The Boosting Approach to Machine Learning: An Overview

Robert E. Schapire

Summary

Boosting is a general method for improving the accuracy of any given learning algorithm. Focusing primarily on the AdaBoost algorithm, this chapter overviews some of the recent work on boosting including analyses of AdaBoost's training error and generalization error; boosting's connection to game theory and linear programming; the relationship between boosting and logistic regression; extensions of AdaBoost for multiclass classification problems; methods of incorporating human knowledge into boosting; and experimental and applied work using boosting.

8.1 Introduction

Machine learning studies automatic techniques for learning to make accurate predictions based on past observations. For example, suppose that we would like to build an email filter that can distinguish spam (junk) email from non-spam. The machine-learning approach to this problem would be the following: Start by gathering as many examples as possible of both spam and non-spam emails. Next, feed these examples, together with labels indicating if they are spam or not, to your favorite machine-learning algorithm which will automatically produce a classification or prediction rule. Given a new, unlabeled email, such a rule attempts to predict if it is spam or not. The goal, of course, is to generate a rule that makes the most accurate predictions possible on new test examples.

1Robert E. Schapire is with AT&T Labs – Research, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932, USA (URL: www.research.att.com/~schapire).
Building a highly accurate prediction rule is certainly a difficult task. On the other hand, it is not hard at all to come up with very rough rules of thumb that are only moderately accurate. An example of such a rule is something like the following: “If the phrase ‘buy now’ occurs in the email, then predict it is spam.” Such a rule will not even come close to covering all spam messages; for instance, it really says nothing about what to predict if ‘buy now’ does not occur in the message. On the other hand, this rule will make predictions that are significantly better than random guessing.

Boosting, the machine-learning method that is the subject of this chapter, is based on the observation that finding many rough rules of thumb can be a lot easier than finding a single, highly accurate prediction rule. To apply the boosting approach, we start with a method or algorithm for finding the rough rules of thumb. The boosting algorithm calls this “weak” or “base” learning algorithm repeatedly, each time feeding it a different subset of the training examples (or, to be more precise, a different distribution or weighting over the training examples\(^2\)). Each time it is called, the base learning algorithm generates a new weak prediction rule, and after many rounds, the boosting algorithm must combine these weak rules into a single prediction rule that, hopefully, will be much more accurate than any one of the weak rules.

To make this approach work, there are two fundamental questions that must be answered: first, how should each distribution be chosen on each round, and second, how should the weak rules be combined into a single rule? Regarding the choice of distribution, the technique that we advocate is to place the most weight on the examples most often misclassified by the preceding weak rules; this has the effect of forcing the base learner to focus its attention on the “hardest” examples. As for combining the weak rules, simply taking a (weighted) majority vote of their predictions is natural and effective.

There is also the question of what to use for the base learning algorithm, but this question we purposely leave unanswered so that we will end up with a general boosting procedure that can be combined with any base learning algorithm.

*Boosting* refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb in a manner similar to that suggested above. This chapter presents an overview of some of the recent work on boosting, focusing especially on the AdaBoost algorithm which has undergone intense theoretical study and empirical testing.

\(^2\)A distribution over training examples can be used to generate a subset of the training examples simply by sampling repeatedly from the distribution.
8. The Boosting Approach to Machine Learning: An Overview

Table 8.1. The boosting algorithm AdaBoost.

| Given: (x₁, y₁), ..., (xₘ, yₘ) where xᵢ ∈ X, yᵢ ∈ Y = {-1, +1} |
| Initialize D₁(i) = 1/m. |
| For t = 1, ..., T: |
| • Train base learner using distribution Dₜ. |
| • Get base classifier hₜ : X → ℝ. |
| • Choose αₜ ∈ ℝ. |
| • Update: |
|   Dₜ₊₁(i) = Dₜ(i) \exp(-αₜyᵢhₜ(xᵢ)) / Zₜ |
|   where Zₜ is a normalization factor (chosen so that Dₜ₊₁ will be a distribution). |
| Output the final classifier: |
|   H(x) = \text{sign} \left( \sum_{t=1}^{T} αₜhₜ(x) \right). |

8.2 AdaBoost

Working in Valiant's PAC (probably approximately correct) learning model [75], Kearns and Valiant [41, 42] were the first to pose the question of whether a "weak" learning algorithm that performs just slightly better than random guessing can be "boosted" into an arbitrarily accurate "strong" learning algorithm. Schapire [66] came up with the first provable polynomial-time boosting algorithm in 1989. A year later, Freund [26] developed a much more efficient boosting algorithm which, although optimal in a certain sense, nevertheless suffered like Schapire's algorithm from certain practical drawbacks. The first experiments with these early boosting algorithms were carried out by Drucker, Schapire and Simard [22] on an OCR task.

The AdaBoost algorithm, introduced in 1995 by Freund and Schapire [32], solved many of the practical difficulties of the earlier boosting algorithms, and is the focus of this paper. Pseudocode for AdaBoost is given in Table 8.1 in the slightly generalized form given by Schapire and Singer [70]. The algorithm takes as input a training set (x₁, y₁), ..., (xₘ, yₘ) where each xᵢ belongs to some domain or instance space X, and each label yᵢ is in some label set Y. For most of this paper, we assume Y = {-1, +1}; in Section 8.7, we discuss extensions to the multiclass case. AdaBoost calls a given weak or base learning algorithm repeatedly in a series of rounds t = 1, ..., T. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution on training example i on round t is denoted Dₜ(i). Initially, all weights are set equally, but on each round, the weights of incorrectly classified exam-
amples are increased so that the base learner is forced to focus on the hard examples in the training set.

The base learner's job is to find a base classifier \( h_t : X \rightarrow \mathbb{R} \) appropriate for the distribution \( D_t \). (Base classifiers were also called rules of thumb or weak prediction rules in Section 8.1.) In the simplest case, the range of each \( h_t \) is binary, i.e., restricted to \( \{-1, +1\} \); the base learner's job then is to minimize the error

\[
\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].
\]

Once the base classifier \( h_t \) has been received, AdaBoost chooses a parameter \( \alpha_t \in \mathbb{R} \) that intuitively measures the importance that it assigns to \( h_t \). In the figure, we have deliberately left the choice of \( \alpha_t \) unspecified. For binary \( h_t \), we typically set

\[
\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
\]

(8.1)

as in the original description of AdaBoost given by Freund and Schapire [32]. More on choosing \( \alpha_t \) follows in Section 8.3. The distribution \( D_t \) is then updated using the rule shown in the figure. The final or combined classifier \( H \) is a weighted majority vote of the \( T \) base classifiers where \( \alpha_t \) is the weight assigned to \( h_t \).

### 8.3 Analyzing the training error

The most basic theoretical property of AdaBoost concerns its ability to reduce the training error, i.e., the fraction of mistakes on the training set. Specifically, Schapire and Singer [70], in generalizing a theorem of Freund and Schapire [32], show that the training error of the final classifier is bounded as follows:

\[
\frac{1}{m} |\{i : H(x_i) \neq y_i\}| \leq \frac{1}{m} \sum_i \exp(-y_i f(x_i)) = \prod_t Z_t
\]

(8.2)

where henceforth we define

\[
f(x) = \sum_t \alpha_t h_t(x)
\]

(8.3)

so that \( H(x) = \text{sign}(f(x)) \). (For simplicity of notation, we write \( \sum_i \) and \( \sum_t \) as shorthand for \( \sum_{i=1}^m \) and \( \sum_{t=1}^T \), respectively.) The inequality follows from the fact that \( e^{-y_if(x_i)} \geq 1 \) if \( y_i \neq H(x_i) \). The equality can be proved straightforwardly by unraveling the recursive definition of \( D_t \).

Eq. (8.2) suggests that the training error can be reduced most rapidly (in a greedy way) by choosing \( \alpha_t \) and \( h_t \) on each round to minimize

\[
Z_t = \sum_i D_t(i) \exp(-\alpha_t y_i h_t(x_i)).
\]

(8.4)
In the case of binary classifiers, this leads to the choice of \( \alpha_t \) given in Eq. (8.1) and gives a bound on the training error of

\[
\prod_t Z_t = \prod_t \left[ 2 \sqrt{\epsilon_t (1 - \epsilon_t)} \right] = \prod_t \sqrt{1 - 4 \gamma_t^2} \leq \exp \left( -2 \sum_t \gamma_t^2 \right) \tag{8.5}
\]

where we define \( \gamma_t = 1/2 - \epsilon_t \). This bound was first proved by Freund and Schapire [32]. Thus, if each base classifier is slightly better than random so that \( \gamma_t \geq \gamma \) for some \( \gamma > 0 \), then the training error drops exponentially fast in \( T \) since the bound in (8.5) is at most \( e^{-2T\gamma^2} \). This bound, combined with the bounds on generalization error given below prove that AdaBoost is indeed a boosting algorithm in the sense that it can efficiently convert a true weak learning algorithm (that can always generate a classifier with a weak edge for any distribution) into a strong learning algorithm (that can generate a classifier with an arbitrarily low error rate, given sufficient data).

Eq. (8.2) points to the fact that, at heart, AdaBoost is a procedure for finding a linear combination \( f \) of base classifiers which attempts to minimize

\[
\sum_i \exp(-y_i f(x_i)) = \sum_i \exp \left( -y_i \sum_t \alpha_t h_t(x_i) \right). \tag{8.6}
\]

Essentially, on each round, AdaBoost chooses \( h_t \) (by calling the base learner) and then sets \( \alpha_t \) to add one more term to the accumulating weighted sum of base classifiers in such a way that the sum of exponents above will be maximally reduced. In other words, AdaBoost is doing a kind of steepest descent search to minimize Eq. (8.6) where the search is constrained at each step to follow coordinate directions (where we identify coordinates with the weights assigned to base classifiers). This view of boosting and its generalization are examined in considerable detail by Duffy and Helmbold [23], Mason et al. [51, 52] and Friedman [35]. See also Section 8.6.

Schapire and Singer [70] discuss the choice of \( \alpha_t \) and \( h_t \) in the case that \( h_t \) is real-valued (rather than binary). In this case, \( h_t(x) \) can be interpreted as a “confidence-rated prediction” in which the sign of \( h_t(x) \) is the predicted label, while the magnitude \( |h_t(x)| \) gives a measure of confidence. Here, Schapire and Singer advocate choosing \( \alpha_t \) and \( h_t \) so as to minimize \( Z_t \) ((8.4)) on each round.

### 8.4 Generalization error

In studying and designing learning algorithms, we are of course interested in performance on examples not seen during training, i.e., in the generalization error, the topic of this section. Unlike Section 8.3 where the training examples were arbitrary, here we assume that all examples (both train
and test) are generated i.i.d. from some unknown distribution on \( X \times Y \). The generalization error is the probability of misclassifying a new example, while the test error is the fraction of mistakes on a newly sampled test set (thus, generalization error is expected test error). Also, for simplicity, we restrict our attention to binary base classifiers.

Freund and Schapire [32] showed how to bound the generalization error of the final classifier in terms of its training error, the size \( m \) of the sample, the VC-dimension\(^3\) \( d \) of the base classifier space and the number of rounds \( T \) of boosting. Specifically, they used techniques from Baum and Haussler [5] to show that the generalization error, with high probability, is at most\(^4\)

\[
\hat{Pr} [H(x) \neq y] + \tilde{O} \left( \sqrt{\frac{Td}{m}} \right)
\]

where \( \hat{Pr} [\cdot] \) denotes empirical probability on the training sample. This bound suggests that boosting will overfit if run for too many rounds, i.e., as \( T \) becomes large. In fact, this sometimes does happen. However, in early experiments, several authors [8, 21, 59] observed empirically that boosting often does not overfit, even when run for thousands of rounds. Moreover, it was observed that AdaBoost would sometimes continue to drive down the generalization error long after the training error had reached zero, clearly contradicting the spirit of the bound above. For instance, the left side of Fig. 8.1 shows the training and test curves of running boosting on top of Quinlan’s C4.5 decision-tree learning algorithm [60] on the “letter” dataset.

In response to these empirical findings, Schapire et al. [69], following the work of Bartlett [3], gave an alternative analysis in terms of the margins of the training examples. The margin of example \((x, y)\) is defined to be

\[
\text{margin}_f(x, y) = \frac{y f(x)}{\sum_t |\alpha_t|} = \frac{y \sum_t \alpha_t h_t(x)}{\sum_t |\alpha_t|}.
\]

It is a number in \([-1, +1]\) and is positive if and only if \( H \) correctly classifies the example. Moreover, as before, the magnitude of the margin can be interpreted as a measure of confidence in the prediction. Schapire et al. proved that larger margins on the training set translate into a superior upper bound on the generalization error. Specifically, the generalization

---

\(^3\)The Vapnik-Chervonenkis (VC) dimension is a standard measure of the “complexity” of a space of binary functions. See, for instance, refs. [6, 76] for its definition and relation to learning theory.

\(^4\)The “soft-Oh” notation \( \tilde{O} (\cdot) \), here used rather informally, is meant to hide all logarithmic and constant factors (in the same way that standard “big-Oh” notation hides only constant factors).
Figure 8.1. Error curves and the margin distribution graph for boosting C4.5 on the letter dataset as reported by Schapire et al. [69]. Left: the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of rounds of boosting. The horizontal lines indicate the test error rate of the base classifier as well as the test error of the final combined classifier. Right: The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively.

error is at most

\[ \hat{\Pr} [\text{margin}_f(x, y) \leq \theta] + \tilde{O} \left( \sqrt{\frac{d}{m\theta^2}} \right) \]

for any \( \theta > 0 \) with high probability. Note that this bound is entirely independent of \( T \), the number of rounds of boosting. In addition, Schapire et al. proved that boosting is particularly aggressive at reducing the margin (in a quantifiable sense) since it concentrates on the examples with the smallest margins (whether positive or negative). Boosting’s effect on the margins can be seen empirically, for instance, on the right side of Fig. 8.1 which shows the cumulative distribution of margins of the training examples on the “letter” dataset. In this case, even after the training error reaches zero, boosting continues to increase the margins of the training examples effecting a corresponding drop in the test error.

Although the margins theory gives a qualitative explanation of the effectiveness of boosting, quantitatively, the bounds are rather weak. Breiman [9], for instance, shows empirically that one classifier can have a margin distribution that is uniformly better than that of another classifier, and yet be inferior in test accuracy. On the other hand, Koltchinskii, Panchenko and Lozano [44, 45, 46, 58] have recently proved new margin-theoretic bounds that are tight enough to give useful quantitative predictions.

Attempts (not always successful) to use the insights gleaned from the theory of margins have been made by several authors [9, 37, 50]. In addition, the margin theory points to a strong connection between boosting and the
support-vector machines of Vapnik and others [7, 14, 77] which explicitly attempt to maximize the minimum margin.

8.5 A connection to game theory and linear programming

The behavior of AdaBoost can also be understood in a game-theoretic setting as explored by Freund and Schapire [31, 33] (see also Grove and Schuurmans [37] and Breiman [9]). In classical game theory, it is possible to put any two-person, zero-sum game in the form of a matrix \( M \). To play the game, one player chooses a row \( i \) and the other player chooses a column \( j \). The loss to the row player (which is the same as the payoff to the column player) is \( M_{ij} \). More generally, the two sides may play randomly, choosing distributions \( P \) and \( Q \) over rows or columns, respectively. The expected loss then is \( P^T M Q \).

Boosting can be viewed as repeated play of a particular game matrix. Assume that the base classifiers are binary, and let \( \mathcal{H} = \{ h_1, \ldots, h_n \} \) be the entire base classifier space (which we assume for now to be finite). For a fixed training set \( (x_1, y_1), \ldots, (x_m, y_m) \), the game matrix \( M \) has \( m \) rows and \( n \) columns where

\[
M_{ij} = \begin{cases} 
1 & \text{if } h_j(x_i) = y_i \\
0 & \text{otherwise.} 
\end{cases}
\]

The row player now is the boosting algorithm, and the column player is the base learner. The boosting algorithm’s choice of a distribution \( D_t \) over training examples becomes a distribution \( P \) over rows of \( M \), while the base learner’s choice of a base classifier \( h_t \) becomes the choice of a column \( j \) of \( M \).

As an example of the connection between boosting and game theory, consider von Neumann’s famous minmax theorem which states that

\[
\max_Q \min_P P^T MQ = \min_P \max_Q P^T MQ
\]

for any matrix \( M \). When applied to the matrix just defined and reinterpreted in the boosting setting, this can be shown to have the following meaning: If, for any distribution over examples, there exists a base classifier with error at most \( 1/2 - \gamma \), then there exists a convex combination of base classifiers with a margin of at least \( 2\gamma \) on all training examples. AdaBoost seeks to find such a final classifier with high margin on all examples by combining many base classifiers; so in a sense, the minmax theorem tells us that AdaBoost at least has the potential for success since, given a “good” base learner, there must exist a good combination of base classifiers. Going much further, AdaBoost can be shown to be a special case of a more general algorithm for playing repeated games, or for approximately
solving matrix games. This shows that, asymptotically, the distribution over training examples as well as the weights over base classifiers in the final classifier have game-theoretic interpretations as approximate minmax or maxmin strategies.

The problem of solving (finding optimal strategies for) a zero-sum game is well known to be solvable using linear programming. Thus, this formulation of the boosting problem as a game also connects boosting to linear, and more generally convex, programming. This connection has led to new algorithms and insights as explored by Rätsch et al. [62], Grove and Schuurmans [37] and Demiriz, Bennett and Shawe-Taylor [17].

In another direction, Schapire [68] describes and analyzes the generalization of both AdaBoost and Freund’s earlier “boost-by-majority” algorithm [26] to a broader family of repeated games called “drifting games.”

8.6 Boosting and logistic regression

Classification generally is the problem of predicting the label $y$ of an example $x$ with the intention of minimizing the probability of an incorrect prediction. However, it is often useful to estimate the probability of a particular label. Friedman, Hastie and Tibshirani [34] suggested a method for using the output of AdaBoost to make reasonable estimates of such probabilities. Specifically, they suggested using a logistic function, and estimating

$$\Pr_f [y = +1 \mid x] = \frac{e^{f(x)}}{e^{f(x)} + e^{-f(x)}} \quad (8.7)$$

where, as usual, $f(x)$ is the weighted average of base classifiers produced by AdaBoost ((8.3)). The rationale for this choice is the close connection between the log loss (negative log likelihood) of such a model, namely,

$$\sum_i \ln \left( 1 + e^{-2y_i f(x_i)} \right) \quad (8.8)$$

and the function that, we have already noted, AdaBoost attempts to minimize:

$$\sum_i e^{-y_i f(x_i)} \quad (8.9)$$

Specifically, it can be verified that Eq. (8.8) is upper bounded by Eq. (8.9). In addition, if we add the constant $1 - \ln 2$ to Eq. (8.8) (which does not affect its minimization), then it can be verified that the resulting function and the one in Eq. (8.9) have identical Taylor expansions around zero up to second order; thus, their behavior near zero is very similar. Finally, it
can be shown that, for any distribution over pairs \((x, y)\), the expectations

\[
E \left[ \ln \left( 1 + e^{-2yf(x)} \right) \right]
\]

and

\[
E \left[ e^{-yf(x)} \right]
\]

are minimized by the same (unconstrained) function \(f\), namely,

\[
f(x) = \frac{1}{2} \ln \left( \frac{\Pr [ y = +1 \mid x]}{\Pr [ y = -1 \mid x]} \right).
\]

Thus, for all these reasons, minimizing Eq. (8.9), as is done by AdaBoost, can be viewed as a method of approximately minimizing the negative log likelihood given in Eq. (8.8). Therefore, we may expect Eq. (8.7) to give a reasonable probability estimate.

Of course, as Friedman, Hastie and Tibshirani point out, rather than minimizing the exponential loss in (8.6), we could attempt instead to directly minimize the logistic loss in (8.8). To this end, they propose their LogitBoost algorithm. A different, more direct modification of AdaBoost for logistic loss was proposed by Collins, Schapire and Singer [13]. Following up on work by Kivinen and Warmuth [43] and Lafferty [47], they derive this algorithm using a unification of logistic regression and boosting based on Bregman distances. This work further connects boosting to the maximum-entropy literature, particularly the iterative-scaling family of algorithms [15, 16]. They also give unified proofs of convergence to optimality for a family of new and old algorithms, including AdaBoost, for both the exponential loss used by AdaBoost and the logistic loss used for logistic regression. See also the later work of Lebanon and Lafferty [48] who showed that logistic regression and boosting are in fact solving the same constrained optimization problem, except that in boosting, certain normalization constraints have been dropped.

For logistic regression, we attempt to minimize the loss function

\[
\sum_i \ln \left( 1 + e^{-y_if(x_i)} \right)
\]

which is the same as in (8.8) except for an inconsequential change of constants in the exponent. The modification of AdaBoost proposed by Collins, Schapire and Singer to handle this loss function is particularly simple. In AdaBoost, unraveling the definition of \(D_t\) given in Table 8.1 shows that \(D_t(i)\) is proportional (i.e., equal up to normalization) to

\[
\exp \left( -y_if_{t-1}(x_i) \right)
\]

where we define

\[
f_t(x) = \sum_{t'} \alpha_{t'} h_{t'}(x).
\]
To minimize the loss function in (8.10), the only necessary modification is to redefine $D_t(i)$ to be proportional to

$$
\frac{1}{1 + \exp(y_i f_{t-1}(x_i))}.
$$

A very similar algorithm is described by Duffy and Helmbold [23]. Note that in each case, the weight on the examples, viewed as a vector, is proportional to the negative gradient of the respective loss function. This is because both algorithms are doing a kind of functional gradient descent, an observation that is spelled out and exploited by Breiman [9], Duffy and Helmbold [23], Mason et al. [51, 52] and Friedman [35].

Besides logistic regression, there have been a number of approaches taken to apply boosting to more general regression problems in which the labels $y_t$ are real numbers and the goal is to produce real-valued predictions that are close to these labels. Some of these, such as those of Ridgeway [63] and Freund and Schapire [32], attempt to reduce the regression problem to a classification problem. Others, such as those of Friedman [35] and Duffy and Helmbold [24] use the functional gradient descent view of boosting to derive algorithms that directly minimize a loss function appropriate for regression. Another boosting-based approach to regression was proposed by Drucker [20].

### 8.7 Multiclass classification

There are several methods of extending AdaBoost to the multiclass case. The most straightforward generalization [32], called AdaBoost.M1, is adequate when the base learner is strong enough to achieve reasonably high accuracy, even on the hard distributions created by AdaBoost. However, this method fails if the base learner cannot achieve at least 50% accuracy when run on these hard distributions.

For the latter case, several more sophisticated methods have been developed. These generally work by reducing the multiclass problem to a larger binary problem. Schapire and Singer’s [70] algorithm AdaBoost.MH works by creating a set of binary problems, for each example $x$ and each possible label $y$, of the form: “For example $x$, is the correct label $y$ or is it one of the other labels?” Freund and Schapire’s [32] algorithm AdaBoost.M2 (which is a special case of Schapire and Singer’s [70] AdaBoost.MR algorithm) instead creates binary problems, for each example $x$ with correct label $y$ and each incorrect label $y'$ of the form: “For example $x$, is the correct label $y$ or $y'$?”

These methods require additional effort in the design of the base learning algorithm. A different technique [67], which incorporates Dietterich and Bakiri’s [19] method of error-correcting output codes, achieves similar provable bounds to those of AdaBoost.MH and AdaBoost.M2, but can be
used with any base learner that can handle simple, binary labeled data. Schapire and Singer [70] and Allwein, Schapire and Singer [2] give yet another method of combining boosting with error-correcting output codes.

8.8 Incorporating human knowledge

Boosting, like many machine-learning methods, is entirely data-driven in the sense that the classifier it generates is derived exclusively from the evidence present in the training data itself. When data is abundant, this approach makes sense. However, in some applications, data may be severely limited, but there may be human knowledge that, in principle, might compensate for the lack of data.

In its standard form, boosting does not allow for the direct incorporation of such prior knowledge. Nevertheless, Rochery et al. [64, 65] describe a modification of boosting that combines and balances human expertise with available training data. The aim of the approach 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 first step in this approach is for a human expert to construct by hand a rule \( p \) mapping each instance \( x \) to an estimated probability \( p(x) \in [0, 1] \) that is interpreted as the guessed probability that instance \( x \) will appear with label +1. There are various methods for constructing such a function \( p \), and the hope is that this difficult-to-build function need not be highly accurate for the approach to be effective.

Rochery et al.’s basic idea is to replace the logistic loss function in (8.10) with one that incorporates prior knowledge, namely,

\[
\sum_i \ln \left( 1 + e^{-y_i f(x_i)} \right) + \eta \sum_i \text{RE} \left( p(x_i) \parallel \frac{1}{1 + e^{-f(x_i)}} \right)
\]

where \( \text{RE} (p \parallel q) = p \ln(p/q) + (1 - p) \ln((1 - p)/(1 - q)) \) is binary relative entropy. The first term is the same as that in (8.10). The second term gives a measure of the distance from the model built by boosting to the human’s model. Thus, we balance the conditional likelihood of the data against the distance from our model to the human’s model. The relative importance of the two terms is controlled by the parameter \( \eta \).

8.9 Experiments and applications

Practically, AdaBoost has many advantages. It is fast, simple and easy to program. It has no parameters to tune (except for the number of round \( T \)). It requires no prior knowledge about the base learner and so can be
Figure 8.2. Comparison of C4.5 versus boosting stumps and boosting C4.5 on a set of 27 benchmark problems as reported by Freund and Schapire [30]. Each point in each scatterplot shows the test error rate of the two competing algorithms on a single benchmark. The y-coordinate of each point gives the test error rate (in percent) of C4.5 on the given benchmark, and the x-coordinate gives the error rate of boosting stumps (left plot) or boosting C4.5 (right plot). All error rates have been averaged over multiple runs.

flexibly combined with *any* method for finding base classifiers. Finally, it comes with a set of theoretical guarantees given sufficient data and a base learner that can reliably provide only moderately accurate base classifiers. This is a shift in mind set for the learning-system designer: instead of trying to design a learning algorithm that is accurate over the entire space, we can instead focus on finding base learning algorithms that only need to be better than random.

On the other hand, some caveats are certainly in order. The actual performance of boosting on a particular problem is clearly dependent on the data and the base learner. Consistent with theory, boosting can fail to perform well given insufficient data, overly complex base classifiers or base classifiers that are too weak. Boosting seems to be especially susceptible to noise [18] (more on this in Section sec:exps).

AdaBoost has been tested empirically by many researchers, including [4, 18, 21, 40, 49, 59, 73]. For instance, Freund and Schapire [30] tested AdaBoost on a set of UCI benchmark datasets [54] using C4.5 [60] as a base learning algorithm, as well as an algorithm that finds the best "decision stump" or single-test decision tree. Some of the results of these experiments are shown in Fig. 8.2. As can be seen from this figure, even boosting the weak decision stumps can usually give as good results as C4.5,
Figure 8.3. Comparison of error rates for AdaBoost and four other text categorization methods (naive Bayes, probabilistic TF-IDF, Rocchio and sleeping experts) as reported by Schapire and Singer [71]. The algorithms were tested on two text corpora — Reuters newswire articles (left) and AP newswire headlines (right) — and with varying numbers of class labels as indicated on the x-axis of each figure.

while boosting C4.5 generally gives the decision-tree algorithm a significant improvement in performance.

In another set of experiments, Schapire and Singer [71] used boosting for text categorization tasks. For this work, base classifiers were used that test on the presence or absence of a word or phrase. Some results of these experiments comparing AdaBoost to four other methods are shown in Fig. 8.3. In nearly all of these experiments and for all of the performance measures tested, boosting performed as well or significantly better than the other methods tested. As shown in Fig. 8.4, these experiments also demonstrated the effectiveness of using confidence-rated predictions [70], mentioned in Section 8.3 as a means of speeding up boosting.

Boosting has also been applied to text filtering [72] and routing [39], “ranking” problems [28], learning problems arising in natural language processing [1, 12, 25, 38, 55, 78], image retrieval [74], medical diagnosis [53], and customer monitoring and segmentation [56, 57].

Rochery et al.’s [64, 65] method of incorporating human knowledge into boosting, described in Section 8.8, was applied to two speech categorization tasks. In this case, the prior knowledge took the form of a set of hand-built rules mapping keywords to predicted categories. The results are shown in Fig. 8.5.

The final classifier produced by AdaBoost when used, for instance, with a decision-tree base learning algorithm, can be extremely complex and difficult to comprehend. With greater care, a more human-understandable final classifier can be obtained using boosting. Cohen and Singer [11] showed how to design a base learning algorithm that, when combined with AdaBoost, results in a final classifier consisting of a relatively small set of rules similar to those generated by systems like RIPPER [10], IREP [36] and C4.5rules [60]. Cohen and Singer’s system, called SLIPPER, is fast, accurate and produces quite compact rule sets. In other work, Freund and Mason [29] showed how to apply boosting to learn a generalization of decision trees called “alternating trees.” Their algorithm produces a single alternating tree rather than an ensemble of trees as would be obtained by running AdaBoost on top of a decision-tree learning algorithm. On the other hand, their learning algorithm achieves error rates comparable to those of a whole ensemble of trees.

A nice property of AdaBoost is its ability to identify outliers, i.e., examples that are either mislabeled in the training data, or that are inherently ambiguous and hard to categorize. Because AdaBoost focuses its weight on the hardest examples, the examples with the highest weight often turn
out to be outliers. An example of this phenomenon can be seen in Fig. 8.6 taken from an OCR experiment conducted by Freund and Schapire [30].

When the number of outliers is very large, the emphasis placed on the hard examples can become detrimental to the performance of AdaBoost. This was demonstrated very convincingly by Dietterich [18]. Friedman, Hastie and Tibshirani [34] suggested a variant of AdaBoost, called “Gentle AdaBoost” that puts less emphasis on outliers. Rüetsch, Onoda and Müller [61] show how to regularize AdaBoost to handle noisy data. Freund [27] suggested another algorithm, called “BrownBoost,” that takes a more radical approach that de-emphasizes outliers when it seems clear that they are “too hard” to classify correctly. This algorithm, which is an adaptive version of Freund’s [26] “boost-by-majority” algorithm, demonstrates an intriguing connection between boosting and Brownian motion.

8.10 Conclusion

In this overview, we have seen that there have emerged a great many views or interpretations of AdaBoost. First and foremost, AdaBoost is a genuine boosting algorithm: given access to a true weak learning algorithm that always performs a little bit better than random guessing on every distribution over the training set, we can prove arbitrarily good bounds on the training error and generalization error of AdaBoost.
Besides this original view, AdaBoost has been interpreted as a procedure based on functional gradient descent, as an approximation of logistic regression and as a repeated-game playing algorithm. AdaBoost has also been shown to be related to many other topics, such as game theory and linear programming, Bregman distances, support-vector machines, Brownian motion, logistic regression and maximum-entropy methods such as iterative scaling.

All of these connections and interpretations have greatly enhanced our understanding of boosting and contributed to its extension in ever more practical directions, such as to logistic regression and other loss-minimization problems, to multiclass problems, to incorporate regularization and to allow the integration of prior background knowledge.

We also have discussed a few of the growing number of applications of AdaBoost to practical machine learning problems, such as text and speech categorization.

References


8. The Boosting Approach to Machine Learning: An Overview


8. The Boosting Approach to Machine Learning: An Overview


8. The Boosting Approach to Machine Learning: An Overview


