPPER uses a cross-validation technique to choose the number of rules in the set (or equivalently, the number of rounds of boosting). In rough terms, this means the given dataset is repeatedly split into a training set and a validation set. After training on the training set, the best number of rounds is selected based on performance on the validation set. Training is then repeated on the entire dataset for the selected number of rounds.

Figure 9.3 compares the test accuracy of SLIPPER on 32 benchmark datasets with a number of other well-established algorithms for learning rule-sets, namely C4.5rules, C5.0rules, and two variants of RIPPER (see the bibliographic notes for further reading on these). The comparison is evidently quite favorable. Moreover, the rule-sets that are found tend to be reasonably compact compared to most of the other methods, and of a form that is often understandable to humans.

9.4 Alternating decision trees

We turn next to a second application of the confidence-rated framework.

Some of the best performance results for boosting have been obtained using decision trees as base hypotheses. However, when this approach is taken, the resulting combined hypothesis may be quite big, being the weighted majority vote (or thresholded sum) of a possibly large forest of trees, which themselves may individually be rather sizeable. In many cases, the size and complexity of such a combined hypothesis is justified by its high accuracy. But sometimes, it is important to find a classifier that is not only accurate, but also somewhat more compact and understandable. We saw in Section 9.3 how rule-sets can be learned for this purpose. Here, we describe an alternative method in which boosting is used to learn a *single*, though non-standard, decision tree that can often be reasonably compact and comprehensible while still giving accurate predictions. The basic idea is to use weak hypotheses that roughly correspond to *paths* through a tree, rather than an *entire* tree, and to select them in a manner that makes it possible for the combined hypothesis to be arranged conveniently in the form of a tree.

The particular kind of tree that is found in this way is called an *alternating decision tree (ADT)*. Figure 9.4 shows an example of such a tree which resembles, but clearly is also quite distinct from, an ordinary decision tree. An ADT consists of levels that alternate between two types of nodes: *Splitter nodes*, drawn using rectangles in the figure, are each labeled with a test or condition as in ordinary decision trees; while *prediction nodes*, drawn as ellipses, are associated with a real-valued (i.e., confidence-rated) prediction. In an ordinary decision tree, any instance defines a *single* path from the root to a leaf. In an ADT, in contrast, an instance defines *multiple* paths through the tree. For instance, in Figure 9.4, we have shaded the nodes along all of the paths defined by an instance in which $a = 1$ and $b = 0.5$. These paths are determined by starting at the root and working our way down. When a splitter node is reached, we branch next to just one child based on the result of the test associated with the node, just as in an ordinary decision tree. But when a prediction node is reached, we need to traverse to *all* of its children.

The real-valued prediction associated with an ADT on a particular instance is the sum of the values at the prediction nodes along *all* of the paths defined by that instance. For instance, in the example above, this prediction would be

$$0.5 - 0.5 + 0.3 - 0.2 + 0.1 = +0.2.$$

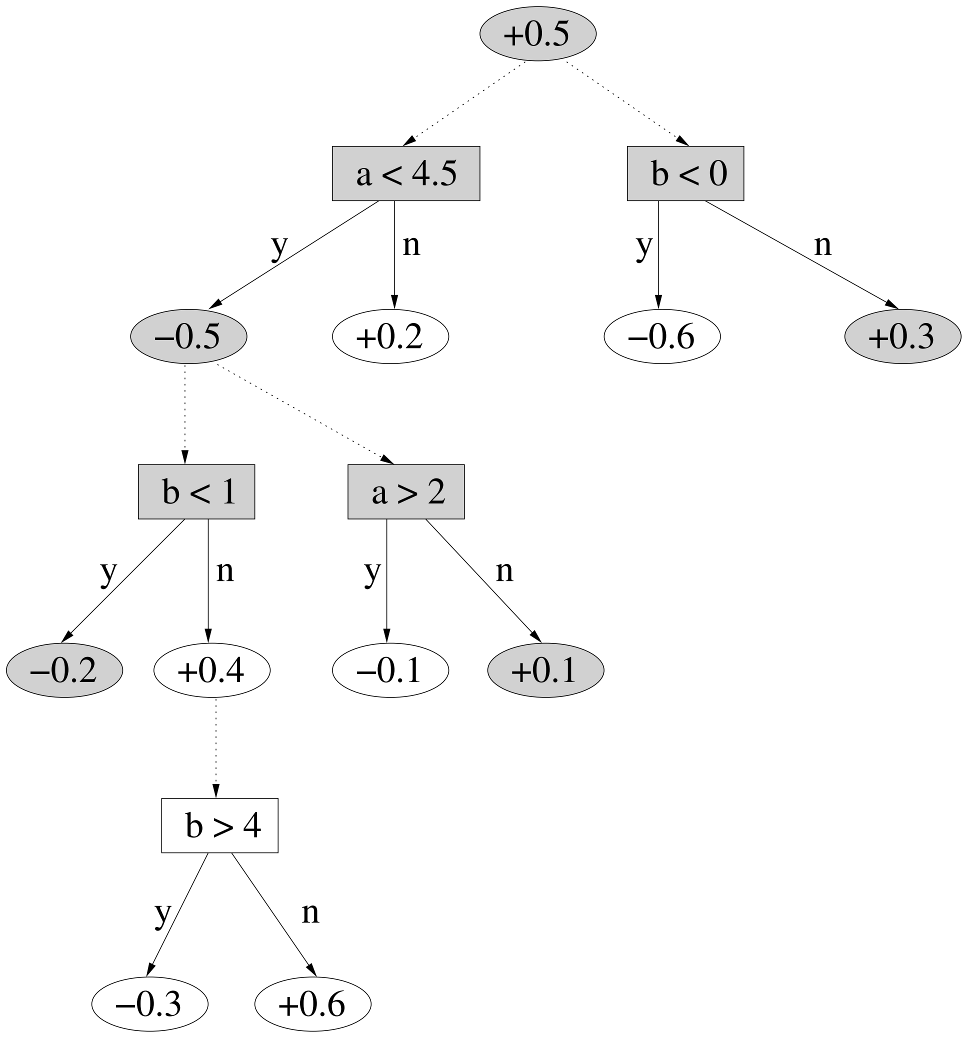

Figure 9.4: An example of an alternating decision tree. Nodes have been shaded along all paths defined by an instance in which $a = 1$ and $b = 0.5$.

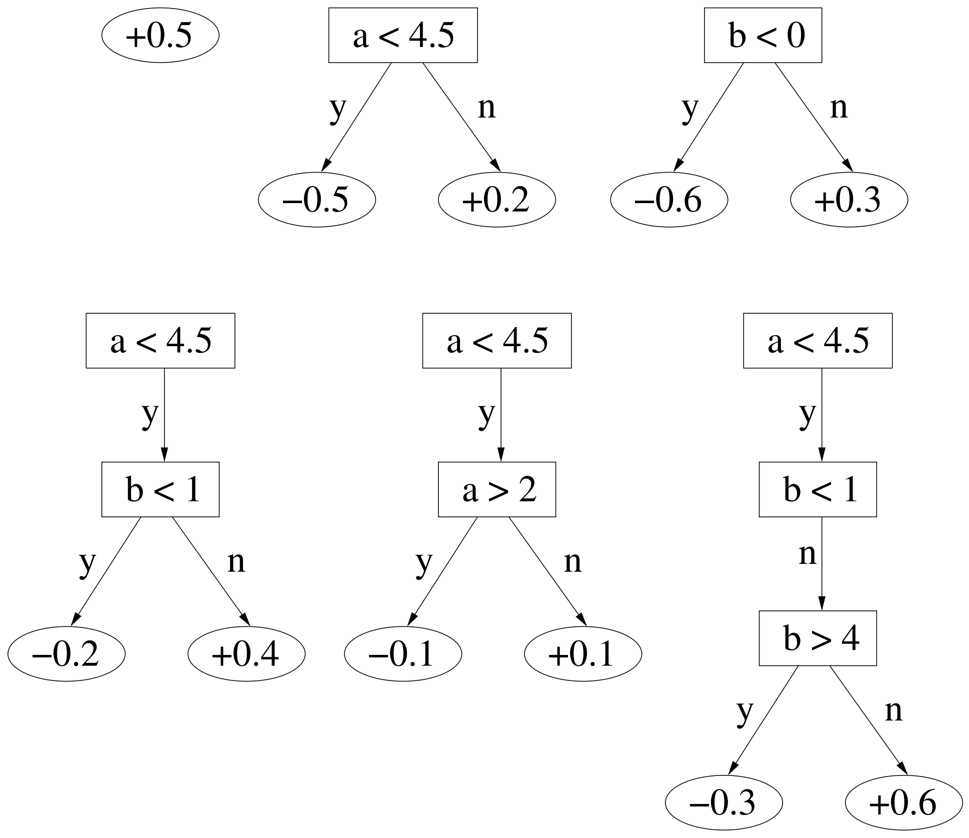

Figure 9.5: A decomposition of the alternating decision tree in Figure 9.4 into six branch predictors.

As usual, taking the sign of this value provides the predicted classification, in this case, $+1$.

In form, ADT's generalize both ordinary decision trees and boosted decision stumps, while preserving much of the comprehensibility of both.

To learn an ADT, we can use boosting with appropriately defined weak hypotheses. To see this, we note that any ADT can be decomposed into a sum of simpler hypotheses, one for each splitter node (as well as the root), and each in the form of a single path or branch through the tree. For instance, the tree in Figure 9.4 can be decomposed into six such *branch predictors* as shown in Figure 9.5. Each of these is evaluated like an ordinary decision tree, but with the proviso that if an evaluation "falls off" the branch, then the result is zero. Thus, the bottom-right branch predictor in the figure evaluates to $+0.6$ on an instance in which $a = 1$ and $b = 2$, but evaluates to 0 on any instance in which $a \geq 4.5$ or $b < 1$. The top-left branch predictor always evaluates to the constant $+0.5$. By the manner in which ADT's are evaluated, it can be seen that the ADT in Figure 9.4 is functionally equivalent, in terms of its predictions, to the sum of the branch predictors in Figure 9.5. Moreover, any ADT can be decomposed in this way.

(Note that there is no ordering associated with these branch predictors. The point is simply that the ADT can be decomposed into an unordered sum of branch predictors. And although the boosting technique described below constructs a set of branch predictors one by one, there will still be considerable variation in the order in which they are added.)

So our approach to learning an ADT is to use boosting with weak hypotheses that are branch predictors as above, but which are appropriately constrained so that the final, resulting set of branch predictors can be arranged as an ADT.

Every branch predictor is defined by a *condition* B given by the test at the last splitter node along the branch, together with a *precondition* P which holds if and only if all the tests along the path to that last splitter node hold. For instance, the bottom-right branch predictor in Figure 9.5 has the condition "$b > 4$" and precondition "$(a < 4.5) \wedge (b \geq 1)$". In general, the branch predictor computes a function of the form

$$h(x) = \begin{cases} 0 & \text{if } P \text{ does not hold on } x \\ c_1 & \text{if } P \text{ holds and } B \text{ holds on } x \\ c_2 & \text{if } P \text{ holds but } B \text{ does not hold on } x \end{cases} \tag{9.21}$$

where c_1 and c_2 are the real-valued predictions at its leaves.

Thus, branch predictors are *both* abstaining and domain-partitioning. Learning such weak hypotheses can be accomplished by straightforwardly combining the techniques of Section 9.2.4 and Section 9.2.6: In general, an abstaining domain-partitioning hypothesis h is associated with a partition of the domain into disjoint blocks $X_0, X_1, \ldots, X_N$ as in Section 9.2.6, but with the added restriction that h abstain on X_0 (so that $h(x) = 0$ for all $x \in X_0$). Then it can be shown, as before, that the best c_j (prediction for h on X_j) for $j = 1, \ldots, N$ is computed as in Eq. (9.14), but for this choice, we have

$$Z = W^0 + 2 \sum_{j=1}^{N} \sqrt{W_+^j W_-^j} \tag{9.22}$$

where

$$W^0 \doteq \sum_{i: x_i \in X_0} D(i) = \Pr_{i \sim D}\left[x_i \in X_0\right].$$

(For simplicity, we are ignoring issues regarding the smoothing of predictions as in Section 9.2.6; these could also be applied here.)

These ideas immediately provide a means of choosing a branch predictor on each round of boosting for a given set $\mathcal{P}$ of candidate preconditions, and set $\mathcal{B}$ of candidate conditions. In particular, for each $P \in \mathcal{P}$ and $B \in \mathcal{B}$, we consider the corresponding branch predictor (Eq. (9.21)) and compute its Z-value as in Eq. (9.22), selecting the one for which this value is minimized. Then the real-valued predictions c_1 and c_2 are given by Eq. (9.14).

What should we use for the sets $\mathcal{P}$ and $\mathcal{B}$? The set of conditions $\mathcal{B}$ can be some set of fixed base conditions, such as those used for decision stumps. As for the set of preconditions, in order that the resulting set of branch predictors be equivalent to an ADT, we need to use preconditions corresponding to paths to splitter nodes that have already been added to the tree. Thus, this set will grow from round to round. In particular, initially the tree is empty and we let $\mathcal{P} = \{\mathbf{true}\}$ where $\mathbf{true}$ is a condition that always holds. When a new branch predictor defined by precondition P and condition B is found on round t, both $P \wedge B$ and $P \wedge \neg B$, corresponding to the two splits of this branch, are added to $\mathcal{P}$.

Finally, for the root node, we initialize the ADT using a weak hypothesis that predicts a constant real value, where this value is set using the methods of Section 9.2. Putting these ideas together leads to Algorithm 9.4. Here, on round t, we write $W_t(C)$ for the sum of the weights of examples for which condition C holds, and among these, we write $W_t^+(C)$ and $W_t^-(C)$ for the total weights of just the positive and negative examples (respectively). These implicitly depend on the current distribution D_t. Thus,

$$W_t^b(C) \doteq \Pr_{i \sim D_t}\left[C \text{ holds on } x_i \ \wedge \ y_i = b\right] \tag{9.23}$$

for $b \in \{-1, +1\}$, and

$$W_t(C) \doteq \Pr_{i \sim D_t}\left[C \text{ holds on } x_i\right]. \tag{9.24}$$

Although the output of this pseudocode is the (thresholded) sum of branch predictors, this sum can immediately be put in the form of an ADT as previously discussed, a data structure that might also provide the basis for the most convenient and efficient implementation. Various other techniques might also be employed for improved efficiency, such as the use of unnormalized weights as discussed in Section 9.2.4.

Because the base hypotheses used by ADT's tend to be rather weak, there is a tendency for the algorithm to overfit; in practice, this typically must be controlled using some form of cross-validation.

```
Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$
    set $\mathcal{B}$ of base conditions.
Initialize:
  • $h_0(x) = \frac{1}{2}\ln((1 + r_0)/(1 - r_0))$ for all $x$ where $r_0 = (1/m)\sum_{i=1}^{m} y_i$.
  • $\mathcal{P} \leftarrow \{\mathbf{true}\}$.
  • $D_1(i) = \begin{cases} 1/(1+r_0) & \text{if } y_i = +1 \\ 1/(1-r_0) & \text{if } y_i = -1. \end{cases}$
For $t = 1, \ldots, T$:
  • Find $P \in \mathcal{P}$ and $B \in \mathcal{B}$ that minimize
      $Z_t = W_t(\neg P) + 2\sqrt{W_t^+(P \wedge B) W_t^-(P \wedge B)} + 2\sqrt{W_t^+(P \wedge \neg B) W_t^-(P \wedge \neg B)}$
    where $W_t^+$, $W_t^-$ and $W_t$ are defined in Eqs. (9.23) and (9.24).
  • Let $h_t$ be the corresponding branch predictor:
      $h_t(x) = \begin{cases} 0 & \text{if } P \text{ does not hold on } x \\ \frac{1}{2}\ln\left(\frac{W_t^+(P \wedge B)}{W_t^-(P \wedge B)}\right) & \text{if } P \text{ holds and } B \text{ holds on } x \\ \frac{1}{2}\ln\left(\frac{W_t^+(P \wedge \neg B)}{W_t^-(P \wedge \neg B)}\right) & \text{if } P \text{ holds but } B \text{ does not hold on } x. \end{cases}$
  • $\mathcal{P} \leftarrow \mathcal{P} \cup \{P \wedge B, P \wedge \neg B\}$.
  • Update:
      $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$.
Output the final hypothesis:
      $H(x) = \operatorname{sign}\left(\sum_{t=0}^{T} h_t(x)\right)$.
```

Algorithm 9.4: The alternating decision tree algorithm.

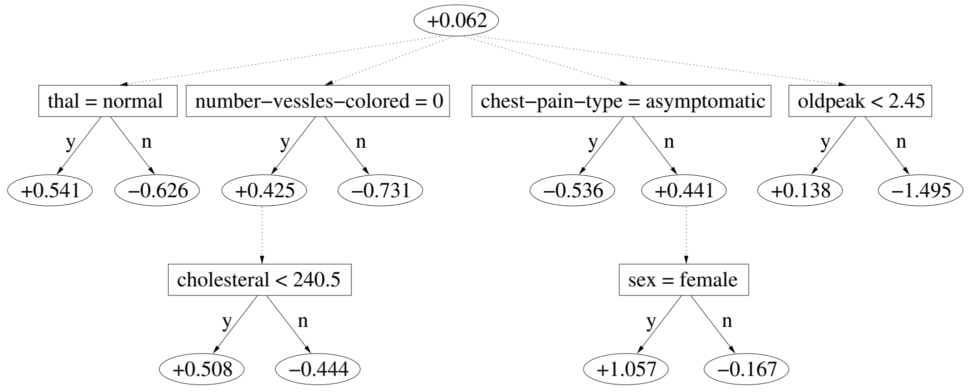

Figure 9.6: The alternating decision tree constructed for the heart disease dataset.

As an illustration of how ADT's can be interpreted, Figure 9.6 shows the tree constructed by this algorithm when run for six rounds on the heart-disease dataset described in Section 1.2.3 where "healthy" and "sick" classes have been identified with labels $+1$ and -1, respectively. For this dataset, ADT's achieve a test error of about 17%, about the same as decision stumps, and better than boosting on decision trees which gives an error around 20% while using a combined hypothesis that may be two orders of magnitude larger. This smaller size already makes ADT's more interpretable, but so does their structure as we now discuss.

To begin, we note that the meaning of splitter nodes can largely be understood in isolation. For instance, from the figure, we can infer that a cholesterol level above 240.5 and an asymptomatic chest pain are both predictors of heart problems as indicated by the fact that they both generate negative contributions to the prediction sum. We also can analyze the interactions of the nodes. Parallel splitter nodes, such as the four nodes in the first level, represent little or no interaction. For instance, the fact that the "thal" test is normal increases the likelihood that the person is healthy irrespective of the "number of vessels colored" or the type of chest pain. In contrast, the significance of the two decision nodes in the second level depends on the evaluation of their ancestral decision nodes. Specifically, regarding the node "sex = female", the fact that a patient is a male appears to be more predictive of a heart problem when chest pain is symptomatic than in the population in general. This implies that only when the chest pain is symptomatic is it worthwhile to consider the patient's gender. The root of the tree is associated with the fixed (unconditional) contribution of 0.062, a small positive number indicating that (according to this training set) there are slightly more healthy people than sick.

Summary

In summary, in this chapter we have explored a general framework for boosting using confidence-rated weak hypotheses. This framework provides general principles for the selection and construction of weak hypotheses, as well as the modification of AdaBoost. Within this framework, we have seen how old algorithms can be adjusted and new algorithms derived, leading to substantial improvements in speed, accuracy and interpretability.

Bibliographic notes

The overall approach taken in this chapter is due to Schapire and Singer [204]. This includes the framework and almost all of the results of Sections 9.1 and 9.2, as well as Algorithm 9.1. However, the highly efficient technique for handling sparse weak hypotheses given in Section 9.2.4 and leading to Algorithms 9.2 and 9.3 is an adaptation of work by Collins [52] (see also Collins and Koo [53]). The experiments in Section 9.2.6, including Table 9.1 and Figure 9.2, are based on results reported by Schapire and Singer [205].

The splitting criterion given in Eq. (9.15) for generating domain-partitioning base hypotheses, including decision trees, was also proposed by Kearns and Mansour [132], although their motivation was rather different. The techniques used for assigning confidences as in Eq. (9.14) to the individual predictions of such base hypotheses, and also for smoothing these predictions as in Eq. (9.16), are closely related to those suggested earlier by Quinlan [182].

In allowing weak hypotheses which can abstain to various degrees, the framework given here is analogous to Blum's "specialist" model of on-line learning [25].

The SLIPPER algorithm and experiments described in Section 9.3, including Figure 9.3, are due to Cohen and Singer [50]. SLIPPER's method for building rules (that is, the weak learner) is similar to that of previous methods for learning rule-sets, particularly Cohen's RIPPER [49] and Fürnkranz and Widmer's IREP [104]. The C4.5rules and C5.0rules algorithms for rule-set induction use decision-tree techniques as developed by Quinlan [183].

The alternating decision tree algorithm of Section 9.4 is due to Freund and Mason [91], including the adapted Figure 9.6. ADT's are similar to the option trees of Buntine [41], developed further by Kohavi and Kunz [137].

Some of the exercises in this chapter are based on material from [54, 132, 204].

Exercises

9-1. Given the same notation and assumptions of Section 9.2.3, let $\tilde{\alpha}$ be the value of α given in Eq. (9.7), and let $\hat{\alpha}$ be the value of α which exactly minimizes

Eq. (9.5). Show that $\tilde{\alpha}$ and $\hat{\alpha}$ have the same sign, and that $|\tilde{\alpha}| \le |\hat{\alpha}|$.

9-2. Suppose the base functions h in $\mathcal{H}$ are confidence-rated with range $[-1, +1]$, that is, $h : \mathcal{X} \to [-1, +1]$. Most of the definitions of margin, convex hull, etc. from Sections 5.1 and 5.2 carry over immediately to this setting without modification. For any $h \in \mathcal{H}$ and any value $\nu \in [-1, +1]$, let

$$h'_{h,\nu}(x) \doteq \begin{cases} +1 & \text{if } h(x) \ge \nu \\ -1 & \text{else} \end{cases}$$

and let

$$\mathcal{H}' \doteq \left\{h'_{h,\nu} : h \in \mathcal{H}, \nu \in [-1, +1]\right\}$$

be the space of all such functions.

(a) For fixed h and x, suppose ν is chosen uniformly at random from $[-1, +1]$. Compute the expected value of $h'_{h,\nu}(x)$.

(b) Let d' be the VC-dimension of $\mathcal{H}'$. Show that the bound given in Theorem 5.5 holds in this setting with probability at least $1 - \delta$ for all $f \in \mathrm{co}(\mathcal{H})$, but with d replaced by d'.

9-3. Let $\mathcal{H}$ and $\mathcal{H}'$ be as in Ex. 9-2.

(a) Show that if a training set is linearly separable with margin $\theta > 0$ using functions from $\mathcal{H}$ (so that Eq. (3.10) holds for some $g_1, \ldots, g_k \in \mathcal{H}$), then the data is γ-empirically weakly learnable by classifiers in $\mathcal{H}'$ (using an exhaustive weak learner) for some $\gamma > 0$.

(b) Prove or disprove that the converse holds in general.

9-4. Let $\mathcal{H} = \{\hbar_1, \ldots, \hbar_N\}$ be a space of weak classifiers, each with range $\{-1, +1\}$. Suppose the sets $P \doteq \{1 \le i \le m : y_i = +1\}$ and $C_j \doteq \{1 \le i \le m : \hbar_j(x_i) = +1\}$ have been precomputed, and that they are quite small compared to m (so that most examples are negative, and the weak classifiers predict -1 on most examples). Show how to implement AdaBoost using an exhaustive weak learner over $\mathcal{H}$ in such a way that:

1. evaluating the weighted error (with respect to distribution D_t) of any particular weak classifier $\hbar_j$ takes time $O(|C_j| + |P|)$ so that an exhaustive weak learner can be implemented in time $O\left(\sum_{j=1}^{N}(|C_j| + |P|)\right)$; and
2. given the currently selected weak classifier $h_t = \hbar_{j_t}$, the running time of the boosting algorithm (*not* including the call to the weak learner) is $O(|C_{j_t}| + |P|)$.

In other words, the running time should only depend on the number of positive examples, and number of examples predicted positive by the weak classifiers.

9-5. Given our usual dataset $(x_1, y_1), \ldots, (x_m, y_m)$, a decision tree can be constructed using a greedy, top-down algorithm. Specifically, let $\mathcal{H}$ be a set of binary functions $h : \mathcal{X} \to \{-1, +1\}$ representing a class of possible splits for the internal nodes of the tree. Initially, the tree consists only of a leaf at its root. Then, on each of a sequence of iterations, one leaf ℓ of the current tree $\mathcal{T}$ is selected and replaced by an internal node associated with some split h, leading to two new leaves depending on the outcome of that split. We write $\mathcal{T}_{\ell \to h}$ to represent the newly formed tree. An example is shown in Figure 9.7.

To describe how ℓ and h are chosen, let $I : [0,1] \to \mathbb{R}_+$ be an *impurity function* for which $I(p) = I(1-p)$, and which is increasing on $[0, 1/2]$. For $b \in \{-1, +1\}$, and for ℓ a leaf of a tree, let $n^b(\ell)$ denote the number of training examples (x_i, y_i) such that x_i reaches leaf ℓ, and $y_i = b$. Also, let $n(\ell) = n^-(\ell) + n^+(\ell)$. Overloading notation, we define the impurity of a leaf ℓ to be $I(\ell) \doteq I(n^+(\ell)/n(\ell))$, and the impurity of the entire tree $\mathcal{T}$ to be

$$I(\mathcal{T}) \doteq \frac{1}{m} \sum_{\ell \in \mathcal{T}} n(\ell) \cdot I(\ell),$$

where summation is over all leaves ℓ of the tree $\mathcal{T}$.

To grow the tree as above, on each iteration, the leaf ℓ and split h is chosen that effects the greatest drop in impurity

$$\Delta I(\ell, h) \doteq \frac{n(\ell)}{m} \cdot \left[I(\ell) - \left(\frac{n(\ell_+^h)}{n(\ell)} \cdot I(\ell_+^h) + \frac{n(\ell_-^h)}{n(\ell)} \cdot I(\ell_-^h) \right) \right]$$

where ℓ_+^h and ℓ_-^h are the two leaves that would be created if ℓ were replaced by an internal node with split h.

(a) Show that $\Delta I(\ell, h) = I(\mathcal{T}) - I(\mathcal{T}_{\ell \to h})$.

A decision tree can be viewed as defining a domain-partitioning hypothesis since the sets of examples reaching each leaf are disjoint from one another. Suppose a *real-valued* prediction is assigned to each leaf so that the tree defines a real-valued function F whose value, on any instance x, is given by the leaf that is reached by x. Suppose further that these values are chosen to minimize the exponential loss on the training set (as in Eq. (7.3)) over all real-valued functions of the particular form specified by the given tree. Let $L(\mathcal{T})$ be the resulting loss for a tree $\mathcal{T}$.

(b) Show that if

$$I(p) \doteq 2\sqrt{p(1-p)} \tag{9.25}$$

then $L(\mathcal{T}) = I(\mathcal{T})$.

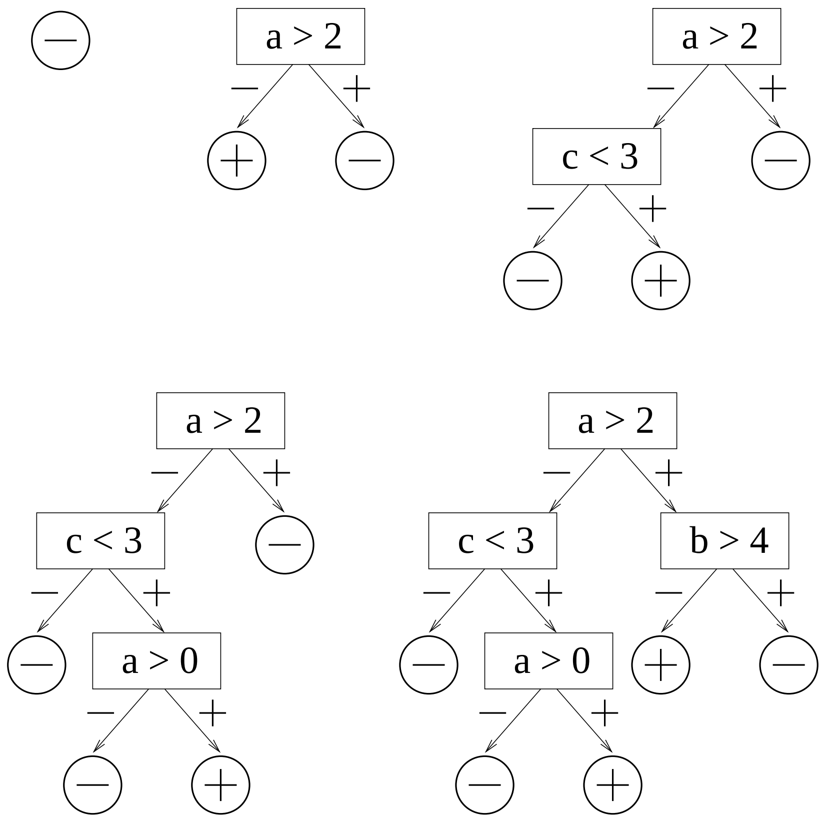

Figure 9.7: Several steps in the construction of a decision tree using decision stumps as splitting functions (with the construction progressing left to right, and then top to bottom). At each step, one leaf is replaced by a new internal node and two new leaves.

(c) For this same choice of impurity function, and for any tree $\mathcal{T}$, show how to assign a binary label in $\{-1, +1\}$ to each leaf ℓ so that the resulting tree-classifier will have training error at most $I(\mathcal{T})$.

(d) Consider using each of the losses listed below in place of exponential loss. In each case, determine how the impurity function $I(p)$ should be redefined so that $L(\mathcal{T}) = I(\mathcal{T})$ (where $L(\mathcal{T})$, as above, is the minimum loss of any real-valued function of the form given by the tree $\mathcal{T}$). Also in each case, explain how to assign a binary label to each leaf, as in part (c), so that the resulting tree-classifier's training error is at most $I(\mathcal{T})$.

   i. Logistic loss (using base-2 logarithm): $(1/m)\sum_{i=1}^{m} \lg(1+\exp(-y_i F(x_i)))$.

   ii. Square loss: $(1/m)\sum_{i=1}^{m}(y_i - F(x_i))^2$.

9-6. Continuing the last exercise, assume henceforth that we are using exponential loss and the impurity function in Eq. (9.25). Also, let us assume the data is empirically γ-weakly learnable by $\mathcal{H}$ (as defined in Section 2.3.3).

(a) Let $\mathcal{T}_{*\to h}$ denote the tree that would result if *every* leaf of the tree $\mathcal{T}$ were replaced by an internal node that splits on h, with each new node leading to two new leaves (so that h appears many times in the tree). For any tree $\mathcal{T}$, show that there exists a split $h \in \mathcal{H}$ such that $L(\mathcal{T}_{*\to h}) \le L(\mathcal{T})\sqrt{1-4\gamma^2}$.

(b) For any tree $\mathcal{T}$ with t leaves, show that there exists a leaf ℓ of $\mathcal{T}$ and a split $h \in \mathcal{H}$ such that

$$\Delta I(\ell, h) \ge \frac{1-\sqrt{1-4\gamma^2}}{t} \cdot I(\mathcal{T}) \ge \frac{2\gamma^2}{t} \cdot I(\mathcal{T}).$$

[*Hint:* For the second inequality, first argue that $\sqrt{1-x} \le 1 - \frac{1}{2}x$ for $x \in [0, 1]$.]

(c) Show that after T rounds of the greedy algorithm described in the last exercise, the resulting tree $\mathcal{T}$ (with binary leaf predictions chosen as in Ex. 9-5(c)) will have training error at most $\exp(-2\gamma^2 \mathrm{H}_T)$ where $\mathrm{H}_T \doteq \sum_{t=1}^{T}(1/t)$ is the T-th harmonic number. (Since $\ln(T+1) \le \mathrm{H}_T \le 1 + \ln T$ for $T \ge 1$, this bound is at most $(T+1)^{-2\gamma^2}$.)

9-7. Let $\mathcal{H} = \{\hbar_1, \ldots, \hbar_N\}$ be a space of real-valued base hypotheses, each with range $[-1, +1]$.

(a) Suppose on each round that the base hypothesis $h_t \in \mathcal{H}$ which maximizes $|r_t|$ as in Section 9.2.3, is selected, and that α_t is chosen as in Eq. (9.7). Show how to modify the proof of Section 8.2 to prove that this algorithm asymptotically minimizes exponential loss (in the same sense as in Theorem 8.4).

(b) Prove the same result when h_t and α_t are instead selected on each round to exactly minimize Z_t (Eq. (9.2)).