7 Loss Minimization and Generalizations of Boosting

In recent years many, if not most, statistical and machine learning methods have been based in one way or another on the optimization of an *objective* or *loss function*. For instance, in the simplest form of linear regression, given examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$, one seeks to find a weight vector $\mathbf{w}$ such that $\mathbf{w} \cdot \mathbf{x}_i$ will be a good approximation of y_i. More precisely, the goal is to find $\mathbf{w} \in \mathbb{R}^n$ minimizing the average (or sum) of the squared errors:

$$L(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2.$$

Here, the squared error of each example $(\mathbf{w} \cdot \mathbf{x}_i - y_i)^2$ is the loss function—in this case, the *square* or *quadratic loss*—and the goal is to minimize the average of the losses over all m examples. A host of other techniques, including neural networks, support-vector machines, maximum likelihood, logistic regression, and many more, can be viewed similarly as optimization of some objective function defined over a set of real-valued parameters.

This approach of defining and then optimizing a specific optimization function has many advantages. First of all, this approach allows us to make the goal of the learning method clear and explicit. This clarity can help tremendously in understanding what a learning method is doing, and in proving properties of the method, for instance, that an iterative procedure eventually converges to something useful. A second major benefit is the decoupling of the objective of learning (minimization of some function) from the particular numerical method that is applied to reach this goal. This means, for instance, that fast, general-purpose numerical methods can be developed and applied to a range of learning objectives. Finally, objective functions can often be easily modified to fit new learning challenges; a number of examples are given in this chapter.

All this leads to the question of whether AdaBoost, too, like so many other modern learning methods, is in fact a procedure for optimizing some associated objective function. Certainly, AdaBoost was not designed with this purpose in mind. Nevertheless, as will be

seen in this chapter, there is indeed a loss function, called the exponential loss, that AdaBoost turns out to greedily minimize. This realization is helpful in a number of ways. First, the fact that AdaBoost is minimizing this particular loss function helps us to understand the algorithm, and is useful in extending the algorithm, for instance, as a tool for estimating conditional probabilities. Second, the AdaBoost algorithm itself can be viewed as a particular technique for minimizing this loss function. This understanding means that AdaBoost can be generalized to handle loss functions other than exponential loss, thus admitting the derivation of boosting-like procedures for other purposes, such as regression (prediction of real-valued labels).

As an important example, these insights help to expose the close connection between AdaBoost and logistic regression, one of the oldest and most widely used statistical approaches for learning to classify discretely labeled data. As we will see, the exponential loss function associated with AdaBoost is related to the loss function for logistic regression. Moreover, AdaBoost can be almost trivially modified to minimize logistic regression's loss function. This view also helps us to see how the predictions made by AdaBoost can be used to estimate the *probability* of a particular example being labeled $+1$ or -1, rather than the classification problem of predicting the most likely label, which we have focused on through most of this book. Finally, this view provides a unified framework in which AdaBoost and logistic regression can be regarded as sibling algorithms in the context of convex optimization and information geometry, topics that will be explored further in chapter 8.

Although the interpretation of AdaBoost as a method for optimizing a particular objective function is very useful, a certain note of caution is in order. It is indisputable that AdaBoost minimizes exponential loss. Nevertheless, this does not mean that AdaBoost's effectiveness comes as a direct consequence of this property. Indeed, we will see that other methods for minimizing the same loss associated with AdaBoost can perform arbitrarily poorly. This means that AdaBoost's effectiveness must in some way follow from the particular dynamics of the algorithm—not just *what* it is minimizing, but *how* it is doing it.

This chapter also studies *regularization*, a commonly used "smoothing" technique for avoiding overfitting by bounding the magnitude of the weights computed on the base classifiers. Regularization and boosting turn out to be linked fundamentally. In particular, we will see that the behavior of α-Boost (a variant of AdaBoost encountered in section 6.4.3), when run for a limited number of rounds, can be regarded as a reasonable approximation of a particular form of regularization. In other words, stopping boosting after fewer rounds can be viewed in this sense as a method of regularization which may be appropriate when working with limited or especially noisy data that might otherwise lead to overfitting. Further, when applied in its weakest form, we will see that regularization produces classifiers with margin-maximizing properties similar to those at the core of our understanding of AdaBoost, as seen in chapter 5.

As further examples of how the general techniques presented in this chapter can be applied, we show in closing how two practical learning scenarios which arise naturally as a result of limitations in the availability of data might be handled through the careful design of an appropriate loss function.

7.1 AdaBoost's Loss Function

So what is the loss function that is naturally associated with AdaBoost? For most of this book, the emphasis has been on minimizing the probability of making an incorrect prediction. That is, the loss of interest for a classifier H on labeled example (x, y) has been the *classification loss* or *0-1 loss*

$$\mathbf{1}\{H(x) \neq y\},$$

which is equal to 1 if the classifier H misclassifies (x, y), and 0 otherwise. Indeed, chapter 3 focused on deriving bounds on AdaBoost's training error

$$\frac{1}{m}\sum_{i=1}^{m}\mathbf{1}\{H(x_i) \neq y_i\} \tag{7.1}$$

where $(x_1, y_1), \ldots, (x_m, y_m)$ is the given training set, and where, as before, H is the combined classifier of the form

$$H(x) = \text{sign}(F(x)),$$

and

$$F(x) \doteq \sum_{t=1}^{T}\alpha_t h_t(x) \tag{7.2}$$

is the linear combination of weak classifiers computed by AdaBoost.

So is AdaBoost a method for minimizing the objective function in equation (7.1)? The answer is "no," in the sense that it can be shown that AdaBoost will not necessarily find the combined classifier of the form above that minimizes equation (7.1). In fact, this problem turns out to be NP-complete, meaning that no polynomial-time algorithm is believed to exist for it. Moreover, on close inspection of the proof of theorem 3.1, we can see that at least with regard to the choice of α_t's, the algorithm was not optimized for the purpose of minimizing the training error in equation (7.1) per se, but rather an *upper bound* on the training error.

This is brought out most clearly in equation (3.3) of the proof of theorem 3.1 (where, throughout the current discussion, we fix D_1 to the uniform distribution). There, we upper bounded the training error

Algorithm 7.1
A greedy algorithm for minimizing exponential loss

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $F_0 \equiv 0$.
For $t = 1, \ldots, T$:

- Choose $h_t \in \mathcal{H}$, $\alpha_t \in \mathbb{R}$ to minimize

$$\frac{1}{m} \sum_{i=1}^{m} \exp(-y_i (F_{t-1}(x_i) + \alpha_t h_t(x_i)))$$

  (over all choices of α_t and h_t).
- Update:

$$F_t = F_{t-1} + \alpha_t h_t.$$

Output F_T.

$$\frac{1}{m} \sum_{i=1}^{m} \mathbf{1}\{\text{sign}(F(x_i)) \neq y_i\} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}\{y_i F(x_i) \leq 0\}$$

by the *exponential loss*

$$\frac{1}{m} \sum_{i=1}^{m} e^{-y_i F(x_i)}, \tag{7.3}$$

using the bound $\mathbf{1}\{x \leq 0\} \leq e^{-x}$. This is the only step where an inequality was used in that proof; all other steps, including, for instance, the greedy choice of α_t, involved strict equalities. Thus, in a nutshell, AdaBoost's training error was first upper bounded by the exponential loss in equation (7.3), which in turn is greedily minimized by the algorithm.

We claim that AdaBoost is in fact a greedy procedure for minimizing equation (7.3). More precisely, consider algorithm 7.1, which iteratively constructs a linear combination $F = F_T$ of the form given in equation (7.2), on each round choosing α_t and h_t so as to cause the greatest decrease in the exponential loss of equation (7.3). We claim that this greedy procedure is equivalent to AdaBoost, making the same choices of α_t and h_t if given the same data and base hypothesis space (and assuming throughout that we are using an exhaustive weak learner that always chooses h_t to minimize the weighted training error ϵ_t over all $h_t \in \mathcal{H}$). The proof of this is embedded in the proof of theorem 3.1. Note first that, using

the notation from theorem 3.1, equation (3.2) of that proof shows that on any round t, and for all examples i,

$$\frac{1}{m}e^{-y_i F_{t-1}(x_i)} = D_t(i)\left(\prod_{t'=1}^{t-1} Z_{t'}\right). \tag{7.4}$$

This implies that

$$\begin{aligned}\frac{1}{m}\sum_{i=1}^{m} e^{-y_i F_t(x_i)} &= \frac{1}{m}\sum_{i=1}^{m}\exp(-y_i(F_{t-1}(x_i)+\alpha_t h_t(x_i))) \\ &= \sum_{i=1}^{m} D_t(i)\left(\prod_{t'=1}^{t-1} Z_{t'}\right)e^{-y_i\alpha_t h_t(x_i)} \\ &\propto \sum_{i=1}^{m} D_t(i)e^{-y_i\alpha_t h_t(x_i)} \doteq Z_t\end{aligned}$$

(where $f \propto g$ here means that f is equal to g times a positive constant that does not depend on α_t or h_t). Thus, minimizing the exponential loss on round t as in algorithm 7.1 is equivalent to minimizing the normalization factor Z_t. Moreover, in equation (3.7) we showed that for a given h_t with weighted error ϵ_t,

$$Z_t = e^{-\alpha_t}\ (1-\epsilon_t) + e^{\alpha_t}\ \epsilon_t,$$

an expression that is minimized for exactly the choice of α_t used by AdaBoost. Thus, for a given h_t, α_t greedily minimizes the exponential loss for round t. Furthermore, plugging in this minimizing choice of α_t gives

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}, \tag{7.5}$$

which is monotonically increasing for $0 \le \epsilon_t \le \frac{1}{2}$, and decreasing for $\frac{1}{2} \le \epsilon_t \le 1$ (see figure 7.1). Thus, the combination of α_t and h_t that minimizes exponential loss for round t is found by choosing α_t as above after first choosing h_t with weighted error ϵ_t as far from $\frac{1}{2}$ as possible; or, alternatively, assuming that $-h$ can be chosen whenever h can be, this is the same as choosing ϵ_t as close to zero as possible. This, of course, is exactly what AdaBoost is doing, in concert with the weak learner.

7.2 Coordinate Descent

When put into slightly different terms, we will see in this section that this basic algorithmic technique turns out to be a numerical method called coordinate descent, which can immediately be applied to other objective functions.

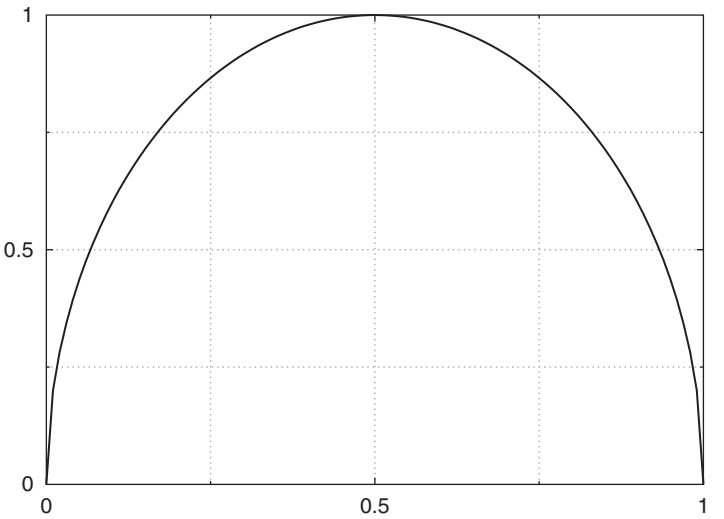


Figure 7.1
A plot of the function $Z(\epsilon) = 2\sqrt{\epsilon(1-\epsilon)}$, as in equation (7.5).

7.2.1 Generalizing AdaBoost

Suppose for simplicity that we are working over a finite space $\mathcal{H}$ of N base hypotheses. Because the space is finite, we can list all of its N members explicitly so that

$$\mathcal{H} = \{\hbar_1, \ldots, \hbar_N\}.$$

To be clear about the notation, $\hbar_j$ represents the j-th weak hypothesis under an arbitrary but fixed indexing of all the weak hypotheses in $\mathcal{H}$, while h_t, as used in AdaBoost (algorithm 1.1 (p. 5) or, equivalently, algorithm 7.1), represents the weak hypothesis from $\mathcal{H}$ that was selected on round t. Note that N, though assumed to be finite, is typically extremely large.

We know that AdaBoost seeks a linear combination of $\mathcal{H}$ as in equation (7.2). Since each h_t is equal to some $\hbar_j \in \mathcal{H}$, such a combination can be re-expressed in the new notation as

$$F_{\boldsymbol{\lambda}}(x) \doteq \sum_{j=1}^{N} \lambda_j \hbar_j(x) \tag{7.6}$$

for some set of weights $\boldsymbol{\lambda}$. Further, the exponential loss function to be minimized can be written as

$$L(\lambda_1, \ldots, \lambda_N) \doteq \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i F_{\boldsymbol{\lambda}}(x_i))$$

Algorithm 7.2
A generic greedy coordinate descent algorithm

Goal: minimization of $L(\lambda_1, \ldots, \lambda_N)$.
Initialize: $\lambda_j \leftarrow 0$ for $j = 1, \ldots, N$.
For $t = 1, \ldots, T$:

- Let j, α minimize $L(\lambda_1, \ldots, \lambda_{j-1}, \lambda_j + \alpha, \lambda_{j+1}, \ldots, \lambda_N)$ over $j \in \{1, \ldots, N\}, \alpha \in \mathbb{R}$.
- $\lambda_j \leftarrow \lambda_j + \alpha$.

Output $\lambda_1, \ldots, \lambda_N$.

$$= \frac{1}{m} \sum_{i=1}^{m} \exp\left(-y_i \sum_{j=1}^{N} \lambda_j \hbar_j(x_i) \right). \tag{7.7}$$

As we have seen, AdaBoost behaves as though the goal were minimization of this loss, which we have here expressed as a real-valued function L over N real-valued parameters, or weights, $\lambda_1, \ldots, \lambda_N$. The method that it uses is to select, on each round t, a weak classifier $h_t \in \mathcal{H}$ and a real number α_t, and then to add a new term $\alpha_t h_t$ to F_{t-1} as in the update step of algorithm 7.1. Since h_t is in $\mathcal{H}$, it must be the same as some $\hbar_j$, so choosing h_t is equivalent to choosing one of the weights λ_j. Further, adding $\alpha_t h_t$ to F_{t-1} is then equivalent to adding α_t to λ_j, that is, applying the update

$$\lambda_j \leftarrow \lambda_j + \alpha_t.$$

Thus, on each round, AdaBoost adjusts just *one* of the weights λ_j. Moreover, the argument given in section 7.1 shows that both the weight λ_j and the adjustment α_t are chosen so as to cause the greatest decrease in the loss function L.

In this sense, AdaBoost can be regarded as a *coordinate descent* method which seeks to minimize its objective function L by iteratively descending along just one coordinate direction at a time. Generic pseudocode for coordinate descent is given as algorithm 7.2. This is in contrast, say, to ordinary gradient descent, which we discuss in section 7.3, and which adjusts *all* of the weights $\lambda_1, \ldots, \lambda_N$ simultaneously on every iteration. When the size N of the weak-classifier space $\mathcal{H}$ is very large (as is typically the case), a sequential update procedure like coordinate descent may make more sense since it leads to a sparse solution, that is, one in which the vast majority of the λ_j's remain equal to zero. This has clear computational benefits since many computations can be carried out without regard to the base hypotheses that have zero weight; indeed, it is for this reason that AdaBoost's running time does not depend at all on the total number of base hypotheses in $\mathcal{H}$ (although

the weak learner's running time might). There may also be statistical benefits, as was seen in section 4.1, where we proved generalization bounds that depended directly on T, the number of nonzero weights, but only logarithmically on $N = |\mathcal{H}|$, the total number of weights.

One still needs to search, on each round, for the best single weight to update, but in many cases this search, at least approximately, can be carried out efficiently. For instance, in AdaBoost this amounts to the familiar search for a base classifier with minimum weighted error, and can be carried out using any standard learning algorithm.

7.2.2 Convergence

The exponential loss function L in equation (7.7) can be shown to be convex in the parameters $\lambda_1, \ldots, \lambda_N$ (see appendix A.7). This is a very nice property because it means that a search procedure like coordinate descent cannot get stuck in local minima since there are none. If the algorithm reaches a point $\boldsymbol{\lambda}$ at which no adjustment along a coordinate direction leads to a lower value of L, then it must be that the partial derivative $\partial L/\partial \lambda_j$ along any coordinate λ_j is equal to zero. This implies that the *gradient*

$$\nabla L \doteq \left\langle \frac{\partial L}{\partial \lambda_1}, \ldots, \frac{\partial L}{\partial \lambda_N} \right\rangle$$

is also equal to zero, which, since L is convex, is enough to conclude that a global minimum has been reached.

These facts, however, are not in themselves sufficient to conclude that such a global minimum will ever be reached. In fact, even though L is convex and nonnegative, it is entirely possible for it not to attain a global minimum at *any* finite value of $\boldsymbol{\lambda}$. Instead, its minimum might be attained only when some or all of the λ_j's have grown to infinity in a particular direction. For instance, for an appropriate choice of data, the function L could be

$$L(\lambda_1, \lambda_2) = \tfrac{1}{3}\left(e^{\lambda_1 - \lambda_2} + e^{\lambda_2 - \lambda_1} + e^{-\lambda_1 - \lambda_2}\right).$$

The first two terms together are minimized when $\lambda_1 = \lambda_2$, and the third term is minimized when $\lambda_1 + \lambda_2 \to +\infty$. Thus, the minimum of L in this case is attained when we fix $\lambda_1 = \lambda_2$, and the two weights together grow to infinity at the same pace.

Despite these difficulties, it will be proved in chapter 8 that coordinate descent—that is, AdaBoost—does indeed converge asymptotically to the global minimum of the exponential loss.

7.2.3 Other Loss Functions

Clearly, this coordinate-descent approach to function minimization can be applied to other objective functions as well. To be easy to implement, effective, and efficient, the objective

function L must be amenable to an efficient search for the best coordinate to adjust, and the amount of adjustment must also be easy to compute. Moreover, to avoid local minima, convexity and smoothness of the function L appear to be useful qualities.

For example, all of the same ideas can be applied to a quadratic loss function in place of the exponential loss. Thus, given data $(x_1, y_1), \ldots, (x_m, y_m)$, where $y_i \in \mathbb{R}$, and given a space of real-valued functions $\mathcal{H} = \{\hbar_1, \ldots, \hbar_N\}$, the goal is to find a linear combination

$$F_{\boldsymbol{\lambda}} = \sum_{j=1}^{N} \lambda_j \hbar_j$$

with low square loss

$$L(\lambda_1, \ldots, \lambda_N) \doteq \frac{1}{m} \sum_{i=1}^{m} (F_{\boldsymbol{\lambda}}(x_i) - y_i)^2.$$

This is standard linear regression, but we imagine here that the cardinality N of $\mathcal{H}$ is enormous—for instance, $\mathcal{H}$ might be the space of all decision trees, a truly vast space of functions. Applying coordinate descent in this case leads to a procedure like algorithm 7.1, but with the exponential loss appearing in that algorithm replaced by

$$\frac{1}{m} \sum_{i=1}^{m} (F_{t-1}(x_i) + \alpha_t h_t(x_i) - y_i)^2.$$

For a given choice of h_t, it can be shown, using straightforward calculus, that the minimizing value of α_t is

$$\alpha_t = \sum_{i=1}^{m} r_i \frac{h_t(x_i)}{\|h_t\|_2^2} \tag{7.8}$$

where r_i is the "residual"

$$r_i \doteq y_i - F_{t-1}(x_i), \tag{7.9}$$

and

$$\|h_t\|_2 = \sqrt{\sum_{i=1}^{m} h_t(x_i)^2}.$$

For this choice of α_t, the change in L is

$$-\frac{1}{m}\left(\sum_{i=1}^{m} r_i \frac{h_t(x_i)}{\|h_t\|_2}\right)^2. \tag{7.10}$$

Thus, h_t should be chosen to maximize (the absolute value of) equation (7.10). This is equivalent, up to a possible sign change in h_t, to saying that h_t should be chosen to minimize

$$\frac{1}{m}\sum_{i=1}^{m}\left(\frac{h_t(x_i)}{\|h_t\|_2} - r_i\right)^2,$$

that is, its ℓ_2-distance to the residuals (after normalizing).

7.3 Loss Minimization Cannot Explain Generalization

From the foregoing, it might seem tempting to conclude that AdaBoost's effectiveness as a learning algorithm is derived from the choice of loss function that it apparently aims to minimize—in other words, that AdaBoost works *only because* it minimizes exponential loss. If this were true, then it would follow that a still better algorithm could be designed using more powerful and sophisticated approaches to optimization than AdaBoost's comparatively meek approach.

However, it is critical to keep in mind that minimization of exponential loss by itself is *not* sufficient to guarantee low generalization error. On the contrary, it is very much possible to minimize the exponential loss (using a procedure other than AdaBoost) while suffering quite substantial generalization error (relative, say, to AdaBoost). We make this point with both a theoretical argument and an empirical demonstration.

Beginning with the former, in the setup given above, our aim is to minimize equation (7.7). Suppose the data is linearly separable so that there exist $\lambda_1, \ldots, \lambda_N$ for which $y_i F_{\boldsymbol{\lambda}}(x_i) > 0$ for all i. In this case, given any such set of parameters $\boldsymbol{\lambda}$, we can trivially minimize equation (7.7) simply by multiplying $\boldsymbol{\lambda}$ by a large positive constant c, which is equivalent to multiplying $F_{\boldsymbol{\lambda}}$ by c so that

$$\frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i F_{c\boldsymbol{\lambda}}(x_i)\right) = \frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i c F_{\boldsymbol{\lambda}}(x_i)\right)$$

must converge to zero as $c \to \infty$. Of course, multiplying by $c > 0$ has no impact on the predictions $H(x) = \text{sign}(F_{\boldsymbol{\lambda}}(x))$. This means that the exponential loss, in the case of linearly separable data, can be minimized by *any* set of separating parameters $\boldsymbol{\lambda}$ multiplied by a large but inconsequential constant. Said differently, knowing that $\boldsymbol{\lambda}$ minimizes the exponential loss in this case tells us nothing about $\boldsymbol{\lambda}$ except that the combined classifier $H(x)$ has zero training error. Otherwise, $\boldsymbol{\lambda}$ is entirely unconstrained. The complexity, or VC-dimension,

of such classifiers is roughly the number of base classifiers N (see lemma 4.1). Since VC-dimension provides both lower and upper bounds on the amount of data needed for learning, this implies that the performance can be quite poor in the typical case that N is very large.

In contrast, given the weak learning assumption, AdaBoost's generalization performance will be much better, on the order of $\log N$, as seen in chapter 5. This is because AdaBoost does not construct an arbitrary zero-training-error classifier, but rather one with large (normalized) margins, a property that does not follow from its status as a method for minimizing exponential loss.

To be more concrete, we consider empirically three different algorithms for minimizing exponential loss and how they compare on a specific dataset. In this experiment, the data was generated synthetically with each instance $\mathbf{x}$ a 10,000-dimensional $\{-1, +1\}$-valued vector, that is, a point in $\{-1, +1\}^{10,000}$. Each of the 1000 training and 10,000 test examples was generated uniformly at random from this space. The label y associated with an instance $\mathbf{x}$ was defined to be the majority vote of three designated coordinates of $\mathbf{x}$; that is,

$$y = \text{sign}(x_a + x_b + x_c)$$

for some fixed and distinct values a, b, and c. The weak hypotheses used were associated with coordinates. Thus, the weak-hypothesis space $\mathcal{H}$ included, for each of the 10,000 coordinates j, a weak hypothesis h of the form $h(\mathbf{x}) = x_j$ for all $\mathbf{x}$. (The negatives of these were also included.)

Three different algorithms were tested on this data. The first was ordinary AdaBoost using an exhaustive weak learner that, on each round, finds the minimum-weighted-error weak hypothesis. In the results below, we refer to this as *exhaustive AdaBoost*.

The second algorithm was *gradient descent* on the loss function given in equation (7.7). In this standard approach, we iteratively adjust $\boldsymbol{\lambda}$ by taking a series of steps, each in the direction that locally causes the quickest decrease in the loss L; this direction turns out to be the negative gradient. Thus, we begin at $\boldsymbol{\lambda} = \mathbf{0}$, and on each round we adjust $\boldsymbol{\lambda}$ using the update

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \alpha \nabla L(\boldsymbol{\lambda})$$

where α is a step size. In these experiments, $\alpha > 0$ was chosen on each round using a *line search* to find the value that (approximately) causes the greatest decrease in the loss in the given direction.

As we will see, gradient descent is much faster than AdaBoost at driving down the exponential loss (where, for the purposes of this discussion, speed is with reference to the number of rounds, not the overall computation time). A third algorithm that is much slower was also tested. This algorithm is actually the same as AdaBoost except that the weak learner does not actively search for the best, or even a good, weak hypothesis. Rather, on

every round, the weak learner simply selects one weak hypothesis h uniformly at random from $\mathcal{H}$, returning either it or its negation $-h$, whichever has the lower weighted error (thus ensuring a weighted error no greater than $\frac{1}{2}$). We refer to this method as *random AdaBoost*.

All three algorithms are guaranteed to minimize the exponential loss (almost surely, in the case of random AdaBoost). But that does *not* mean that they will necessarily perform the same on actual data in terms of classification accuracy. It is true that the exponential loss function L in equation (7.7) is convex, and therefore that it can have no local minima. But that does not mean that the minimum is unique. For instance, the function

$$\tfrac{1}{2}\left(e^{\lambda_1-\lambda_2}+e^{\lambda_2-\lambda_1}\right)$$

is minimized at any values for which $\lambda_1 = \lambda_2$. In fact, in the typical case that N is very large, we expect the minimum of L to be realized at a rather large set of values $\boldsymbol{\lambda}$. The fact that two algorithms both minimize L only guarantees that both solutions will be in this set, telling us essentially nothing about their relative accuracy.

The results of these experiments are shown in table 7.1. Regarding speed, the table shows that, as commented above, gradient descent is extremely fast at minimizing exponential loss, while random AdaBoost is unbearably slow, though eventually effective. Exhaustive AdaBoost is somewhere in between.

As for accuracy, the table shows that both gradient descent and random AdaBoost performed very poorly on this data, with test errors never dropping significantly below 40%. In contrast, exhaustive AdaBoost quickly achieved and maintained perfect test accuracy beginning after the third round.

Of course, this artificial example is not meant to show that exhaustive AdaBoost is always a better algorithm than the other two methods. Rather, the point is that AdaBoost's strong performance as a classification algorithm cannot be credited—at least not exclusively—to

Table 7.1
Results of the experiment described in section 7.3

	% Test Error [# Rounds]					
Exp. Loss	Exhaustive AdaBoost		Gradient Descent		Random AdaBoost	
10^{-10}	0.0	[94]	40.7	[5]	44.0	[24,464]
10^{-20}	0.0	[190]	40.8	[9]	41.6	[47,534]
10^{-40}	0.0	[382]	40.8	[21]	40.9	[94,479]
10^{-100}	0.0	[956]	40.8	[70]	40.3	[234,654]

The numbers in brackets are the number of rounds required for each algorithm to reach specified values of the exponential loss. The unbracketed numbers show the percent test error achieved by each algorithm at the point in its run where the exponential loss first dropped below the specified values. All results are averaged over ten random repetitions of the experiment.

its effect on the exponential loss. If this were the case, then any algorithm achieving equally low exponential loss should have equally low generalization error. But this is far from what we see in this example where exhaustive AdaBoost's very low exponential loss is matched by the competitors, but their test errors are not even close. Clearly, some other factor beyond its exponential loss must be at work to explain exhaustive AdaBoost's comparatively strong performance.

Indeed, these results are entirely consistent with the margins theory of chapter 5, which does have something direct to say about generalization error. That theory states that the generalization error can be bounded in terms of the number of training examples, the complexity of the base classifiers, and the distribution of the normalized margins on the training set. The first two of these are the same for all three methods tested. However, there were very significant differences in the margin distributions, which are shown in figure 7.2. As can be seen, exhaustive AdaBoost achieves very large margins of at least 0.33 on all of the training examples, in correlation with its excellent accuracy. In sharp contrast, both of the poorly performing competitors had margins below 0.07 on nearly all of the training examples (even lower for random AdaBoost).

Minimization of exponential loss is a fundamental property of AdaBoost, and one that opens the door for a range of practical generalizations of the algorithm. However, the examples in this section demonstrate that any understanding of AdaBoost's generalization

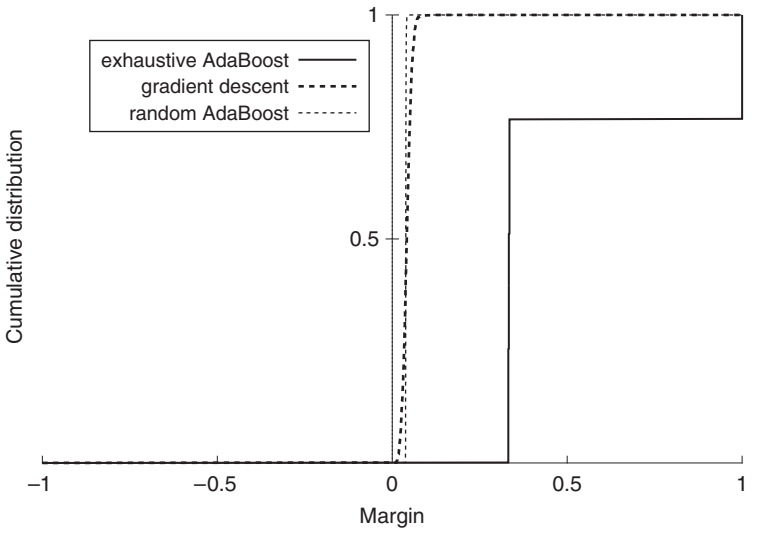


Figure 7.2
The distributions of margins achieved by three algorithms on synthetic data at the point where their exponential loss first dropped below 10^{-40}.

capabilities must in some way take into account the particular dynamics of the algorithm, not just the objective function but also what procedure is actually being used to minimize it.

7.4 Functional Gradient Descent

AdaBoost, as was seen in section 7.2, can be viewed as coordinate-descent minimization of a particular optimization function. This view is useful and general, but can be cumbersome to implement for other loss functions when the choice of best adjustment along the best coordinate is not straightforward to find. In this section, we give a second view of AdaBoost as an algorithm for optimization of an objective function, and we will see that this new view can also be generalized to other loss functions, but in a way that may overcome possible computational difficulties with the coordinate descent view. In fact, in many cases the choice of best base function to add on a given round will turn out to be a matter of finding a classifier with minimum error rate, just as in the case of boosting. Thus, this technique allows the minimization of many loss functions to be reduced to a sequence of ordinary classification problems.

7.4.1 A Different Generalization

In the coordinate descent view, we regarded our objective function, as in equation (7.7), as a function of a set of parameters $\lambda_1, \ldots, \lambda_N$ representing the weights over all of the base functions in the space $\mathcal{H}$. All optimizations then were carried out by manipulating these parameters.

The new view provides a rather different approach in which the focus is on entire *functions*, rather than on a set of parameters. In particular, our objective function $\mathcal{L}$ now takes as input another function F. In the case of the exponential loss associated with AdaBoost, this would be

$$\mathcal{L}(F) \doteq \frac{1}{m} \sum_{i=1}^{m} e^{-y_i F(x_i)}. \tag{7.11}$$

Thus, $\mathcal{L}$ is a *functional*, that is, a function whose input argument is itself a function, and the goal is to find F minimizing $\mathcal{L}$ (possibly with some constraints on F).

In fact, for the purpose of optimizing equation (7.11), we only care about the value of F at $x_1, \ldots, x_m$. Thus, we can think of $\mathcal{L}$ as a function of just the values $F(x_1), \ldots, F(x_m)$, which we can regard as m ordinary real variables. In other words, if we for the moment write $F(x_i)$ as f_i, then our goal can be viewed as that of minimizing

$$\mathcal{L}(f_1, \ldots, f_m) \doteq \frac{1}{m} \sum_{i=1}^{m} e^{-y_i f_i}.$$

In this way, $\mathcal{L}$ can be thought of as a real-valued function on $\mathbb{R}^m$.

How can we optimize such a function? As described in section 7.3, gradient descent is a standard approach, the idea being to iteratively take small steps in the direction of steepest descent, which is the negative gradient. Applying this idea here means repeatedly updating F by the rule

$$F \leftarrow F - \alpha \nabla \mathcal{L}(F) \tag{7.12}$$

where $\nabla \mathcal{L}(F)$ represents the gradient of $\mathcal{L}$ at F, and α is some small positive value, sometimes called the *learning rate*. If we view F only as a function of $x_1, \ldots, x_m$, then its gradient is the vector in $\mathbb{R}^m$,

$$\nabla \mathcal{L}(F) \doteq \left\langle \frac{\partial \mathcal{L}(F)}{\partial F(x_1)}, \ldots, \frac{\partial \mathcal{L}(F)}{\partial F(x_m)} \right\rangle,$$

and the gradient descent update in equation (7.12) is equivalent to

$$F(x_i) \leftarrow F(x_i) - \alpha \frac{\partial \mathcal{L}(F)}{\partial F(x_i)}$$

for $i = 1, \ldots, m$.

This technique is certainly simple. The problem is that it leaves F entirely unconstrained in form, and therefore makes overfitting a certainty. Indeed, as presented, this approach does not even specify meaningful predictions on test points not seen in training. Therefore, to constrain F, we impose the limitation that each update to F must come from some class of base functions $\mathcal{H}$. That is, any update to F must be of the form

$$F \leftarrow F + \alpha h \tag{7.13}$$

for some $\alpha > 0$ and some $h \in \mathcal{H}$. Thus, if each $h \in \mathcal{H}$ is defined over the entire domain (not just the training set), then F will be so defined as well, and hopefully will give meaningful and accurate predictions on test points if the functions in $\mathcal{H}$ are simple enough.

How should we select the function $h \in \mathcal{H}$ to add to F as in equation (7.13)? We have seen that moving in a negative gradient direction $-\nabla \mathcal{L}(F)$ may be sensible, but might not be feasible since updates must be in the direction of some $h \in \mathcal{H}$. What we can do, however, is to choose the base function $h \in \mathcal{H}$ that is *closest* in direction to the negative gradient. Ignoring issues of normalization, this can be done by choosing that $h \in \mathcal{H}$ which maximizes its inner product with the negative gradient (since inner product measures how much two vectors are aligned), that is, which maximizes

$$-\nabla \mathcal{L}(F) \cdot h = -\sum_{i=1}^{m} \frac{\partial \mathcal{L}(F)}{\partial F(x_i)} h(x_i). \tag{7.14}$$

Algorithm 7.3
AnyBoost, a generic functional gradient descent algorithm

Goal: minimization of $\mathcal{L}(F)$.
Initialize: $F_0 \equiv 0$.
For $t = 1, \ldots, T$:

- Select $h_t \in \mathcal{H}$ that maximizes $-\nabla\mathcal{L}(F_{t-1}) \cdot h_t$.
- Choose $\alpha_t > 0$.
- Update: $F_t = F_{t-1} + \alpha_t h_t$.

Output F_T.

Once h has been chosen, the function F can be updated as in equation (7.13) for some appropriate $\alpha > 0$. One possibility is simply to let α be a small positive constant. An alternative is to select α so that $\mathcal{L}$ (or some approximation of $\mathcal{L}$) is minimized by performing a one-dimensional line search.

The general approach that we have described here is called *functional gradient descent*. The resulting procedure, in a general form, is called *AnyBoost*, and is shown as algorithm 7.3.

In the AdaBoost case, the loss function is as given in equation (7.11). The partial derivatives are easily computed to be

$$\frac{\partial\mathcal{L}(F)}{\partial F(x_i)} = \frac{-y_i e^{-y_i F(x_i)}}{m}.$$

Thus, on round t, the goal is to find h_t maximizing

$$\frac{1}{m}\sum_{i=1}^{m} y_i h_t(x_i) e^{-y_i F_{t-1}(x_i)}, \tag{7.15}$$

which, in our standard AdaBoost notation, is proportional to

$$\sum_{i=1}^{m} D_t(i) y_i h_t(x_i) \tag{7.16}$$

by equation (7.4). Assuming h_t has range $\{-1, +1\}$, equation (7.16) can be shown to equal $1 - 2\epsilon_t$, where, as usual,

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

Thus, maximizing equation (7.15) is equivalent to minimizing the weighted error ϵ_t as in AdaBoost. As for the choice of α_t, we have already seen that AdaBoost chooses α_t to minimize exponential loss (see section 7.1). Thus, AdaBoost is a special case of the general functional gradient descent technique given as algorithm 7.3 (assuming exhaustive weak-hypothesis selection). A variant in which a small constant learning rate $\alpha > 0$ is used instead yields the α-Boost algorithm discussed in section 6.4.

Applying this framework to the square loss, as in section 7.2.3, gives a similar algorithm. In this case,

$$\mathcal{L}(F) \doteq \frac{1}{m}\sum_{i=1}^{m}(F(x_i) - y_i)^2$$

so that

$$\frac{\partial \mathcal{L}(F)}{\partial F(x_i)} = \frac{2}{m}\left(F(x_i) - y_i\right).$$

Thus, h_t should be chosen to maximize

$$\frac{2}{m}\sum_{i=1}^{m} h_t(x_i) r_i$$

where r_i is the residual as in equation (7.9). This is nearly the same optimization criterion for choosing h_t as in equation (7.10), but without explicit normalization. Such an h_t could be found, for instance, using a classification learning algorithm as explained in section 7.4.3. Once h_t is selected, the minimizing α_t can be chosen as in equation (7.8).

7.4.2 Relation to Coordinate Descent

The ease of working with the optimization problem that is central to the functional gradient descent approach (namely, maximization of equation (7.14)) is a key practical advantage over the coordinate descent view where we attempted to find a parameter λ_j whose adjustment would cause the greatest drop in the objective function. In fact, however, the two views are rather closely related, and the functional gradient descent view can be derived naturally as an approximation of coordinate descent. In particular, rather than selecting the very best coordinate, we might attempt a compromise between coordinate descent and gradient descent in which we instead select and update the coordinate in whose direction the *negative gradient* is largest. Such a variant of coordinate descent is sometimes called a Gauss-Southwell procedure. Thus, if the optimization function is $L(\lambda_1, \ldots, \lambda_N)$, then on each round of coordinate descent, we select the λ_j for which $-\partial L/\partial \lambda_j$ is largest. If in addition the objective function L can be written in the form

$$L(\lambda_1, \ldots, \lambda_N) = \mathcal{L}(F_{\boldsymbol{\lambda}}),$$

where $F_{\boldsymbol{\lambda}}$ is as in equation (7.6), then by the chain rule from calculus (see appendix A.6), this is equivalent to adjusting that λ_j for which

$$-\frac{\partial L}{\partial \lambda_j} = -\sum_{i=1}^{m} \frac{\partial \mathcal{L}(F_{\boldsymbol{\lambda}})}{\partial F_{\boldsymbol{\lambda}}(x_i)} h_j(x_i)$$

is maximized. This, of course, is exactly what is done in functional gradient descent.

7.4.3 Using Classification and Regression for General Loss Functions

Generalizing what was done above for AdaBoost, we can show that the central problem of maximizing $-\nabla\mathcal{L}(F_{t-1}) \cdot h_t$ on each round of AnyBoost (algorithm 7.3) can be viewed as an ordinary classification problem if each h_t is constrained to have range $\{-1, +1\}$. To see this, let

$$\ell_i = -\frac{\partial \mathcal{L}(F_{t-1})}{\partial F(x_i)},$$

and let

$$\tilde{y}_i = \text{sign}(\ell_i)$$

$$d(i) = \frac{|\ell_i|}{\sum_{i=1}^{m} |\ell_i|}.$$

The problem then is to maximize

$$\begin{aligned}\sum_{i=1}^{m} \ell_i h_t(x_i) &\propto \sum_{i=1}^{m} d(i)\tilde{y}_i h_t(x_i) \\ &= 1 - 2 \sum_{i:\tilde{y}_i \neq h_t(x_i)} d(i) \\ &= 1 - 2\, \mathbf{Pr}_{i\sim d}[\tilde{y}_i \neq h_t(x_i)]\end{aligned}$$

(where $f \propto g$ means that f is equal to g times a positive constant that does not depend on h_t). Thus, to maximize $\nabla\mathcal{L}(F_{t-1}) \cdot h_t$, we can create "pseudolabels" $\tilde{y}_i \in \{-1, +1\}$ as above, and assign a probability weight $d(i)$ to each example. The maximization problem then becomes equivalent to finding a classifier h_t having minimum weighted error with respect to the probability distribution defined by the weights $d(i)$ on a (pseudo) training set $(x_1, \tilde{y}_1), \ldots, (x_m, \tilde{y}_m)$. Note that these pseudolabels $\tilde{y}_i$ vary from round to round, and might or might not agree with any labels which might have been provided as part

of the "real" dataset (although in AdaBoost's case, they always will). Thus, in this fashion, any optimization problem can in principle be reduced to a sequence of classification problems.

Alternatively, rather than seeking a function $h_t \in \mathcal{H}$ that is similar to the negative gradient $-\nabla\mathcal{L}(F_{t-1})$ by maximizing their inner product, we can instead try to minimize the Euclidean distance between them (where we continue to treat these functions as vectors in $\mathbb{R}^m$). That is, the idea is to modify algorithm 7.3 so that rather than maximizing $-\nabla\mathcal{L}(F_{t-1}) \cdot h_t$, we instead attempt to minimize

$$\|-\nabla\mathcal{L}(F_{t-1}) - h_t\|_2^2 = \sum_{i=1}^{m} \left(-\frac{\partial\mathcal{L}(F_{t-1})}{\partial F(x_i)} - h_t(x_i) \right)^2. \tag{7.17}$$

Finding such an h_t is itself a least-squares regression problem where the real-valued pseudolabels now are

$$\tilde{y}_i = -\frac{\partial\mathcal{L}(F_{t-1})}{\partial F(x_i)},$$

so that equation (7.17) becomes

$$\sum_{i=1}^{m} (\tilde{y}_i - h_t(x_i))^2.$$

In this formulation, it is natural to allow h_t to be real-valued and, moreover, to assume that it can be scaled by any constant (in other words, if h is in the class of allowable functions $\mathcal{H}$, then ch is assumed to be as well, for any scalar $c \in \mathbb{R}$). After h_t has been chosen, the weight α_t can be selected using the methods already discussed, such as a line search for the value that causes the greatest drop in loss. Thus, in this way, any loss-minimization problem can be reduced to a sequence of regression problems.

For instance, returning to the example of square loss discussed above, the pseudolabels are proportional to the residuals $\tilde{y}_i = (2/m)r_i$ so that, on each round, the problem is to find h_t that is close, in terms of squared difference, to the residuals (times an irrelevant constant). For this purpose, we might employ a decision-tree algorithm like CART designed, in part, for such regression problems. Once a tree is found, the value of α_t that effects the greatest decrease in square loss can be set as in equation (7.8). (On the other hand, in practice it is often necessary to limit the magnitude of the weights to avoid overfitting, for instance, using regularization, or by selecting α_t that is only a fraction of that given in equation (7.8).) In any case, the resulting combined hypothesis F_T will now be a weighted average of regression trees.

7.5 Logistic Regression and Conditional Probabilities

Next we study the close connection between AdaBoost and logistic regression, beginning with a brief description of the latter method. This will lead both to a boosting-like algorithm for logistic regression and to a technique for using AdaBoost to estimate conditional probabilities.

7.5.1 Logistic Regression

As usual, we assume that we are given data $(x_1, y_1), \ldots, (x_m, y_m)$ where $y_i \in \{-1, +1\}$. We also assume we are given a set of real-valued base functions, or what are sometimes called *features*, $\mathcal{H} = \{h_1, \ldots, h_N\}$. These play a role analogous to weak hypotheses in the context of boosting, and they are formally equivalent. Until now, we have generally taken these base functions/hypotheses to be binary ($\{-1, +1\}$-valued) classifiers, but most of the discussion that follows holds when they are real-valued instead. Boosting using real-valued base hypotheses is studied in greater detail in chapter 9.

In *logistic regression*, the goal is to estimate the *conditional probability* of the label y, given a particular example x, rather than merely to predict whether y is positive or negative. Further, we posit that this conditional probability has a particular parametric form, specifically, a sigmoid function of a linear combination of the features. That is, we posit that instance-label pairs (x, y) are generated randomly in such a way that the true conditional probability of a positive label is equal to

$$\mathbf{Pr}[y = +1 \mid x; \boldsymbol{\lambda}] = \sigma\left(\sum_{j=1}^{N} \lambda_j h_j(x)\right) \tag{7.18}$$

for some setting of the parameters $\boldsymbol{\lambda} = \langle \lambda_1, \ldots, \lambda_N \rangle$, and where

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{7.19}$$

is a sigmoid function with range $[0, 1]$. (See figure 7.3.) As before, let $F_{\boldsymbol{\lambda}}$ be as in equation (7.6). Note that $\sigma(z)$ is greater than, equal to, or smaller than $\frac{1}{2}$ when z is positive, zero, or negative (respectively). Thus, in words, this model is positing that a linear hyperplane in "feature space" (namely, $F_{\boldsymbol{\lambda}}(x) = 0$) separates the points which are more likely to be positive from those which are more likely to be negative. Furthermore, the closer the point is to this separating hyperplane, the more uncertain is its classification.

Note that the conditional probability of a negative label is

$$\begin{aligned}\mathbf{Pr}[y = -1 \mid x; \boldsymbol{\lambda}] &= 1 - \sigma(F_{\boldsymbol{\lambda}}(x)) \\ &= \sigma(-F_{\boldsymbol{\lambda}}(x))\end{aligned}$$

by straightforward algebra. Thus, for $y \in \{-1, +1\}$, we can write

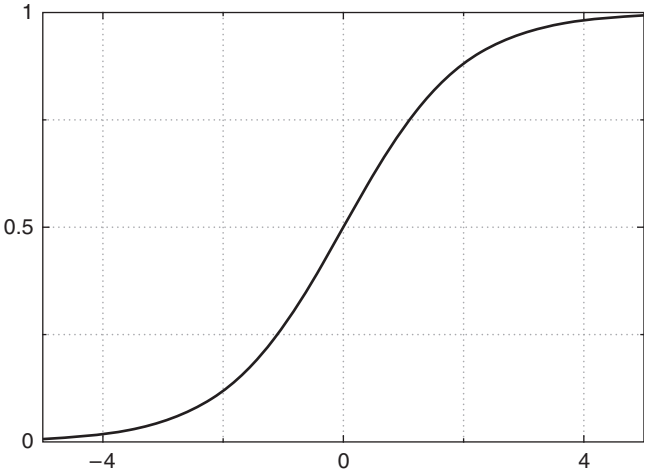


Figure 7.3
A plot of the sigmoid function $\sigma(z)$ given in equation (7.19).

$$\mathbf{Pr}[y \mid x; \boldsymbol{\lambda}] = \sigma(y F_{\boldsymbol{\lambda}}(x)).$$

How can we find the parameters $\boldsymbol{\lambda}$ so that we can estimate these conditional probabilities? A very standard statistical approach is to find the parameters which maximize the conditional likelihood of the data, that is, the probability of observing the given labels y_i, conditioned on the instances x_i. In our case, the conditional likelihood of example (x_i, y_i), for some setting of the parameters $\boldsymbol{\lambda}$, is simply

$$\mathbf{Pr}[y_i \mid x_i; \boldsymbol{\lambda}] = \sigma(y_i F_{\boldsymbol{\lambda}}(x_i)).$$

Thus, assuming independence, the conditional likelihood of the entire dataset is

$$\prod_{i=1}^{m} \sigma(y_i F_{\boldsymbol{\lambda}}(x_i)).$$

Maximizing this likelihood is equivalent to minimizing its negative logarithm, which is (after multiplying by $1/m$) equal to

$$-\frac{1}{m}\sum_{i=1}^{m} \ln \sigma(y_i F_{\boldsymbol{\lambda}}(x_i)) = \frac{1}{m}\sum_{i=1}^{m} \ln\left(1 + e^{-y_i F_{\boldsymbol{\lambda}}(x_i)}\right). \tag{7.20}$$

This is the loss function to be minimized by logistic regression, which we henceforth refer to as *logistic loss*. Once the parameters $\boldsymbol{\lambda}$ which minimize this loss have been found, conditional probabilities of the labels for a test instance x can be estimated as in equation (7.18).

Alternatively, a "hard" classification can be obtained as usual by thresholding, that is, by computing $\text{sign}(F_{\boldsymbol{\lambda}}(x))$.

As discussed in section 7.2, AdaBoost minimizes the exponential loss given in equation (7.7). Since $\ln(1+x) \leq x$ for $x > -1$, it is clear that logistic loss is upper bounded by exponential loss. Moreover, if the natural logarithm in equation (7.20) is replaced by log base 2 (which is the same as multiplying by the constant $\log_2 e$), then logistic loss, like exponential loss, upper bounds the classification loss, that is, the training error

$$\frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{y_i F_{\boldsymbol{\lambda}}(x_i) \leq 0\} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}\{y_i \neq \text{sign}(F_{\boldsymbol{\lambda}}(x_i))\}.$$

The relationship among these loss functions can be seen in figure 7.4. However, as we will see, the connection between exponential loss and logistic loss goes much deeper.

Both exponential and logistic loss give upper bounds on the classification error. Moreover, the two loss functions are very close when the (unnormalized) margin $z = yF_{\boldsymbol{\lambda}}(x)$ is positive. However, they diverge tremendously when z is negative, with exponential loss growing exponentially, but logistic loss growing only linearly (since $\ln(1+e^{-z}) \approx -z$ when z is large and negative). This suggests that logistic loss could be somewhat better behaved in some situations.

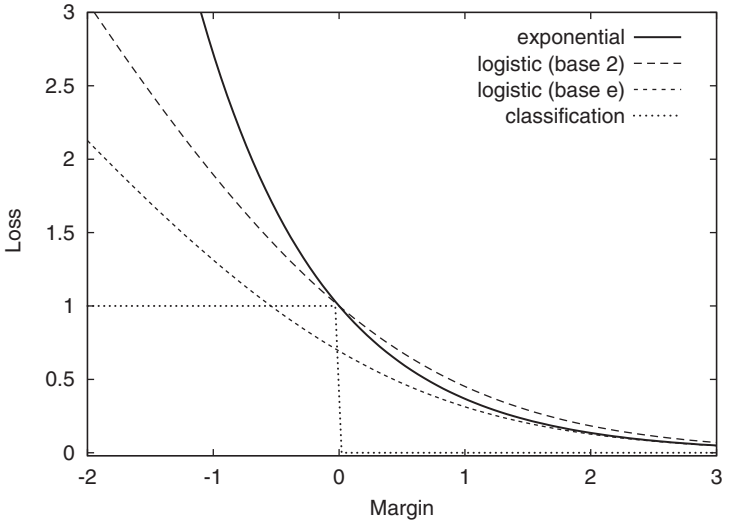


Figure 7.4
A plot of the exponential loss, the logistic loss (using both logarithm base 2 and logarithm base e), and classification loss. Each loss is plotted as a function of the unnormalized margin $yF(x)$.

7.5.2 Modifying AdaBoost for Logistic Loss

How, then, can we minimize the logistic loss function? Standard techniques, based, for instance, on gradient descent or Newton's method, are less effective when the number of base functions is very large. In this situation, a boosting-like approach may be appropriate.

In sections 7.2 and 7.4, we discussed techniques that generalize AdaBoost for minimizing an objective function, so it seems natural to try to apply these to logistic loss. The first of these approaches was coordinate descent, which entails repeatedly finding and adjusting the parameter λ_j that admits the greatest decrease in the objective function. Unfortunately, for logistic loss this turns out to be difficult analytically.

The other approach was functional gradient descent, in which, on each iteration, we select the base function that is closest to the negative functional gradient of the objective function. In this case, the functional of interest is

$$\mathcal{L}(F) = \sum_{i=1}^{m} \ln\left(1 + e^{-y_i F(x_i)}\right), \tag{7.21}$$

and its partial derivatives are

$$\frac{\partial \mathcal{L}(F)}{\partial F(x_i)} = \frac{-y_i}{1 + e^{y_i F(x_i)}}.$$

Thus, this approach prescribes iteratively adding to F a multiple of some base function $h \in \mathcal{H}$ that maximizes

$$\sum_{i=1}^{m} \frac{y_i h(x_i)}{1 + e^{y_i F(x_i)}}. \tag{7.22}$$

In other words, the idea is to weight example i by

$$\frac{1}{1 + e^{y_i F(x_i)}} \tag{7.23}$$

and then to find h most correlated with the labels y_i with respect to this set of weights. These weights are almost the same as for AdaBoost, where the weights

$$e^{-y_i F(x_i)} \tag{7.24}$$

are used instead. However, the weights in equation (7.24) may be highly unbounded, while the weights in equation (7.23) are likely to be much more moderate, always being bounded in the range [0, 1].

Thus, the functional gradient descent approach suggests how to choose a base function h on each round. However, the approach does not specify what multiple of h to add to F, that is, how to select α in the method's iterative update

$$F \leftarrow F + \alpha h. \tag{7.25}$$

Possibilities include a line search to select the α that causes the greatest decrease in $\mathcal{L}(F)$, or simply choosing α to be "suitably small." Neither approach seems to be easily amenable to an analytic treatment.

However, there is another approach that essentially reduces the problem, on each round, to the same sort of tractable minimization encountered for exponential loss. The idea is to derive an upper bound on the change in the loss which can be minimized as a proxy for the actual change in loss. In particular, consider an update as in equation (7.25) where $\alpha \in \mathbb{R}$ and $h \in \mathcal{H}$. We can compute and upper bound the change $\Delta\mathcal{L}$ in the logistic loss when the old F is replaced by $F + \alpha h$ as follows:

$$\begin{aligned}
\Delta\mathcal{L} &\doteq \mathcal{L}(F + \alpha h) - \mathcal{L}(F) \\
&= \sum_{i=1}^{m} \ln\left(1 + e^{-y_i(F(x_i)+\alpha h(x_i))}\right) - \sum_{i=1}^{m} \ln\left(1 + e^{-y_i F(x_i)}\right) \qquad (7.26) \\
&= \sum_{i=1}^{m} \ln\left(\frac{1 + e^{-y_i(F(x_i)+\alpha h(x_i))}}{1 + e^{-y_i F(x_i)}}\right) \\
&= \sum_{i=1}^{m} \ln\left(1 + \frac{e^{-y_i(F(x_i)+\alpha h(x_i))} - e^{-y_i F(x_i)}}{1 + e^{-y_i F(x_i)}}\right) \\
&\le \sum_{i=1}^{m} \frac{e^{-y_i(F(x_i)+\alpha h(x_i))} - e^{-y_i F(x_i)}}{1 + e^{-y_i F(x_i)}} \\
&= \sum_{i=1}^{m} \frac{e^{-y_i \alpha h(x_i)} - 1}{1 + e^{y_i F(x_i)}} \qquad (7.27)
\end{aligned}$$

where the inequality uses $\ln(1 + z) \le z$ for $z > -1$, and each of the equalities follows from simple algebraic manipulations. The idea now is to choose α and h so as to minimize the upper bound in equation (7.27). Conveniently, this upper bound has exactly the same form as the objective that is minimized by AdaBoost on every round. In other words, minimizing equation (7.27) is equivalent to minimizing

$$\sum_{i=1}^{m} D(i) e^{-y_i \alpha h(x_i)}$$

where the weights $D(i)$ are equal (or proportional) to

Algorithm 7.4
AdaBoost.L, a variant of AdaBoost for minimizing logistic loss

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $F_0 \equiv 0$.
For $t = 1, \ldots, T$:

- $D_t(i) = \dfrac{1}{\mathcal{Z}_t} \cdot \dfrac{1}{1 + e^{y_i F_{t-1}(x_i)}}$ for $i = 1, \ldots, m$,
  where $\mathcal{Z}_t$ is a normalization factor.
- Choose $\alpha_t \in \mathbb{R}$ and $h_t \in \mathcal{H}$ to minimize (or approximately minimize if a heuristic search is used):

$$\sum_{i=1}^{m} D_t(i) e^{-y_i \alpha_t h_t(x_i)}.$$

- Update: $F_t = F_{t-1} + \alpha_t h_t$.

Output F_T.

$$\frac{1}{1 + e^{y_i F(x_i)}}.$$

As discussed in section 7.1, this is exactly the form minimized by AdaBoost where on each round $D(i)$ was instead chosen to be $e^{-y_i F(x_i)}$. Thus, as was the case for AdaBoost, if each h is binary, then the best h is the one with minimum weighted error ϵ with respect to D, and the best α is

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

Putting these ideas together leads to algorithm 7.4, called *AdaBoost.L*.[1] As noted earlier, this procedure is identical to AdaBoost, except that in AdaBoost, $D_t(i)$ is instead made proportional to $e^{-y_i F_{t-1}(x_i)}$. In particular, this means that the same weak learner can be used without any changes at all.

With this slight modification in the choice of D_t, it can be proved, using techniques to be developed in chapter 8, that this procedure asymptotically minimizes the logistic loss rather than the exponential loss (see exercise 8.6). But at this point, we can already get

1. This algorithm has sometimes also been called LogAdaBoost and LogitBoost, although the latter name is erroneous since the original LogitBoost algorithm also utilized a Newton-like search that is not incorporated into AdaBoost.L. (See exercises 7.8 and 7.9.)

some intuition as to why the procedure is effective, despite the fact that we are minimizing an upper bound on each round. Consider the derivative, with respect to α, of the upper bound in equation (7.27), compared with the derivative of the actual change in loss in equation (7.26). When evaluated at $\alpha = 0$, the two derivatives are equal to one another; specifically, they are both equal to the negative of equation (7.22). This means that if the bound in equation (7.27) is never negative for any α (so that no improvement is possible), then we must be at a minimum of this function so that its derivative is zero at $\alpha = 0$, which must also be the case for the derivative of the change in loss in equation (7.26). Therefore, if this is the case for all $h \in \mathcal{H}$, so that no progress is possible in the upper bound, then it must be that we have also reached a minimum of the logistic loss (which by convexity must be a global minimum).

Note that this approach is essentially the same as functional gradient descent, but with a particular specification in the choice of α_t. In particular, the weights $D_t(i)$ on examples are the same as for functional gradient descent, and if binary weak hypotheses are used, then the choice of h_t will be identical.

As noted earlier, the weights $D_t(i)$ used in algorithm 7.4 are far less drastic than those for AdaBoost, and in particular never leave the range [0, 1]. This may provide less sensitivity to outliers than with AdaBoost.

7.5.3 Estimating Conditional Probabilities

An apparent advantage of logistic regression is its ability to estimate the conditional probability of y for a given example x. In many contexts, such a capability can be extremely useful since it is so much more informative than an unqualified prediction of the most probable label. In this section, we will see that AdaBoost, which until now has been described strictly as a method for classification, also can be used to estimate conditional probabilities.

Earlier, we derived logistic regression by positing a particular form for these conditional probabilities, and then used this form to derive a loss function. It turns out that the opposite is possible. That is, beginning with the loss function, we can derive an estimate of conditional label probabilities, a technique that can be applied to exponential loss as well.

Let $\ell(z)$ denote the loss function for F on a single example (x, y) as a function of the unnormalized margin $z = yF(x)$. Thus, for logistic regression

$$\ell(z) = \ln(1 + e^{-z}), \tag{7.28}$$

and the goal is to find F minimizing

$$\frac{1}{m} \sum_{i=1}^{m} \ell(y_i F(x_i)). \tag{7.29}$$

This objective function can be regarded as an empirical proxy or estimate for the expected loss

$$\mathbf{E}_{(x,y)\sim\mathcal{D}}[\ell(yF(x))], \tag{7.30}$$

where expectation is over the choice of a random example (x, y) selected from the true distribution $\mathcal{D}$. In other words, we suppose that the ideal goal is minimization of this true *risk*, or expected loss, which in practice must be approximated empirically using a training set.

Let

$$\pi(x) = \mathbf{Pr}_y[y = +1 \mid x]$$

be the true conditional probability that x is positive. Let us suppose for the moment that $\pi(x)$ is known, and let us further suppose that the function F is allowed to be entirely arbitrary and unrestricted (rather than a linear combination of features). How should we choose F? The expectation in equation (7.30) can be rewritten by conditioning on x as

$$\mathbf{E}_x\big[\mathbf{E}_y[\ell(yF(x)) \mid x]\big] = \mathbf{E}_x[\pi(x)\ell(F(x)) + (1-\pi(x))\ell(-F(x))]\,.$$

Thus, the optimal choice of $F(x)$ can be made separately for each x and, in particular, should minimize

$$\pi(x)\ell(F(x)) + (1-\pi(x))\ell(-F(x)).$$

Differentiating this expression with respect to $F(x)$, we see that the optimal $F(x)$ should satisfy

$$\pi(x)\ell'(F(x)) - (1-\pi(x))\ell'(-F(x)) = 0, \tag{7.31}$$

where ℓ' is the derivative of ℓ. For the choice of ℓ given in equation (7.28), this happens to give

$$F(x) = \ln\left(\frac{\pi(x)}{1-\pi(x)}\right).$$

More important for our purposes, this suggests that once an algorithm has computed a function F that approximately minimizes the logistic loss, we can transform $F(x)$ into an estimate of $\pi(x)$ using the inverse of this formula

$$\pi(x) = \frac{1}{1+e^{-F(x)}}, \tag{7.32}$$

which of course is perfectly consistent with the original derivation of logistic regression given in section 7.5.1.

This same approach can be applied to other loss functions as well. Solving equation (7.31) in general for $\pi(x)$ gives

$$\pi(x) = \frac{1}{1+\dfrac{\ell'(F(x))}{\ell'(-F(x))}}.$$

For exponential loss as used by AdaBoost, we have $\ell(z) = e^{-z}$. Plugging in gives

$$\pi(x) = \frac{1}{1 + e^{-2F(x)}}. \tag{7.33}$$

Thus, to convert the output of AdaBoost

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

to a conditional probability, we can simply pass $F(x)$ through the sigmoid given in equation (7.33), a transformation that is almost identical to the one used by logistic regression in equation (7.32).

It is important to note that this approach is founded on two possibly dubious assumptions. First, we assumed that the empirical loss (or risk) in equation (7.29) is a reasonable estimate of the true risk (expected loss) in equation (7.30) for all functions F of interest. This may or may not hold, and indeed is likely to be especially suspect for unbounded loss functions like the logistic and exponential losses. Second, we assumed that the function F computed by logistic regression or AdaBoost is a minimum over *all* functions F without restriction, whereas, as we know, both algorithms compute a function F which is a linear combination of base functions or weak classifiers.

Figure 7.5 shows the result of applying this method to actual data, in this case, the census dataset described in section 5.7.1, using the same setup as before. After training, AdaBoost's combined hypothesis F was used as in equation (7.33) to produce an estimate $\pi(x)$ of the probability of each test example x being positive (income above \$50,000). To produce the *calibration curve* shown in the figure, the 20,000 test examples were sorted by their $\pi(x)$ values, then broken up into contiguous groups of 100 examples each. Each point on each plot corresponds to one such group where the x-coordinate gives the average of the probability estimates $\pi(x)$ for the instances in the group, and the y-coordinate is the fraction of truly positive instances in the group. Thus, if the probability estimates are accurate, then all points should be close to the line $y = x$. In particular, in this figure, with the large training set of size 10,000 that was used, we see that the probability estimates are quite accurate on test data. However, with smaller training sets, performance may degrade significantly for the reasons cited above.

7.6 Regularization

7.6.1 Avoiding Overfitting

Throughout this book, we have discussed the central importance of avoiding overfitting in learning. Certainly, this is important in classification, but when estimating conditional

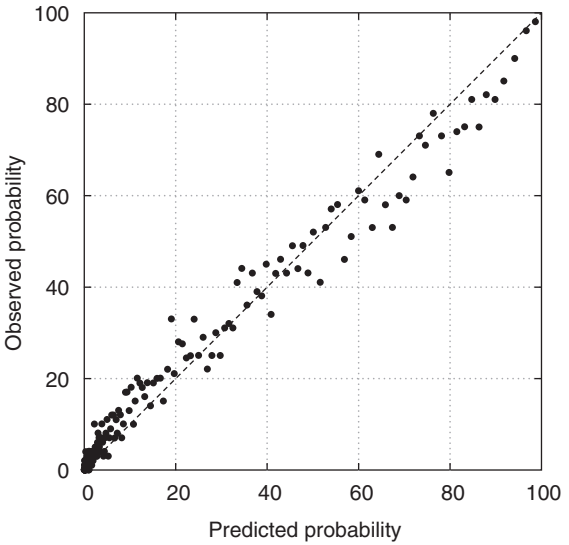


Figure 7.5
A calibration curve for the census dataset. Each point on the plot represents a single cluster of 100 test examples, grouped according to their estimated probability of being positive. The x-coordinate of each point is the average predicted percent probability of a positive label for examples in the group, according to the learned model. The y-coordinate is the actual percent positive in the group. Thus, ideal predictions will be close to the line $y = x$.

probabilities, it can be even more critical. This is because the latter task entails the actual numerical estimation of individual probabilities, while in classification it is enough merely to predict if the probability of an example being positive is larger or smaller than its probability of being negative. Thus, classification demands less of the learning algorithm, and is based on a criterion that can be considerably more forgiving.

The example in section 7.5.3 shows that AdaBoost can effectively estimate probabilities. However, far worse performance is possible when working with smaller training sets or much noisier data. An extreme example is shown in figure 7.6. Here, the artificially generated instances x are real numbers uniformly distributed on the interval $[-2, +2]$, and for any x, the label y is chosen to be $+1$ with probability 2^{-x^2}, and -1 otherwise. This conditional probability, as a function of x, is plotted on the left of the figure. On the right is plotted the estimated conditional probability function computed by running AdaBoost with decision stumps on 500 training examples for 10,000 rounds (and applying equation (7.33)). Clearly, these probability estimates are extremely poor, as are the corresponding classifications. The problem, of course, is excessive overfitting of the noise or randomness in the data.

To avoid overfitting, as we saw in chapter 2, it is necessary to balance how well the learned model fits the data against its complexity. In this example, we have not attempted to limit the complexity of the learned model. Although we might consider restricting the base classifier space $\mathcal{H}$ for this purpose, we here assume that this space has been given and

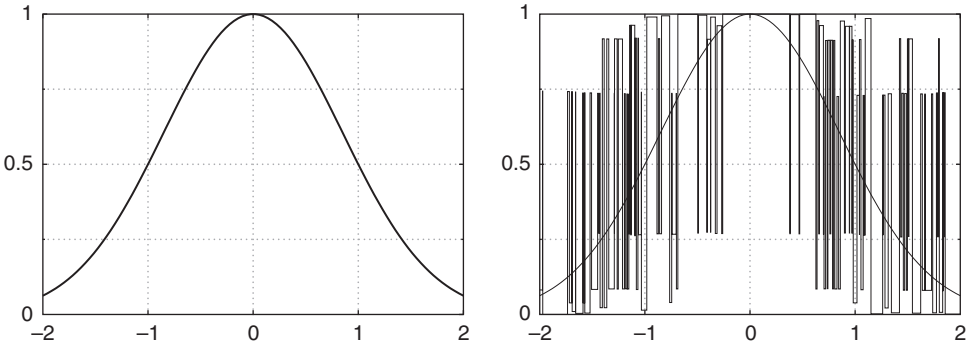


Figure 7.6
Left: A plot of the conditional probability that $y = +1$, given x, on the synthetic example described in section 7.6.1. Right: The result of running AdaBoost with decision stumps on 500 examples generated in such manner. The darker, jagged curve shows the conditional probability as estimated using equation (7.33), overlaid with the true conditional probability.

is fixed so as to emphasize alternative techniques. (Furthermore, in this particular case $\mathcal{H}$ already has very low VC-dimension.) Instead, we focus on the weight vector $\boldsymbol{\lambda}$ computed by AdaBoost as in the formulation given in section 7.2 since, with $\mathcal{H}$ fixed, $\boldsymbol{\lambda}$ entirely defines the learned model.

Since it is a vector in $\mathbb{R}^N$, we can measure the complexity of $\boldsymbol{\lambda}$ naturally by its "size," for instance, its Euclidean length $\|\boldsymbol{\lambda}\|_2$, or as measured using some other norm. Because of its particular relevance to what follows, we focus specifically on the ℓ_1-norm as a complexity measure:

$$\|\boldsymbol{\lambda}\|_1 \doteq \sum_{j=1}^{N} |\lambda_j|.$$

With such a measure in hand, we can explicitly limit complexity by seeking to minimize loss as before, but now subject to a strict bound on the ℓ_1-norm. For exponential loss, this leads to the following constrained optimization problem:

$$\begin{aligned} &\text{minimize: } L(\boldsymbol{\lambda}) \\ &\text{subject to: } \|\boldsymbol{\lambda}\|_1 \leq B \end{aligned} \tag{7.34}$$

for some fixed parameter $B \geq 0$ where, throughout this section, $L(\boldsymbol{\lambda})$ is the exponential loss as defined in equation (7.7). Alternatively, we might define an unconstrained optimization problem that attempts to minimize a weighted combination of the loss and the ℓ_1-norm:

$$L(\boldsymbol{\lambda}) + \beta \|\boldsymbol{\lambda}\|_1 \tag{7.35}$$

for some fixed parameter $\beta \geq 0$. These are both called *regularization* techniques. The two forms given here can be shown to be equivalent in the sense of yielding the same solutions for appropriate settings of the parameters B and β. Note that the minimization in equation (7.35) can be solved numerically, for instance, using coordinate descent as in section 7.2.

Naturally, other loss functions can be used in place of exponential loss, and other regularization terms can be used in place of $\|\boldsymbol{\lambda}\|_1$, such as $\|\boldsymbol{\lambda}\|_2^2$. A favorable property of ℓ_1-regularization, however, is its observed tendency to prefer sparse solutions, that is, vectors $\boldsymbol{\lambda}$ with relatively few nonzero components.

For classification, limiting the norm of $\boldsymbol{\lambda}$ would seem to be of no consequence since scaling $\boldsymbol{\lambda}$ by any positive constant has no effect on the predictions of a classifier of the form $\text{sign}(F_{\boldsymbol{\lambda}}(x))$. However, when combined with loss minimization as above, the effect can be significant for both classification and estimation of conditional probabilities. For instance, continuing the example above, figure 7.7 shows the estimated conditional probability functions that result from minimizing equation (7.35) for various settings of β. As the results on this toy problem illustrate, regularization effectively smooths out noise in the data, leading to much more sensible predictions. But as usual, there is a trade-off, and too much regularization can lead to overly smooth predictions; indeed, at an extreme, $\boldsymbol{\lambda}$ is forced to be $\mathbf{0}$, yielding meaningless probability estimates of $\frac{1}{2}$ on all instances.

7.6.2 A Connection to Boosting with Early Stopping

Regularization based on the ℓ_1-norm turns out to be closely connected to boosting algorithms. In particular, we will see that a simple variant of AdaBoost, when stopped after any number of rounds, can often be viewed as providing an approximate solution to the ℓ_1-regularized constrained optimization problem given in equation (7.34) for a corresponding choice of the parameter B.

To make this precise, for any $B \geq 0$, let $\boldsymbol{\lambda}_B^*$ denote any solution of equation (7.34). Then, as B varies, $\boldsymbol{\lambda}_B^*$ traces out a path or trajectory through $\mathbb{R}^N$. When N is small, we can visualize this trajectory as shown at the top of figure 7.8. In this example, data was taken from the heart-disease dataset described in section 1.2.3. However, for illustrative purposes, rather than using all possible decision stumps for base classifiers, we used only the six stumps shown in table 1.2, so that $N = 6$ in this case.[2] The figure shows all six components of $\boldsymbol{\lambda}_B^*$. Specifically, each curve marked j, for $j = 1, \ldots, 6$, is a plot of the j-th component $\lambda_{B,j}^*$ as a function of B. Thus, the figure depicts the entire trajectory. Notice how, as B is increased, nonzero components are added to $\boldsymbol{\lambda}_B^*$ just one or two at a time so that this solution vector tends to remain sparse as long as possible.

For comparison, we consider next the α-Boost variant of AdaBoost described in section 6.4.3. Recall that this is the same as AdaBoost (algorithm 1.1), except that on each

2. We also modified the predictions of these six stumps so that each predicts $+1$ (healthy) when its condition is satisfied, and -1 (sick) otherwise. Since the weights on the base classifiers can be positive or negative, this change is of no consequence.

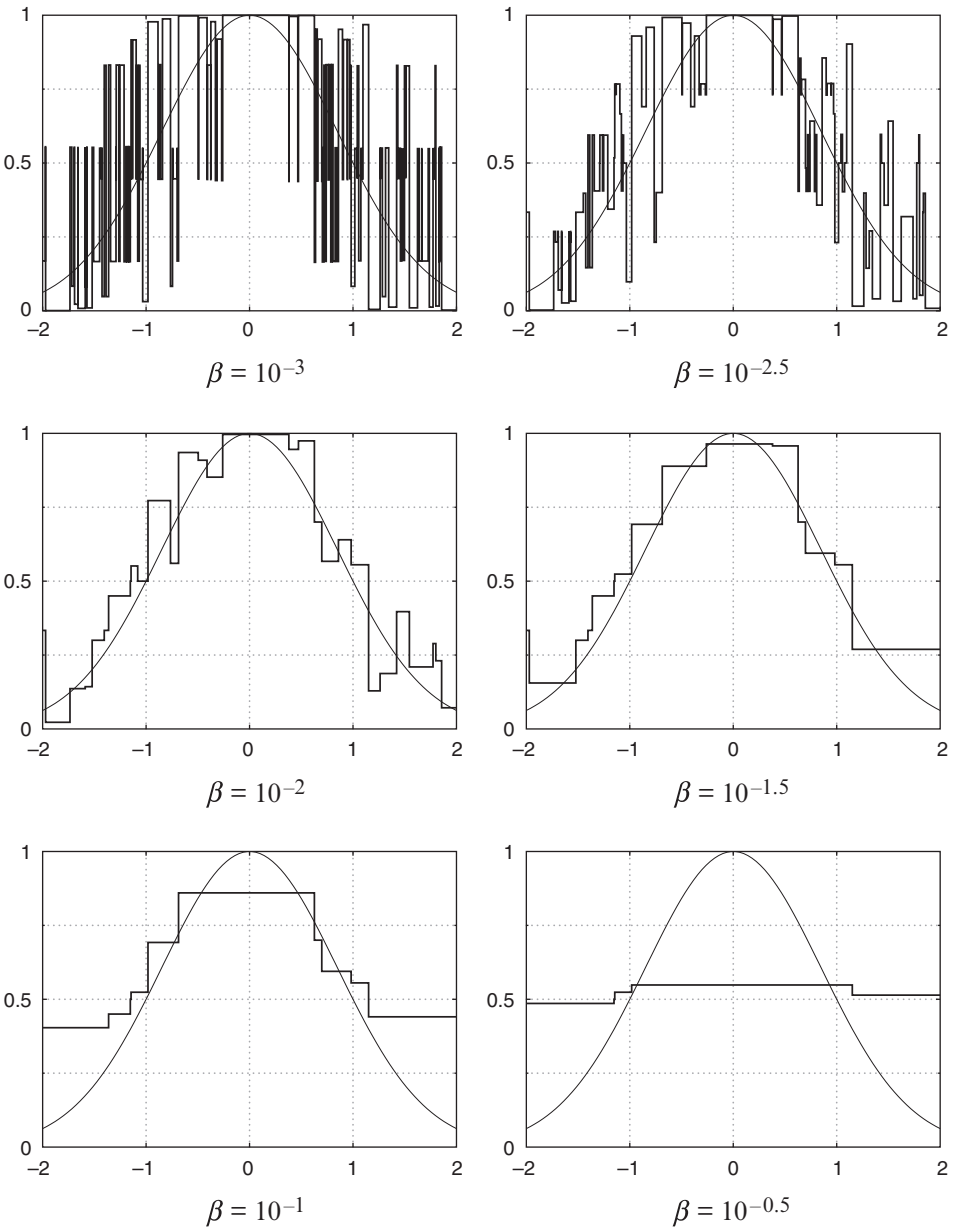


Figure 7.7
The result of minimizing the regularized exponential loss given in equation (7.35), for varying values of β on the same learning problem as in figure 7.6. The darker curves in each figure show the conditional probability as estimated using equation (7.33), overlaid with the true conditional probability.

round, α_t is set equal to some fixed constant α. This is also the same as coordinate descent (section 7.2) or AnyBoost (section 7.4) applied to exponential loss, but using a fixed learning rate α on each iteration. Here, we think of α as a tiny, but positive, constant. Throughout this discussion, we assume that an exhaustive weak learner is used and, furthermore, one which is permitted to choose either a base classifier $\hbar_j \in \mathcal{H}$ or its negation $-\hbar_j$, leading to an update to λ_j of α or $-\alpha$, respectively.

As above, we can plot the trajectory of $\boldsymbol{\lambda}_T$, the weight vector that defines the combined classifier computed by α-Boost after T iterations. This is shown, for the same dataset, in the middle of figure 7.8. Each curve j is a plot of $\lambda_{T,j}$ as a function of time T, multiplied by the constant α, so that the resulting scale αT is equal to the cumulative sum of weight updates after T iterations. (Here, we used $\alpha = 10^{-6}$.)

Remarkably, the two plots at the top of figure 7.8—one for the trajectory of ℓ_1-regularization, and the other for α-Boost—are practically indistinguishable. This shows that, at least in this case, α-Boost, when run for T rounds, computes essentially the same solution vectors as when using ℓ_1-regularization with B set to αT. This also means, of course, that the predictions of the two methods will be nearly identical (for either classification or probability estimation). Thus, early stopping—that is, halting boosting after a limited number of rounds—is in this sense apparently equivalent to regularization.

In fact, this correspondence is known to hold much more generally under appropriate technical conditions. When these conditions do not hold, a variant of α-Boost can be used instead which is known to match the computation of ℓ_1-regularization in general. Although the details are beyond the scope of this book, we can provide some intuition as to why α-Boost's trajectory should be so similar to that of ℓ_1-regularization.

Suppose for some $B \geq 0$ that we have already computed the ℓ_1-regularized solution $\boldsymbol{\lambda} = \boldsymbol{\lambda}^*_B$ for equation (7.34), and that we now wish to compute the solution $\boldsymbol{\lambda}' = \boldsymbol{\lambda}^*_{B+\alpha}$ when B is incremented by α, for some tiny $\alpha > 0$. If we can do this repeatedly, then in principle we can trace out the entire trajectory, or at least a good approximation of it. Let us assume for this discussion that $\|\boldsymbol{\lambda}\|_1 = B$, since if $\|\boldsymbol{\lambda}\|_1 < B$, then it can be argued that $\boldsymbol{\lambda}$ is also a solution for any larger value of B as well (see exercise 7.10). Similarly, assume $\|\boldsymbol{\lambda}'\|_1 = B + \alpha$. Let us define the difference vector $\boldsymbol{\delta} \doteq \boldsymbol{\lambda}' - \boldsymbol{\lambda}$. Clearly, if we can find $\boldsymbol{\delta}$, then we can find $\boldsymbol{\lambda}' = \boldsymbol{\lambda} + \boldsymbol{\delta}$ as well.

By the triangle inequality,

$$B + \alpha = \|\boldsymbol{\lambda}'\|_1 = \|\boldsymbol{\lambda} + \boldsymbol{\delta}\|_1 \leq \|\boldsymbol{\lambda}\|_1 + \|\boldsymbol{\delta}\|_1 = B + \|\boldsymbol{\delta}\|_1. \tag{7.36}$$

In general, equality need not hold here. However, if it happens that the signs of the components of $\boldsymbol{\lambda}$ and $\boldsymbol{\delta}$ align in the right way so that $\lambda_j \delta_j \geq 0$ for all j, then equation (7.36) will indeed hold with equality, implying that $\|\boldsymbol{\delta}\|_1 = \alpha$ in this case. This suggests the heuristic of seeking that $\boldsymbol{\delta}$ with $\|\boldsymbol{\delta}\|_1 = \alpha$ which minimizes $L(\boldsymbol{\lambda}') = L(\boldsymbol{\lambda} + \boldsymbol{\delta})$. When α is small, we have by a Taylor expansion that

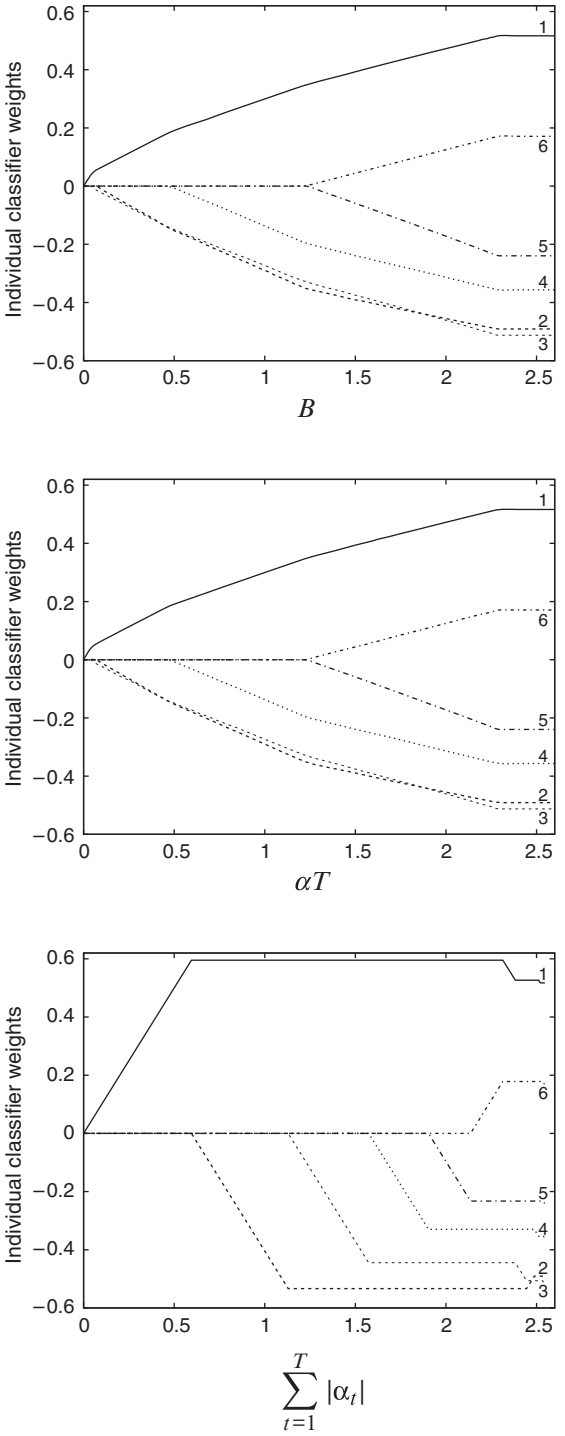


Figure 7.8
(Caption on facing page)

$$L(\boldsymbol{\lambda}+\boldsymbol{\delta}) \approx L(\boldsymbol{\lambda}) + \nabla L(\boldsymbol{\lambda}) \cdot \boldsymbol{\delta} = L(\boldsymbol{\lambda}) + \sum_{j=1}^{N} \frac{\partial L(\boldsymbol{\lambda})}{\partial \lambda_j} \cdot \delta_j.$$

Among all $\boldsymbol{\delta}$ with $\|\boldsymbol{\delta}\|_1 = \alpha$, the right-hand side is minimized when $\boldsymbol{\delta}$ is a vector of all zeros, except for that single component j for which $\left|\partial L(\boldsymbol{\lambda})/\partial \lambda_j\right|$ is largest, which is set to $-\alpha \cdot \text{sign}(\partial L(\boldsymbol{\lambda})/\partial \lambda_j)$.

In fact, the update we have just described for computing $\boldsymbol{\lambda}' = \boldsymbol{\lambda} + \boldsymbol{\delta}$ from $\boldsymbol{\lambda}$ is *exactly* what α-Boost computes on each iteration by the arguments given in section 7.4. Thus, we have derived α-Boost as an approximate method for incrementally following the trajectory of ℓ_1-regularization. However, as noted above, the heuristic depends on the updates $\boldsymbol{\delta}$ having the same component-wise signs as $\boldsymbol{\lambda}$. Informally, this means that the updates applied to the weight λ_j associated with any base classifier h_j should always be of the same sign. It is when this condition holds, as in figure 7.8, that the trajectories will be the same. When it does not hold, when α-Boost occasionally takes a "backwards" step, the correspondence no longer holds exactly.

As a side note, we can compare these trajectories against that of AdaBoost as well. The bottom of figure 7.8 shows such a plot in which the vectors $\boldsymbol{\lambda}$ computed by AdaBoost are shown in a similar manner as before, but now as a function of the cumulative sum of weight updates $\sum_{t=1}^{T} |\alpha_t|$. Compared to the other plots, we can see some correspondence in behavior, but the resemblance is rather coarse and stylized. Thus, this small example suggests that the connection of AdaBoost to ℓ_1-regularization, if present, is rougher than for α-Boost.

7.6.3 A Connection to Margin Maximization

There is another connection between ℓ_1-regularization and boosting, specifically, to margin maximization, which was studied as a core property of AdaBoost in chapter 5. In particular, if the regularization is relaxed to the limit so that $B \to \infty$ in the optimization problem given in equation (7.34) (or, equivalently, if $\beta \to 0$ in minimizing equation (7.35)), then the solution vectors $\boldsymbol{\lambda}_B^*$ turn out asymptotically to maximize the margins of the training examples. This suggests that regularization may generalize well on classification problems as a result of margin-maximizing properties not unlike those of AdaBoost. However, this holds true only as the regularization is weakened to the point of apparently disappearing altogether. In such a regime, good performance cannot be attributed to the kind of smoothing associated with regularization that was discussed above.

Figure 7.8
The trajectories of the weight vectors $\boldsymbol{\lambda}$ computed on the heart-disease dataset of section 1.2.3, using only the six base classifiers from table 1.2 (p. 13). Trajectories are plotted for ℓ_1-regularized exponential loss as in equation (7.34) (top); α-Boost with $\alpha = 10^{-6}$ (middle); and AdaBoost (bottom). Each figure includes one curve for each of the six base classifiers from table 1.2, showing its associated weight as a function of the total weight added.

Note also the subtle distinction between $B \to \infty$ (or equivalently, $\beta \to 0$) and $B = \infty$ ($\beta = 0$). In the former case, regularization yields a maximum margin solution; in the latter case, in which we are minimizing unregularized exponential loss, the solution need not have any such properties (depending on the algorithm used to effect the minimization), as seen, for instance, in the example of section 7.3.

To make this margin-maximizing property precise, let us first define $f_{\boldsymbol{\lambda}}$, for any $\boldsymbol{\lambda} \neq \mathbf{0}$, to be a version of $F_{\boldsymbol{\lambda}}$ (equation (7.6)) in which the weights have been normalized:

$$f_{\boldsymbol{\lambda}}(x) \doteq \frac{F_{\boldsymbol{\lambda}}(x)}{\|\boldsymbol{\lambda}\|_1} = \frac{\sum_{j=1}^{N} \lambda_j h_j(x)}{\|\boldsymbol{\lambda}\|_1}.$$

We assume that the data is linearly separable with positive margin $\theta > 0$, as defined in section 3.2. That is, we assume there exists a vector $\tilde{\boldsymbol{\lambda}}$ for which every training example has margin at least θ, so that

$$y_i f_{\tilde{\boldsymbol{\lambda}}}(x_i) \geq \theta \tag{7.37}$$

for all $i = 1, \ldots, m$. Without loss of generality, we assume $\|\tilde{\boldsymbol{\lambda}}\|_1 = 1$. To simplify notation, we also write F_B^* and f_B^* for $F_{\boldsymbol{\lambda}_B^*}$ and $f_{\boldsymbol{\lambda}_B^*}$, respectively.

We claim that as B is made large, the margins $y_i f_B^*(x_i)$ approach or exceed θ; specifically, we claim that

$$y_i f_B^*(x_i) \geq \theta - \frac{\ln m}{B}$$

for each of the m training examples i, provided $B > (\ln m)/\theta$. To see this, we start with the fact that $\boldsymbol{\lambda}_B^*$ minimizes the exponential loss L among all vectors of ℓ_1-norm at most B; thus, in particular,

$$L(\boldsymbol{\lambda}_B^*) \leq L(B\tilde{\boldsymbol{\lambda}}) \tag{7.38}$$

since $\|B\tilde{\boldsymbol{\lambda}}\|_1 = B$. We can bound the right-hand side using equation (7.37):

$$\begin{aligned} mL(B\tilde{\boldsymbol{\lambda}}) &= \sum_{i=1}^{m} \exp\left(-y_i F_{B\tilde{\boldsymbol{\lambda}}}(x_i)\right) \\ &= \sum_{i=1}^{m} \exp\left(-y_i B f_{\tilde{\boldsymbol{\lambda}}}(x_i)\right) \\ &\leq m e^{-B\theta}. \end{aligned}$$

Further, when $B > (\ln m)/\theta$, this is strictly less than 1. Combined with equation (7.38), this implies, for any i, that

$$\exp\left(-y_i F_B^*(x_i)\right) \leq mL(\boldsymbol{\lambda}_B^*) \leq me^{-B\theta} < 1$$

or, equivalently,

$$y_i \|\boldsymbol{\lambda}_B^*\|_1 f_B^*(x_i) = y_i F_B^*(x_i) \geq B\theta - \ln m > 0.$$

Thus, as claimed,

$$y_i f_B^*(x_i) \geq \frac{B\theta - \ln m}{\|\boldsymbol{\lambda}_B^*\|_1} \geq \theta - \frac{\ln m}{B}$$

since $\|\boldsymbol{\lambda}_B^*\|_1 \leq B$.

So very weak ℓ_1-regularization can be used to find a maximum margin classifier, as can other techniques discussed in section 5.4.2, as well as α-Boost with α very small, as seen in section 6.4.3. Indeed, the fact that both α-Boost and ℓ_1-regularization can be used for this purpose is entirely consistent with our earlier discussion linking the behavior of the two algorithms.

7.7 Applications to Data-Limited Learning

The tools developed in this chapter are quite general purpose, and can be used, through careful design of the loss function, for a range of learning problems. In this last section, we give two examples of how these methods can be applied practically, in both cases to problems that arise as a result of limitations in the availability of data.

7.7.1 Incorporating Prior Knowledge

Throughout this book, we have focused on highly data-driven learning algorithms which derive a hypothesis exclusively through the examination of the training set itself. This approach makes sense when data is abundant. However, in some applications, data may be severely limited. Nevertheless, there may be accessible human knowledge that, in principle, might compensate for the shortage of data.

For instance, in the development of a spoken-dialogue system of the sort described in section 10.3 and more briefly in section 5.7, training a classifier for categorizing spoken utterances requires considerable data. This is a problem, however, because often such a system must be deployed *before* enough data has been collected. Indeed, real data in the form of actual conversations with genuine customers cannot be easily collected until *after* the system has been deployed. The idea then is to use human-crafted knowledge to compensate for this initial dearth of data until enough can be collected following deployment.

In its standard form, boosting does not allow for the direct incorporation of such prior knowledge (other than implicit knowledge encoded in the choice of weak-learning algorithm). Here, we describe a modification of boosting that combines and balances human

expertise with available training data. The aim is to allow the human's rough judgments to be refined, reinforced, and adjusted by the statistics of the training data, but in a manner that does not permit the data to entirely overwhelm human judgments.

The basic idea is to modify the loss function used by boosting so that the algorithm balances two terms, one measuring fit to the training data and the other measuring fit to a human-built model.

As usual, we assume we are given m training examples $(x_1, y_1), \ldots, (x_m, y_m)$ with $y_i \in \{-1, +1\}$. Now, however, we suppose that the number of examples m is rather limited. Our starting point is AdaBoost.L (algorithm 7.4), the boosting-style algorithm for logistic regression presented in section 7.5. We saw earlier that this algorithm is based on the loss function given in equation (7.21), which is the negative log conditional likelihood of the data when the conditional probability that an example x is positive is estimated by $\sigma(F(x))$, where σ is as defined in equation (7.19).

Into this data-driven approach we wish to incorporate prior knowledge. There are of course many forms and representations that "knowledge" can take. Here, we assume that a human "expert" has somehow constructed a function $\tilde{p}(x)$ that estimates, perhaps quite crudely, the conditional probability that any example x will be positive, that is, $\mathbf{Pr}_{\mathcal{D}}[y = +1 \mid x]$.

Given both a prior model and training data, we now have two possibly conflicting goals in constructing a hypothesis: (1) fit the data, and (2) fit the prior model. As before, we can measure fit to the data using log conditional likelihood as in equation (7.21). But how do we measure fit to the prior model? As just discussed, the learning algorithm seeks a model of the form $\sigma(F(x))$ which estimates the conditional distribution of labels for any example x. This is also the same conditional distribution being estimated by the prior model $\tilde{p}$. Thus, to measure the difference between the models, we can use the discrepancy between these conditional distributions. And since distributions are involved, it is natural to measure this discrepancy using relative entropy.

Thus, to measure fit to the prior model, for each example x_i we use the relative entropy between the prior model distribution given by $\tilde{p}(x_i)$ and the distribution over labels associated with our constructed logistic model $\sigma(F(x_i))$. These are then added over all of the training examples so that the overall fit of the constructed hypothesis to the prior model is computed by

$$\sum_{i=1}^{m} \mathrm{RE}_b\left(\tilde{p}(x_i) \;\|\; \sigma(F(x_i))\right) \tag{7.39}$$

where $\mathrm{RE}_b(\cdot \;\|\; \cdot)$ is the binary relative entropy defined in equation (5.36).

So our goal now is to minimize equations (7.21) and (7.39). These can be combined, with the introduction of a parameter $\eta > 0$ measuring their relative importance, to arrive at the modified loss function

$$\sum_{i=1}^{m} \left[\ln\left(1+\exp(-y_i F(x_i))\right) + \eta \mathrm{RE}_b\left(\tilde{p}(x_i) \parallel \sigma(F(x_i))\right)\right].$$

This expression can be rewritten as

$$C + \sum_{i=1}^{m} \left[\ln\left(1+e^{-y_i F(x_i)}\right) + \eta\tilde{p}(x_i)\ln\left(1+e^{-F(x_i)}\right) + \eta(1-\tilde{p}(x_i))\ln\left(1+e^{F(x_i)}\right)\right] \tag{7.40}$$

where C is a term that is independent of F, and so can be disregarded.

Note that this objective function has the same form as equation (7.21) over a larger set, and with the addition of nonnegative weights on each term. Therefore, to minimize equation (7.40), we apply the AdaBoost.L procedure to a larger weighted training set. This new set includes all of the original training examples (x_i, y_i), each with unit weight. In addition, for each training example (x_i, y_i), we create two new training examples $(x_i, +1)$ and $(x_i, -1)$ with weights $\eta\tilde{p}(x_i)$ and $\eta(1-\tilde{p}(x_i))$, respectively. Thus, we triple the number of examples (although, by noticing that (x_i, y_i) occurs twice, we can get away with only doubling the training set). AdaBoost.L can be easily modified to incorporate these weights w_0 in the computation of D_t in algorithm 7.4, using instead

$$D_t(i) \propto \frac{w_0(i)}{1+\exp\left(y_i F_{t-1}(x_i)\right)}$$

(where i ranges over all of the examples in the *new* training set).

One final modification that we make is to add a 0-th base function h_0 that is based on $\tilde{p}$, so as to incorporate $\tilde{p}$ from the start. In particular, we take

$$h_0(x) = \sigma^{-1}(\tilde{p}(x)) = \ln\left(\frac{\tilde{p}(x)}{1-\tilde{p}(x)}\right)$$

and include h_0 in computing the final classifier F.

As a concrete example, this technique can be applied to classify the headlines of news articles from the Associated Press by topic. The dataset used here consists of 29,841 examples (of which only a subset were used for training) over 20 topics or classes. A multiclass extension of the approach above was used based on the kind of techniques developed in chapter 10.

Our framework permits prior knowledge of any kind, so long as it provides estimates, however rough, of the probability of any example belonging to any class. Here is one possible technique for creating such a rough model that was tested experimentally. First, a human with no expertise beyond ordinary knowledge of news events, and with access

Table 7.2
The keywords used for each class on the AP headlines dataset

Class	Keywords
japan	japan, tokyo, yen
bush	bush, george, president, election
israel	israel, jerusalem, peres, sharon, palestinian, israeli, arafat
britx	britain, british, england, english, london, thatcher
gulf	gulf, iraq, saudi, arab, iraqi, saddam, hussein, kuwait
german	german, germany, bonn, berlin, mark
weather	weather, rain, snow, cold, ice, sun, sunny, cloudy
dollargold	dollar, gold, price
hostages	hostages, ransom, holding, hostage
budget	budget, deficit, taxes
arts	art, painting, artist, music, entertainment, museum, theater
dukakis	dukakis, boston, taxes, governor
yugoslavia	yugoslavia
quayle	quayle, dan
ireland	ireland, ira, dublin
burma	burma
bonds	bond, bonds, yield, interest
nielsens	nielsens, rating, t v, tv
boxoffice	box office, movie
tickertalk	stock, bond, bonds, stocks, price, earnings

to the list of categories but not to the data itself, thought up a handful of keywords for each class. This list of keywords is shown in table 7.2. These keywords were produced through an entirely subjective process of free association with general knowledge of what the categories were about (and also the time period during which the data was collected), but no other information or access to the data.

Such a list of keywords can next be used to construct a simple and naive, but very rough, prior model. To do so, we posit that if a keyword w is present, then there is a 90% probability that the correct class is among those that list w as a keyword, and a 10% chance that it is one of the others. For instance, conditional on the presence of the keyword *price* in the headline, the naive prior model estimates that there would be a 90% probability of the correct class being "dollargold" or "tickertalk" (that is, a 45% chance for each one separately), and a 10% chance of one of the other 18 topics being the correct class (giving $10\%/18 \approx 0.6\%$ probability to each). If the keyword is absent, we posit that all classes are equally likely. These conditional probabilities can then be combined using very naive independence assumptions, together with simple probabilistic reasoning.

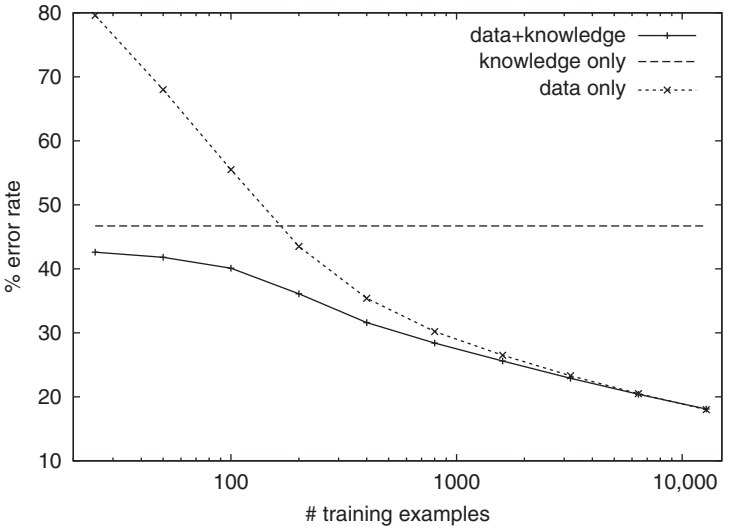


Figure 7.9
Comparison of test error rate using prior knowledge and data separately or together on the AP headlines dataset, measured as a function of the number of training examples. Results are averaged over ten runs, each with a different random partition of the data into training and testing.

Figure 7.9 shows the performance of the method above for training sets of varying sizes, in each case using boosting with decision stumps for 1000 rounds. The figure shows test error rate for boosting with and without prior knowledge, as well as the error rate achieved using the prior model alone with no training examples at all. As can be seen, for fairly small datasets, using prior knowledge gives dramatic improvements over straight boosting, almost halving the error early on, and providing performance equivalent to using two to four times as much data without such knowledge. With a lot of data, as expected, the effect of the prior model is washed out.

7.7.2 Semi-Supervised Learning

In section 5.7.2, we discussed the ubiquitous problem of learning from an abundance of unlabeled examples but a severely limited supply of labeled examples. There, we considered how to make best use of a human annotator through the careful selection of examples for labeling. Whether or not we have the means to obtain additional labels, there sometimes may exist an opportunity to improve classifier accuracy by directly exploiting the unlabeled data, which otherwise is apparently wasted. Intuitively, unlabeled data may provide distributional information that may be helpful for classification (an informal example is given shortly). This problem is often called *semi-supervised learning*. Here, we present one boosting approach based on loss minimization.

In the semi-supervised framework, data is presented in two parts: a set of labeled examples $(x_1, y_1), \ldots, (x_m, y_m)$, together with another, usually much larger, set of unlabeled examples $\tilde{x}_1, \ldots, \tilde{x}_M$. Our approach to handling such mixed data will be to construct an appropriate loss function which can then be optimized using techniques from this chapter. For the labeled examples, we can start with standard AdaBoost, which, as discussed in section 7.1, is based on exponential loss over the labeled examples as in equation (7.3). This loss encourages the construction of a hypothesis F whose sign on labeled examples x_i agrees with their observed labels y_i.

For the unlabeled examples, however, we obviously have no true labels available to match. Nevertheless, in chapter 5 we studied in depth the importance of producing predictions that are highly confident in the sense of having large margins. Moreover, observing the label is not prerequisite to computing such a measure of confidence. This motivates the idea of encouraging F to be large in magnitude, and thus more confident, on the unlabeled examples $\tilde{x}_i$ as well. To do so, we can use a variant of exponential loss, namely,

$$\frac{1}{M}\sum_{i=1}^{M} e^{-|F(\tilde{x}_i)|}, \tag{7.41}$$

an objective whose minimization will have the effect of enlarging the unnormalized absolute margins $|F(\tilde{x}_i)|$. Of course, the theory developed in chapter 5 emphasized the importance of the *normalized* margin (in which the weights of the base hypotheses have been normalized) rather than the unnormalized margin used here, but we also saw that AdaBoost, in minimizing exponential loss, has a tendency to (approximately) maximize the normalized margins as well.

Viewed slightly differently, with respect to a hypothesis F, if we define pseudolabels $\tilde{y}_i = \text{sign}(F(\tilde{x}_i))$, then equation (7.41) can be rewritten as

$$\frac{1}{M}\sum_{i=1}^{M} e^{-\tilde{y}_i F(\tilde{x}_i)}, \tag{7.42}$$

in other words, in the same form as equation (7.3) for labeled pairs $(\tilde{x}_i, \tilde{y}_i)$. Since these are exactly the choices of labels for which equation (7.42) is minimized, this means that the loss in equation (7.41) will tend to be small for those hypotheses F which fit the unlabeled data well for *some* labeling, even if it is not the true, hidden labeling.

For instance, for intuition's sake, we can regard the unlabeled examples geometrically as points in feature space, similar to the setup of section 5.6. Figure 7.10 shows such a set of points. The combined classifier defines a hyperplane in this space defined by the equation $F(x) = 0$. In this example, the line (hyperplane) F_1, which admits a large-margin classification of the data, will clearly be favored by the objective in equation (7.41) over

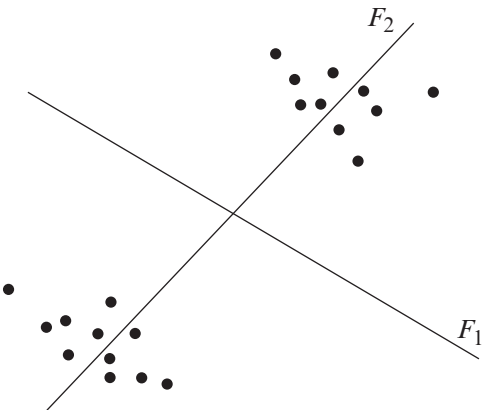


Figure 7.10
A hypothetical set of unlabeled examples in feature space. The linear classifier marked F_1 admits significantly higher margins for some labeling of the points than F_2, and thus will be preferred when using the loss defined by equation (7.41).

F_2, which passes too close to too many examples for this to be possible. This is consistent with our intuition that F_1 divides the data, even if unlabeled, more naturally than F_2.

Combining the loss functions in equations (7.3) and (7.41) yields

$$\frac{1}{m}\sum_{i=1}^{m} e^{-y_i F(x_i)} + \frac{\eta}{M}\sum_{i=1}^{M} e^{-|F(\tilde{x}_i)|}, \tag{7.43}$$

where we have introduced a parameter $\eta > 0$ controlling the relative importance given to the unlabeled examples.

To minimize equation (7.43), we can apply the functional gradient descent approach of section 7.4, specifically, AnyBoost (algorithm 7.3). Using the pseudolabels $\tilde{y}_i$ as in equation (7.42), and differentiating the loss in equation (7.43) to evaluate equation (7.14), this approach prescribes choosing h_t to maximize

$$\frac{1}{m}\sum_{i=1}^{m} y_i h_t(x_i) e^{-y_i F_{t-1}(x_i)} + \frac{\eta}{M}\sum_{i=1}^{M} \tilde{y}_i h_t(x_i) e^{-\tilde{y}_i F_{t-1}(\tilde{x}_i)}.$$

Of course, this has the same form as equation (7.15), meaning that any ordinary base learning algorithm designed for minimization of the weighted classification error on a labeled dataset can be used for this purpose.

Once h_t has been chosen, we need to select $\alpha_t > 0$. To simplify notation, let us for the moment drop subscripts involving t. A line search would choose α to minimize the resulting loss

$$\frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i(F(x_i)+\alpha h(x_i))\right)+\frac{\eta}{M}\sum_{i=1}^{M}\exp\left(-|F(\tilde{x}_i)+\alpha h(\tilde{x}_i)|\right). \tag{7.44}$$

This could be complicated to compute because of the absolute values in the exponents. Instead, a simpler approach would continue to use the pseudolabels so that α is chosen to minimize

$$\frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i(F(x_i)+\alpha h(x_i))\right)+\frac{\eta}{M}\sum_{i=1}^{M}\exp\left(-\tilde{y}_i(F(\tilde{x}_i)+\alpha h(\tilde{x}_i))\right). \tag{7.45}$$

In other words, we can use the usual AdaBoost choice of α based on the weighted error on the entire dataset as augmented by the pseudolabels. Note that equation (7.44) is at most equation (7.45), as follows from the simple fact that

$$\tilde{y}_i(F(\tilde{x}_i)+\alpha h(\tilde{x}_i)) \le |\tilde{y}_i(F(\tilde{x}_i)+\alpha h(\tilde{x}_i))| = |F(\tilde{x}_i)+\alpha h(\tilde{x}_i)|\,.$$

Therefore, although perhaps not best possible, choosing α in this manner so as to minimize equation (7.45) is guaranteed to effect at least as great a decrease in equation (7.44).

Once h_t and α_t have been selected, F_t is computed as in algorithm 7.3, the pseudolabels are then recomputed with respect to F_t, and the process continues.

This algorithm, called *ASSEMBLE.AdaBoost*, was entered into a competition of semi-supervised learning algorithms in 2001, and took first place among 34 entries. In this case, decision trees were used as base classifiers. Also, the pseudolabel for each unlabeled example $\tilde{x}_i$ was initialized to be the label of its nearest neighbor in the labeled set.

Table 7.3 shows the results on the datasets used in this competition, comparing ASSEMBLE.AdaBoost's performance with that of AdaBoost trained just on the labeled

Table 7.3
Test results for ASSEMBLE.AdaBoost on seven datasets used in a 2001 competition of semi-supervised learning algorithms

Dataset	# of Features	# of Classes	# Examples			% Error	% Improve
			Labeled	Unlabeled	Test		
P1	13	2	36	92	100	35.0	8.3
P4	192	9	108	108	216	21.3	21.4
P5	1000	2	50	3952	100	24.0	16.9
P6	12	2	530	2120	2710	24.3	0.1
P8	296	4	537	608	211	42.2	16.2
CP2	21	3	300	820	300	49.3	26.7
CP3	10	5	1000	1269	500	58.8	6.7

The last two columns show the test error of ASSEMBLE.AdaBoost, followed by the percent improvement over running AdaBoost just on the labeled data.

examples. In most cases, the improvement in accuracy was very substantial, often in double digits.

Summary

In summary, we have seen that AdaBoost is closely associated with the exponential loss function. Although this connection may not explain how AdaBoost achieves high test accuracy, it does lead to some widely applicable techniques for extending and generalizing it. Specifically, we have explored how, in two different ways, AdaBoost can be viewed as a special case of more general procedures for optimization of an objective function. Also in this chapter, we have explored the close connections between AdaBoost and logistic regression. We have seen how this leads both to a simple modification of AdaBoost for logistic regression and to a method for using AdaBoost to estimate conditional label probabilities. We also explored how regularization can be used as a smoothing technique for avoiding overfitting, as well as the close connections between boosting and regularization using an ℓ_1-norm. Finally, we saw examples of how these general techniques can be applied practically.

Bibliographic Notes

Breiman [36] was the first to interpret AdaBoost as a method for minimizing exponential loss, as in section 7.1, a connection that was later expanded upon by various other authors [87, 98, 168, 178, 186, 205]. The view of boosting as a form of gradient descent, as in section 7.4, is also due to Breiman [36]. The generic AnyBoost procedure, algorithm 7.3, is taken from Mason et al. [168]. A similar generalization was given by Friedman [100], leading to a general procedure based on solving multiple regression problems as in section 7.4.3. When applied to square loss, the resulting procedures are essentially the same as Mallat and Zhang's work on matching pursuit [163].

Friedman, Hastie, and Tibshirani [98] showed how boosting can be understood in the context of a family of statistical techniques called additive modeling. They also provided justification for the use of exponential loss, particularly in terms of its close relationship to logistic loss as discussed somewhat in section 7.5.1. They showed how a function obtained through the minimization of exponential loss can be used to estimate conditional probabilities, as in section 7.5.3. Although they derived a version of AdaBoost for logistic regression called LogitBoost, the AdaBoost.L algorithm presented in section 7.5.2 came in the later work of Collins, Schapire, and Singer [54].

The discussion and experiment in section 7.3 are similar in spirit to work by Mease and Wyner [169] that attempted to expose some of the difficulties with the so-called "statistical view" of boosting, including its interpretation as a method for optimizing a loss as

expounded in this chapter. Wyner [232] also gives a variant of AdaBoost that performs well while keeping the exponential loss roughly constant, adding further doubts to this interpretation.

Regularization using an ℓ_1-norm on the weights, as in section 7.6.1, is often called "the lasso," and was proposed in the context of least-squares regression by Tibshirani [218]. The connection to α-Boost with early stopping, as discussed in section 7.6.2, was observed by Hastie, Tibshirani, and Friedman [120], and explored further by Rosset, Zhu, and Hastie [192], who also showed that weak ℓ_1-regularization yields a maximum margin solution, as seen in section 7.6.3. A more advanced and general boosting-like algorithm for tracing the entire ℓ_1-regularized trajectory is given by Zhao and Yu [237].

In addition to those given here, other boosting-like algorithms and methods for regression have been proposed, for instance, by Freund and Schapire [95], Ridgeway, Madigan and Richardson [190], and Duffy and Helmbold [79]. In other work, Duffy and Helmbold [78] discussed conditions under which minimization of a loss function using functional gradient descent can lead to a boosting algorithm in the technical PAC sense of section 2.3.

Further background on many of the statistical topics discussed in this chapter, including regression, logistic regression, and regularization, can be found, for instance, in the text by Hastie, Tibshirani, and Friedman [120]. For additional background on general-purpose optimization methods, including coordinate descent and Gauss-Southwell methods, see, for instance, Luenberger and Ye [160].

The NP-completeness of finding a linear threshold function with minimum classification loss, alluded to in section 7.1, was proved by Höffgen and Simon [124].

The method and experiments in section 7.7.1, including table 7.2 and figure 7.9, are taken from Schapire et al. [203, 204], using a dataset of AP headlines prepared by Lewis and Catlett [151] and Lewis and Gale [152]. The ASSEMBLE.AdaBoost algorithm described in section 7.7.2, including the results adapted in table 7.3, are due to Bennett, Demiriz, and Maclin [18].

Some of the exercises in this chapter are based on material from [98, 233].

Exercises

7.1 Consider the following objective function:

$$L(x, y) \doteq \max\{x - 2y, y - 2x\}.$$

a. Sketch a *contour map* of L. That is, in the $\langle x, y\rangle$-plane, plot the *level set* $\{\langle x, y\rangle : L(x, y) = c\}$ for various values of c.

b. Is L continuous? Convex? Does the gradient of L exist at all values of x and y?

c. What is the minimum (or infimum) of L over all $\langle x, y\rangle$? Where is that minimum value realized?

d. Explain what will happen if we apply coordinate descent to L, starting at any initial point $\langle x, y \rangle$. Is coordinate descent successful in this case (that is, at minimizing the objective function)?

7.2 Continuing exercise 7.1, suppose we approximate L by

$$\tilde{L}(x, y) \doteq \frac{1}{\eta} \ln \left(e^{\eta(x-2y)} + e^{\eta(y-2x)} \right)$$

for some large constant $\eta > 0$.

a. Show that

$$\left| L(x, y) - \tilde{L}(x, y) \right| \leq \frac{\ln 2}{\eta}$$

for all $\langle x, y \rangle$.

b. How would a contour plot of $\tilde{L}$ compare with that of L?

c. Explain what will happen if we apply coordinate descent to $\tilde{L}$, starting at $\langle 0, 0 \rangle$. Give the exact sequence of points $\langle x_t, y_t \rangle$ that will be computed, as well as the resulting loss $\tilde{L}(x_t, y_t)$. Is coordinate descent successful in this case?

7.3 Suppose, as in section 7.3, that instances $\mathbf{x}$ are points in $\mathcal{X} = \{-1, +1\}^N$; that we are using coordinates (and their negations) as base classifiers; and that the label y associated with each instance $\mathbf{x}$ is a simple majority vote of k of the coordinates:

$$y = \text{sign} \left(x_{j_1} + \cdots + x_{j_k} \right)$$

for some (unknown, not necessarily distinct) coordinates $j_1, \ldots, j_k \in \{1, \ldots, N\}$ (where k is odd). Let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ be a random training set where the $\mathbf{x}_i$'s are generated according to an arbitrary target distribution $\mathcal{D}$ on $\mathcal{X}$ (and the labels y_i are assigned as just described).

Show that if AdaBoost is run with an exhaustive weak learner for sufficiently many rounds T, then with probability at least $1 - \delta$, the generalization error of the combined classifier H will be at most

$$O\left(\sqrt{\frac{k^2 \ln N}{m} \cdot \ln \left(\frac{m}{k^2 \ln N} \right) + \frac{\ln(1/\delta)}{m}} \right).$$

(An even better bound is also possible.) Thus, m only needs to grow faster than $k^2 \ln N$ for the generalization error to go to zero.

7.4 Continuing exercise 7.3, the exponential loss of equation (7.7), specialized to this setting, has the form

$$L(\boldsymbol{\lambda}) = \frac{1}{m}\sum_{i=1}^{m} \exp(-y_i \boldsymbol{\lambda} \cdot \mathbf{x}_i).$$

a. Compute $\nabla L(\boldsymbol{\lambda})$, the gradient of L at $\boldsymbol{\lambda}$.

b. When using gradient descent, regardless of the learning rate, show that the computed weight vector $\boldsymbol{\lambda}$ will always be in the span of the training instances, meaning

$$\boldsymbol{\lambda} = \sum_{i=1}^{m} b_i \mathbf{x}_i$$

for some $b_1, \ldots, b_m \in \mathbb{R}$. (Throughout this problem, assume gradient descent is started at $\mathbf{0}$.)

c. An $N \times N$ *Hadamard matrix* has all entries in $\{-1, +1\}$, and all columns are orthogonal to one another. Hadamard matrices $\mathbf{H}_N$ in which N is a power of 2 can be constructed recursively with $\mathbf{H}_1 = (1)$, and

$$\mathbf{H}_{2N} = \begin{pmatrix} \mathbf{H}_N & \mathbf{H}_N \\ \mathbf{H}_N & -\mathbf{H}_N \end{pmatrix}.$$

Verify that matrices constructed in this fashion are indeed Hadamard.

d. For N a power of 2, suppose the target distribution $\mathcal{D}$ on instances is uniform on the columns of the Hadamard matrix $\mathbf{H}_N$. Show that regardless of how instances are labeled, the generalization error of a classifier $H(x) = \text{sign}(\boldsymbol{\lambda} \cdot \mathbf{x})$ computed using gradient descent is at least $1/2 - m/(2N)$. (For the purposes of this exercise, let us count a prediction of 0 as half a mistake.) Thus, when the number of training examples m is small compared to the number of dimensions N, the generalization error is sure to be large.

7.5 For given real values $b_1, \ldots, b_m$, $c_1, \ldots, c_m$, and nonnegative weights $w_1, \ldots, w_m$, let

$$f(\alpha) = \sum_{i=1}^{m} w_i (\alpha b_i - c_i)^2.$$

Find explicitly the value of α that minimizes f, and also evaluate f at this value.

7.6 Let $\mathcal{H}$ be a class of real-valued base functions that is closed under scaling (so that if $h \in \mathcal{H}$, then ch is also in $\mathcal{H}$ for all $c \in \mathbb{R}$). Show that choosing $h \in \mathcal{H}$ to minimize the squared error criterion in equation (7.17) is equivalent to using AnyBoost (algorithm 7.3) with appropriately normalized base functions from $\mathcal{H}$. Specifically, show that equation (7.17) is minimized (over $\mathcal{H}$) by a function $ch \in \mathcal{H}$ where $c \in \mathbb{R}$, and where h maximizes $-\nabla \mathcal{L}(F_{t-1}) \cdot h$ over all functions in $\mathcal{H}$ with $\|h\|_2 = 1$. (Assume this maximum exists.)

7.7 *GentleAdaBoost* is an algorithm for minimizing exponential loss derived using a form of a numerical technique called Newton's method. The algorithm turns out to be the same as AdaBoost as depicted in algorithm 7.1, except that h_t and α_t are instead found by minimizing the weighted least-squares problem:

$$\sum_{i=1}^{m} e^{-y_i F_{t-1}(x_i)} (\alpha_t h_t(x_i) - y_i)^2. \tag{7.46}$$

Assume all of the base functions in the space $\mathcal{H}$ are classifiers, that is, $\{-1, +1\}$-valued.

a. Under equivalent conditions (that is, the same dataset, same current value of F_{t-1}, etc.), show that GentleAdaBoost selects the same base classifier h_t as AdaBoost (in other words, h_t, together with some value of α_t, minimizes equation (7.46) if and only if it minimizes the corresponding criterion given in algorithm 7.1).

b. Under equivalent conditions (including the same choice of h_t), let α_t^{GB} and α_t^{AB} denote the values of α_t selected by GentleAdaBoost and AdaBoost, respectively. Show that

i. α_t^{GB} and α_t^{AB} must have the same sign.

ii. $\alpha_t^{\text{GB}} \in [-1, +1]$.

iii. $|\alpha_t^{\text{GB}}| \leq |\alpha_t^{\text{AB}}|$.

7.8 *LogitBoost* (algorithm 7.5) is an algorithm also derived using Newton's method, but applied to the logistic loss. As in exercise 7.7, assume all of the base functions in $\mathcal{H}$ are classifiers.

a. Under equivalent conditions, show that LogitBoost selects the same base classifier as AdaBoost.L. (Assume the algorithms select α_t and h_t using an exhaustive search.)

b. Under equivalent conditions (including the choice of h_t), let α_t^{LB} and α_t^{ABL} denote the values of α_t selected by LogitBoost and AdaBoost.L, respectively. Give explicit expressions for α_t^{LB} and α_t^{ABL} in terms of the data, h_t, D_t, and $\mathcal{Z}_t$, where D_t and $\mathcal{Z}_t$ are as defined in algorithm 7.4.

c. Show that α_t^{LB} and α_t^{ABL} must have the same sign.

d. Prove by example that $|\alpha_t^{\text{LB}}|$ can be strictly smaller or strictly larger than $|\alpha_t^{\text{ABL}}|$.

7.9 Suppose $\mathcal{H}$ consists of a single function $\hbar$ that is identically equal to the constant $+1$. Let the training set consist of $m = 2$ oppositely labeled examples $(x_1, +1)$ and $(x_2, -1)$.

a. For what value of the single parameter λ is the logistic loss in equation (7.20) minimized (where $F_\lambda(x) \doteq \lambda \hbar(x) \equiv \lambda$ for all x)?

b. Suppose LogitBoost is run on this data, but with F_0 initialized to the constant function $F_0 = \lambda_0 \hbar \equiv \lambda_0$, for some $\lambda_0 \in \mathbb{R}$ which is not necessarily zero. Since $h_t = \hbar$ on every round, we can write F_t as $\lambda_t \hbar \equiv \lambda_t$ for some $\lambda_t \in \mathbb{R}$. Write a formula for λ_{t+1} in terms of λ_t. That is, find in closed form the function $U(\lambda)$ which defines the update $\lambda_{t+1} = U(\lambda_t)$.

Algorithm 7.5
LogitBoost, an algorithm based on Newton's method for minimizing logistic loss

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $F_0 \equiv 0$.
For $t = 1, \ldots, T$:

- For $i = 1, \ldots, m$:

$$p_t(i) = \frac{1}{1 + e^{-F_{t-1}(x_i)}}$$

$$z_t(i) = \begin{cases} \frac{1}{p_t(i)} & \text{if } y_i = +1 \\ -\frac{1}{1-p_t(i)} & \text{if } y_i = -1 \end{cases}$$

$$w_t(i) = p_t(i)(1 - p_t(i)).$$

- Choose $\alpha_t \in \mathbb{R}$ and $h_t \in \mathcal{H}$ to minimize (or approximately minimize if a heuristic search is used)

$$\sum_{i=1}^{m} w_t(i)(\alpha_t h_t(x_i) - z_t(i))^2.$$

- Update: $F_t = F_{t-1} + \alpha_t h_t$.

Output F_T.

c. Show that there exists a constant $C > 0$ such that if $|\lambda_0| \leq C$, then $\lambda_t \to 0$ as $t \to \infty$.

d. Show that if $|\lambda_0| \geq 3$, then $|\lambda_t| \to \infty$ as $t \to \infty$.

e. How would AdaBoost.L handle this same setting (with the same initialization of F_0)?

7.10 Let $L : \mathbb{R}^N \to \mathbb{R}$ be any convex function. Suppose $\boldsymbol{\lambda}$ is a solution of the constrained optimization problem given in equation (7.34) for some $B > 0$. Show that if $\|\boldsymbol{\lambda}\|_1 < B$, then $\boldsymbol{\lambda}$ is also a solution of equation (7.34) when B is replaced by any $B' > B$.

7.11 Assume $\mathcal{H}$ consists only of classifiers (that is, base functions with range $\{-1, +1\}$), and that the data is linearly separable with margin $\theta > 0$ (as in equation (7.37)). For any $B > 0$, let $\boldsymbol{\lambda}$ be a solution of equation (7.34) for the loss $L(\boldsymbol{\lambda})$ as defined in equation (7.7). Assume also that $\mathcal{H}$ is closed under negation so that $\lambda_j \geq 0$ for all j, without loss of generality.

Let us define the distribution D as $D(i) = \exp(-y_i F_{\boldsymbol{\lambda}}(x_i))/\mathcal{Z}$ where $\mathcal{Z}$ is a normalization factor, and $F_{\boldsymbol{\lambda}}$ is as in equation (7.6). For each $h_j \in \mathcal{H}$, we also define

$$e_j \doteq \sum_{i=1}^{m} D(i) y_i \hbar_j(x_i),$$

which is (twice) the edge of $\hbar_j$ with respect to D.

a. Show how to express $\partial L / \partial \lambda_j$ explicitly in terms of e_j, and possibly other quantities that are independent of j.

b. Show that for some j, $\partial L / \partial \lambda_j$ is strictly negative. Use this to show that $\|\boldsymbol{\lambda}\|_1 = B$. [*Hint:* If this is not the case, show how to modify $\boldsymbol{\lambda}$ to get strictly lower loss.]

c. We say $\hbar_j$ is *active* if $\lambda_j > 0$. Show that if $\hbar_j$ is active, then it has maximum edge, that is, $e_j \geq e_{j'}$ for all j'.

d. Show that the edges of all active base classifiers are equal, that is, if $\lambda_j > 0$ and $\lambda_{j'} > 0$, then $e_j = e_{j'}$. Furthermore, show that if $\hbar_j$ is active, then $e_j \geq \theta$.

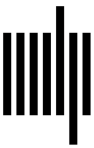
Boosting

Foundations and Algorithms

By: Robert E. Schapire, Yoav Freund

Citation:

Boosting: Foundations and Algorithms
By: Robert E. Schapire, Yoav Freund
DOI: 10.7551/mitpress/8291.001.0001
ISBN (electronic): 9780262301183
Publisher: The MIT Press
Published: 2014



The MIT Press

For information about special quality discounts, please email special_sales@mitpress.mit.edu

This book was set in Times Roman by Westchester Book Composition.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Schapire, Robert E.
Boosting : foundations and algorithms / Robert E. Schapire and Yoav Freund.
p. cm.—(Adaptive computation and machine learning series)
Includes bibliographical references and index.
ISBN 978-0-262-01718-3 (hardcover : alk. paper)
1. Boosting (Algorithms) 2. Supervised learning (Machine learning) I. Freund, Yoav. II. Title.
Q325.75.S33 2012
006.3'1—dc23
2011038972

10 9 8 7 6 5 4 3 2