Explaining AdaBoost

Robert E. Schapire

Abstract Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules. The AdaBoost algorithm of Freund and Schapire was the first practical boosting algorithm, and remains one of the most widely used and studied, with applications in numerous fields. This chapter aims to review some of the many perspectives and analyses of AdaBoost that have been applied to explain or understand it as a learning method, with comparisons of both the strengths and weaknesses of the various approaches.

1 Introduction

Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules. The AdaBoost algorithm of Freund and Schapire [10] was the first practical boosting algorithm, and remains one of the most widely used and studied, with applications in numerous fields. Over the years, a great variety of attempts have been made to "explain" AdaBoost as a learning algorithm, that is, to understand why it works, how it works, and when it works (or fails). It is by understanding the nature of learning at its foundation — both generally and with regard to particular algorithms and phenomena — that the field is able to move forward. Indeed, this has been the lesson of Vapnik's life work.

This chapter aims to review some of the numerous perspectives and analyses of AdaBoost that have been applied to explain or understand it as a learning method, with comparisons of both the strengths and weaknesses of the various approaches. For brevity, the presentation is high level with few technical details. A much more

Robert E. Schapire
Princeton University, Dept. of Computer Science, 35 Olden Street, Princeton, NJ 08540 USA, e-mail: schapire@cs.princeton.edu

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select h_t with low weighted error:

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

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$.
- Update, for $i = 1, \ldots, m$:

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

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

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

Fig. 1 The boosting algorithm AdaBoost.

in-depth exposition of most of the topics of this chapter, including more complete references to the relevant literature, can be found in the recent book by Schapire and Freund [30].

Pseudocode for AdaBoost is shown in Figure 1. Here we are given m labeled training examples $(x_1, y_1), \ldots, (x_m, y_m)$ where the x_i's are in some domain $\mathcal{X}$, and the labels $y_i \in \{-1, +1\}$. On each round $t = 1, \ldots, T$, a distribution D_t is computed as in the figure over the m training examples, and a given *weak learner* or *weak learning algorithm* is applied to find a *weak hypothesis* $h_t : \mathcal{X} \to \{-1, +1\}$, where the aim of the weak learner is to find a weak hypothesis with low weighted error ε_t relative to D_t. The *final* or *combined hypothesis* H computes the sign of a weighted combination of weak hypotheses

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

This is equivalent to saying that H is computed as a weighted majority vote of the weak hypotheses h_t where each is assigned weight α_t. (In this chapter, we use the terms "hypothesis" and "classifier" interchangeably.)

2 Direct Application of VC Theory

We begin by considering how the general theory of Vapnik and Chervonenkis can be applied directly to AdaBoost.

Intuitively, for a learned classifier to be effective and accurate in its predictions, it should meet three conditions: (1) it should have been trained on "enough" training examples; (2) it should provide a good fit to those training examples (usually meaning that it should have low training error); and (3) it should be "simple." This last condition, our expectation that simpler rules are better, is often referred to as *Occam's razor*.

In formalizing these conditions, Vapnik and Chervonenkis [34, 35] established a foundation for understanding the fundamental nature of learning, laying the groundwork for the design of effective and principled learning algorithms. Specifically, they derived upper bounds on the generalization error of a classifier that could be stated in terms of the three conditions above, and along the way, provided workable definitions (such as the VC-dimension) of the slippery and mysterious notion of simplicity.

To understand AdaBoost, the very general and encompassing VC theory is the most sensible starting point. All analyses of learning methods depend in some way on assumptions, since otherwise, learning is quite impossible. From the very beginning, much of the work studying boosting has been based on the assumption that each of the weak hypotheses has accuracy just a little bit better than random guessing; for two-class problems, this means they should each have error below $1/2$, that is, each ε_t should be at most $1/2-\gamma$ for some $\gamma > 0$. This assumption, called the *weak learning condition*, is intrinsic to the mathematical definition of a boosting algorithm which, given this assumption and sufficient data, can provably produce a final hypothesis with arbitrarily small generalization error.

Given the weak learning condition, it is possible to prove that the training error of AdaBoost's final hypothesis decreases to zero very rapidly; in fact, in just $O(\log m)$ rounds (ignoring all other parameters of the problem), the final hypothesis will perfectly fit the training set [10]. Furthermore, we can measure the complexity (that is, lack of simplicity) of the final hypothesis using the VC-dimension which can be computed using combinatorial arguments [2, 10]. Having analyzed both the complexity and training fit of the final hypothesis, one can immediately apply the VC theory to obtain a bound on its generalization error.

Such an analysis predicts the kind of behavior depicted on the left of Figure 2 which shows the error (both training and test) of the final hypothesis as a function of the number of rounds of boosting. As noted above, we expect training error to drop very quickly, but at the same time, the VC-dimension of the final hypothesis is increasing roughly linearly with the number of rounds T. Thus, with improved fit to the training set, the test error drops at first, but then rises again as a result of the final hypothesis becoming overly complex. This is classic overfitting behavior. Indeed, overfitting can happen with AdaBoost as seen on the right side of the figure which shows training and test error on an actual benchmark dataset. However, as we will see shortly, AdaBoost often does not overfit, apparently in direct contradiction of what is predicted by VC theory.

Summarizing this first approach to understanding AdaBoost, a direct application of VC theory shows that AdaBoost can work if provided with enough data and simple weak classifiers which satisfy the weak learning condition, and if run for

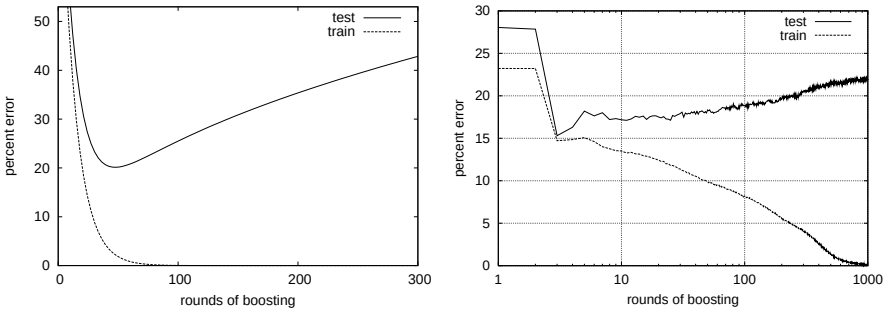

Fig. 2 *Left:* A plot of the theoretical training and test percent errors for AdaBoost, as predicted by the arguments of Section 2. *Right:* The training and test percent error rates obtained using boosting on the Cleveland heart-disease benchmark dataset. (Reprinted from [30] with permission of MIT Press.)

enough but not too many rounds. The theory captures the cases in which AdaBoost does overfit, but also predicts (incorrectly) that AdaBoost will *always* overfit.

Like all of the approaches to be discussed in this chapter, the numerical bounds on generalization error that can be obtained using this technique are horrendously loose.

3 The Margins Explanation

Another actual typical run on a different benchmark dataset is shown on the left of Figure 3. In this case, boosting was used in combination with the decision-tree learning algorithm C4.5 [26] as the weak learner. A single decision tree produced by C4.5 on this dataset has a test error rate of 13.8%. In this example, boosting very quickly drives down the training error; in fact, after only five rounds, the training error is zero so that all training examples are correctly classified. (Note that there is no reason why AdaBoost cannot proceed beyond this point.)

The test performance of boosting on this dataset is extremely good, far better than a single decision tree. And surprisingly, unlike the earlier example, the test error on this dataset never increases, even after 1000 trees have been combined, by which point, the combined classifier involves more than two million decision nodes. Even after the training error hits zero, the test error continues to drop, from 8.4% on round 5 down to 3.1% on round 1000. This pronounced lack of overfitting seems to flatly contradict the intuition and theory discussed in Section 2 which says that simpler is better. Surely, a combination of five trees is much, much simpler than a combination of 1000 trees (about 200 times simpler, in terms of raw size), and both perform equally well on the training set (perfectly, in fact). So how can it be that the far larger and more complex combined classifier performs so much better on the test set?

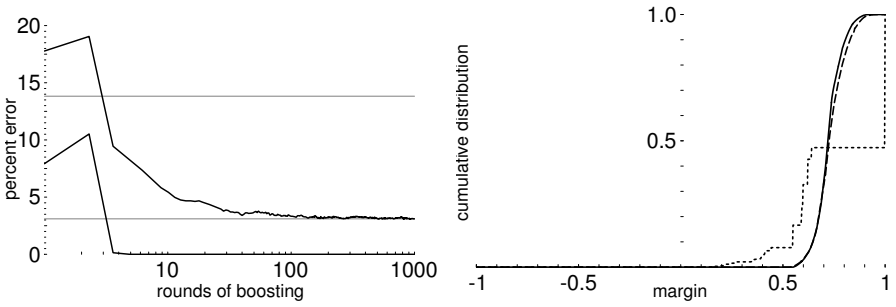

Fig. 3 *Left:* The training and test percent error rates obtained using boosting on an OCR dataset with C4.5 as the base learner. The top and bottom curves are test and training error, respectively. The top horizontal line shows the test error rate using just C4.5. The bottom line shows the final test error rate of AdaBoost after 1000 rounds. *Right:* The margin distribution graph for this same case showing the cumulative distribution of margins of the training instances after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively. (Both figures are reprinted from [31] with permission of the Institute of Mathematical Statistics.)

Such resistance to overfitting is typical of boosting, although, as we saw earlier, boosting certainly *can* overfit. This resistance is one of the properties that make it such an attractive learning algorithm. But how can we understand this behavior?

The *margins explanation* of Schapire et al. [31] was proposed as a way out of this seeming paradox. Briefly, the main idea is the following. The description above of AdaBoost's performance on the training set only took into account the training error, which is zero already after only five rounds. However, training error only tells part of the story in that it only reports the number of examples that are correctly or incorrectly classified. Instead, to understand AdaBoost, we also need to consider how *confident* are the predictions being made by the algorithm. According to this explanation, although the training error — that is, whether or not the predictions are correct — is not changing after round 5, the confidence in those predictions is increasing dramatically with additional rounds of boosting. And it is this increase in confidence which accounts for the better generalization performance.

To measure confidence, we use a quantity called the *margin*. Recall that the combined classifier H is simply a weighted majority vote — that is, the result of a small-scale "election" — of the predictions of the weak classifiers. In a real-world election, confidence in the outcome is measured by the margin of victory, the difference between the fraction of votes received by the winner and the fraction of votes received by the loser. In the same way, we can define margin in our setting as the difference between the weighted fraction of the weak classifiers predicting the correct label and the weighted fraction predicting the incorrect label. When this vote is very close, so that the predicted label $H(x)$ is based on a narrow majority, the margin will be small in magnitude and, intuitively, we will have little confidence in the prediction. On the other hand, when the prediction $H(x)$ is based on a clear and substantial majority of the base classifiers, the margin will be correspondingly large lending greater confidence in the predicted label. Thus, the magnitude of the margin

is a reasonable measure of confidence. Furthermore, the margin will be positive if and only if the overall prediction $H(x)$ is correct.

We can visualize the effect AdaBoost has on the margins of the training examples by plotting their distribution. In particular, we can create a plot showing, for each $\theta \in [-1,+1]$, the fraction of training examples with margin at most θ. For such a cumulative distribution curve, the bulk of the distribution lies where the curve rises the most steeply. Figure 3, on the right, shows such a *margin distribution graph* for the same dataset as above, showing the margin distribution after 5, 100 and 1000 rounds of boosting. Whereas nothing at all is happening to the training error, these curves expose dramatic changes happening on the margin distribution. For instance, after five rounds, although the training error is zero (so that no examples have negative margin), a rather substantial fraction of the training examples (7.7%) have margin below 0.5. By round 100, all of these examples have been swept to the right so that not a single example has margin below 0.5, and nearly all have margin above 0.6.

Thus, this example is indicative of the powerful effect AdaBoost has on the margins, aggressively pushing up those examples with small or negative margin. Moreover, comparing the two sides of Figure 3, we see that this overall increase in the margins appears to be correlated with better performance on the test set.

AdaBoost can be analyzed theoretically along exactly these lines. It is possible to prove first a bound on the generalization error of AdaBoost — or any other voting method — that depends only on the margins of the training examples, and *not* on the number of rounds of boosting. Such a bound predicts that AdaBoost will not overfit regardless of how long it is run, provided that large margins can be achieved (and provided, of course, that the weak classifiers are not too complex relative to the size of the training set).

The second part of such an analysis is to prove that, as observed empirically in Figure 3, AdaBoost generally tends to increase the margins of all training examples, and moreover, the higher the accuracy of the weak hypotheses, the larger will be the margins.

All this suggests that perhaps a more effective learning algorithm could be designed by explicitly attempting to maximize the margins. This was attempted by Breiman [4] (among others) who created an algorithm called arc-gv for maximizing the smallest margin of any training example. Although this algorithm did indeed produce larger margins, its test performance turned out to be slightly *worse* than AdaBoost, apparently contradicting the margins theory. In a follow-up study, Reyzin and Schapire [28] suggested two possible explanations. First, more aggressive margin maximization seems to produce more complex weak hypotheses which tends to raise the potential for overfitting, confounding the experiments. And second, in some cases, arc-gv produces a higher *minimum* margin, but a distribution of margins that is lower overall.

In summary, according to the margins explanation, AdaBoost will succeed without overfitting if the weak-hypothesis accuracies are substantially better than random (since this will lead to large margins), and if provided with enough data relative to the complexity of the weak hypotheses. This is really the only known theory that

explains the cases in which overfitting is not observed. On the other hand, attempted extensions of AdaBoost based on direct maximization of margins have not been entirely successful, though work in this area is ongoing (see, for instance, [22, 36]).

4 Loss Minimization

Many, perhaps even most, learning and statistical methods that are in common use can be viewed as procedures for minimizing a *loss function* (also called a cost or objective function) that in some way measures how well a model fits the observed data. A classic example is least-squares regression in which a sum of squared errors is minimized. AdaBoost, though not originally designed for this purpose, also turns out to minimize a particular loss function. Viewing the algorithm in this light can be helpful for a number of reasons. First, such an understanding can help to clarify the goal of the algorithm and can be useful in proving convergence properties. And second, by decoupling the algorithm from its objective, we may be able to derive better or faster algorithms for the same objective, or alternatively, we might be able to generalize AdaBoost for new challenges.

AdaBoost can be understood as a procedure for greedily minimizing what has come to be called the *exponential loss*, namely,

$$\frac{1}{m}\sum_{i=1}^{m}\exp(-y_i F(x_i))$$

where $F(x)$ is as given in Eq. (1). In other words, it can be shown that the choices of α_t and h_t on each round happen to be the same as would be chosen so as to cause the greatest decrease in this loss. This connection was first observed by Breiman [4] and later expanded upon by others [7, 11, 23, 25, 27, 32].

Why does this loss make sense? Intuitively, minimizing exponential loss strongly favors the choice of a function F for which the sign of $F(x_i)$ is likely to agree with the correct label y_i; since the final hypothesis H is computed as the sign of F, this is exactly the behavior we seek in attempting to minimize the number of mistaken classifications. Another argument that is sometimes made is that the real goal of minimizing classification errors requires the optimization of an objective that is not continuous, differentiable or easily minimized, but which can be approximated by a smooth and convex "surrogate" objective function such as the exponential loss. The exponential loss is also related to the loss used for logistic regression [11].

As a procedure for minimizing this loss, AdaBoost can be viewed as a form of coordinate descent (in which each step is made greedily along one of the coordinate directions), as noted by Breiman [4]. Alternatively, AdaBoost can be viewed as a form of functional gradient descent, as observed by Mason et al. [23] and Friedman [12]. This understanding has led to the immediate generalization of boosting to a wide range of other learning problems and loss functions, such as regression.

From this perspective, 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 plausibly 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).

To demonstrate this point, consider the following experiment from Schapire and Freund [30], which is similar in spirit to the work of Mease and Wyner [24, 37]. Data for this experiment 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 were 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 $\mathbf{x}$. The weak hypotheses used were associated with coordinates so that each was of the form $h(\mathbf{x}) = x_j$ for all $\mathbf{x}$, and for some coordinate j. (The negatives of these were also included.)

Three different algorithms were tested. The first was ordinary AdaBoost using an exhaustive weak learner that, on each round, finds the minimum weighted-error weak hypothesis. We refer to this as *exhaustive AdaBoost*. The second algorithm was *gradient descent* on the exponential loss function (which can be written in a parametric form so that ordinary gradient descent can be applied). The third algorithm was actually the same as AdaBoost except that the weak learner does not actively search for the best weak hypothesis, but rather selects one uniformly at random from the space of possible weak hypotheses; we refer to this method as *random AdaBoost*.

All three algorithms are guaranteed to minimize the exponential loss, 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 is convex, and therefore can have no local minima. But it is possible, and even typical, for the minimum either to be non-unique, or to not exist at all at any finite setting of the parameters. Therefore, different algorithms for the same (convex) loss can yield very different hypotheses.

The results of these experiments are shown in Table 1. Regarding speed (measured by number of rounds), the table shows that 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.

	% 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]

Table 1 Results of the experiment described in Section 4. The numbers in brackets show 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 at which the exponential loss first dropped below the specified values. All results are averaged over ten random repetitions of the experiment. (Reprinted from [30] with permission of MIT Press.)

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 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.

So to summarize, 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, it is important to keep in mind that this perspective is rather limited in terms of what it can tell us about AdaBoost's accuracy as a learning algorithm. The example above demonstrates that any understanding of AdaBoost's generalization capabilities must in some way take into account the particular dynamics of the algorithm — not just the objective function, but what procedure is actually being used to minimize it.

5 Regularization

Without question, AdaBoost minimizes exponential loss. And yet, as was just seen, other algorithms for minimizing this same loss can perform far worse. If the choice of loss function cannot explain how AdaBoost avoids the poor performance of these other algorithms, then how does it do it?

In general, when minimizing a loss function, it has become quite popular and standard to *regularize*, that is, to modify or constrain the optimization problem in a way that attempts to avoid overfitting by limiting complexity or encouraging smoothness. In our context, we have seen that AdaBoost constructs a linear combi-

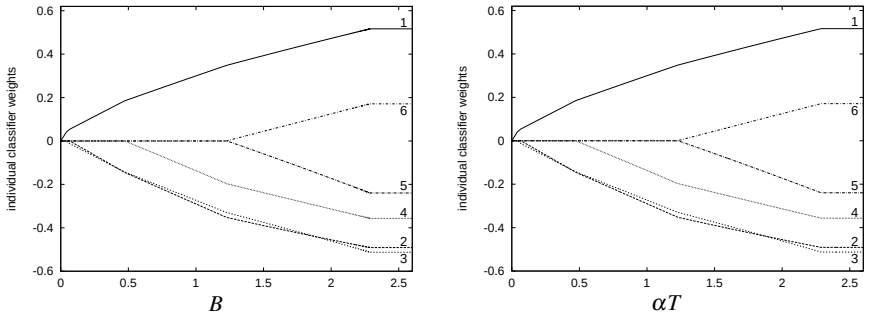

Fig. 4 The trajectories of the weight vectors computed on a benchmark dataset using only six possible weak hypotheses. Trajectories are plotted for ℓ_1-regularized exponential loss as the parameter B varies (left), and for a variant of AdaBoost in which $\alpha_t = \alpha = 10^{-6}$ on every round (right). Each figure includes one curve for each of the six weak hypotheses showing its associated weight as a function of the total weight added. (Reprinted from [30] with permission of MIT Press.)

nation F of weak hypotheses (as in Eq. (1)), and does so in a way that minimizes exponential loss over all such linear combinations. To regularize, we might instead choose our objective to be the minimization of this same loss, but subject to the constraint that the weak-hypothesis weights appearing in F, when viewed collectively as a vector, have ℓ_1-norm bounded by some pre-set parameter $B > 0$. There are many other ways of regularizing (for instance, using a different norm), but this particular form based on the ℓ_1-norm, sometimes called the "lasso," has the especially favorable property that it seems to encourage sparsity, that is, a solution with relatively few nonzero weights [33].

AdaBoost certainly does not explicitly regularize — there is nothing about the algorithm that overtly limits the weights on the weak hypotheses. Nevertheless, is it possible that it is somehow applying some kind of implicit form of regularization? In fact, it turns out that a simple variant of AdaBoost, when stopped after any number of rounds, can often be viewed as providing an approximate solution to ℓ_1-regularized minimization of exponential loss. To see this, consider an experiment in which we compute the solution to this regularized optimization problem for all possible values of the pre-set bound B. As B varies, these weight vectors trace out a path or trajectory, which can be plotted in the unrealistic but illustrative case that the space of possible weak hypotheses is very small. This is shown on the left of Figure 4 on benchmark data using just six possible weak hyptheses. Each curve corresponds to one of the six weak hypotheses and plots its weight at the regularized solution as a function of B. Thus, the figure depicts the entire trajectory.

For comparison, consider a variant of AdaBoost in which α_t, rather than being set as in Figure 1, is chosen on each round to be equal to a fixed small constant $\alpha > 0$. As above, we can plot the trajectory of the weights on the weak hypotheses which define the combined classifier as a function of the number of iterations T, multiplied by the constant α so that the resulting scale αT is equal to the cumulative sum of

weight updates after T iterations. This is shown, for the same dataset, on the right of Figure 4 (using $\alpha = 10^{-6}$).

Remarkably, the two plots are practically indistinguishable. This shows that, at least in this case, a variant of AdaBoost, when run for T rounds, computes essentially the same solution vectors as when using ℓ_1-regularization with B set to αT. Thus, early stopping — that is, halting boosting after a limited number of rounds — is in this sense apparently equivalent to regularization. This correspondence was first observed by Hastie, Tibshirani and Friedman [13], and explored further by Rosset, Zhu and Hastie [29]. Later, Zhao and Yu [40] showed theoretically that the correspondence will hold generally under certain but not all conditions.

All this suggests a plausible explanation for how AdaBoost works: Regularization is a general technique that protects against overfitting by constraining, smoothing, and/or promoting sparsity. As just discussed, AdaBoost with early stopping is related to ℓ_1-regularization. Therefore, AdaBoost avoids overfitting through implicit regularization.

However, there are important deficiencies with this argument. First of all, strictly speaking, it does not apply to AdaBoost, but only to a variant of AdaBoost in which the weights on each round are set to a small fixed constant. And second, this argument only makes sense if we stop AdaBoost after a relatively small number of rounds since it is through early stopping, according to this view, that regularization is actually applied.

What happens if AdaBoost is run for a large number of rounds, as in the cases described in Section 3 where overfitting was apparently absent? According to this view, making the number of rounds T large corresponds to choosing a regularization parameter B that is also large. Thus, when T is very large, the purported regularization must be extremely weak, and in the limit, must become so vanishingly weak as to apparently have no constraining influence at all on the optimization problem that it is meant to constrain. When this happens, how can it be having any effect at all?

In fact, Rosset, Zhu and Hastie [29] proved that if the regularization is relaxed to the limit so that $B \to \infty$, then the resulting (anemically regularized) solutions turn out asymptotically to maximize the margins of the training examples. This means that we can prove something about how well such solutions will perform on new data, but only as a result of their margin-maximizing properties and by applying the margins theory. It is not the regularization that is explaining good performance here since it has been weakened to the point of essentially disappearing altogether.

So to summarize, we have seen a perspective in which boosting with early stopping can be related to ℓ_1-regularization. However, this view does not apply to AdaBoost, but only to a variant. And furthermore, for a large number of rounds, we can only explain good performance, according to this view, by again appealing to the margins theory rather than as a direct result of implicit regularization.

6 Inherently Unpredictable Data

As discussed in Section 3, the margins theory shows that, if given "enough" data, and if the weak learning condition holds, then the generalization error can be made arbitrarily close to zero so that the resulting classifier is essentially perfect. This obviously seems like a good thing. But it should also make us suspicious since, even under the most ideal circumstances, it is usually impossible on real data to get perfect accuracy due to intrinsic noise or uncertainty. In other words, the *Bayes error*, the minimum possible error of any classifier, is usually strictly positive.

So on the one hand, the margins theory tells us that, with enough data, it should be possible to train a perfectly accurate classifier, but on the other hand, the data itself usually makes this impossible. In practice, this is not necessarily a contradiction, even when the weak learning assumption holds. This is because the weak hypothesis space typically is not fixed, but grows in complexity with the size of the training set; for instance, this happens "automatically" when using decision trees as weak hypotheses since the generated trees will usually be bigger if trained with more data. Nevertheless, it would certainly be desirable to have a theory that more directly handles the case in which the Bayes error is nonzero.

Indeed, it has been proved that AdaBoost's combined classifier has an error rate that converges to the Bayes optimal provided that the algorithm is given enough data, that it is run for enough but not too many rounds, and that the weak hypotheses come from a class of functions that is "sufficiently rich." In this sense, the algorithm is said to be *universally consistent*, a property that was proved by Bartlett and Traskin [1] following the work of many others [3, 5, 14, 19, 21, 38, 39].

This means that AdaBoost can (theoretically) learn optimally even in noisy settings. Furthermore, this theory does not depend on the weak learning condition. However, the theory does not explain why AdaBoost can often work even when run for a very large number of rounds since, like all explanations other than the margins theory, it depends on the algorithm being stopped after a finite and relatively small number of rounds. Furthermore, the assumption of sufficient richness among the weak hypotheses can also be problematic.

Regarding this last point, Long and Servedio [18] presented an example of a learning problem which shows just how far off a universally consistent algorithm like AdaBoost can be from optimal when this assumption does not hold, even when the noise affecting the data is seemingly very mild. In this example, each data point has its label inverted with quite low probability, say 1%. The Bayes optimal classifier has an error rate that is also just 1%, and is obtainable by a classifier of the same form used by AdaBoost. Nevertheless, AdaBoost, in this case, will provably produce a classifier whose error exceeds 50%, in other words, at least as bad as random guessing. In fact, this result holds even if the learning algorithm is provided with unlimited training data. And it is really not a result about AdaBoost at all — it is really about algorithms based on loss minimization. The same result applies to any method that minimizes exponential loss, as well as most other commonly used convex losses. It also holds even if regularization is applied. For instance, it can be

dataset	noise	AdaBoost	BrownBoost
letter	0%	3.7	4.2
	10%	10.8	7.0
	20%	15.7	10.5
satimage	0%	4.9	5.2
	10%	12.1	6.2
	20%	21.3	7.4

Table 2 The results of running AdaBoost and BrownBoost on the "letter" and "satimage" benchmark datasets. After converting to binary by combining the classes into two arbitrary groups, each dataset was split randomly into training and test sets, and corrupted for training with artificial noise at the given rates. The entries of the table show percent error on *uncorrupted* test examples. All results are averaged over 50 random repetitions of the experiment. (These experiments were conducted by Evan Ettinger, Sunsern Cheamanunkul and Yoav Freund, and were reported in [30].)

shown that the same result applies to support-vector machines, logistic regression, linear regression, lasso, ridge regression, and many more.

So this example shows that such consistency results can fail badly if the weak classifiers are not rich enough. It also shows that AdaBoost (and most other loss-based methods) can be very susceptible to noise, even with regularization, at least on artificially constructed datasets. This susceptibility to noise has also been observed in practice, for instance, by Dietterich [6], and Maclin and Opitz [20].

How then should we handle noise and outliers? Certainly, these must be a problem on "real-world" datasets, and yet, AdaBoost often works well anyway. So one approach is simply not to worry about it. Theoretically, various approaches to handling noise in boosting have also been proposed, often using techniques based on "branching programs" [15, 16, 17].

Yet another approach is based on an entirely different boosting algorithm called *boost-by-majority*, due to Freund [8]. In a certain sense, this algorithm turns out to be exactly optimally efficient as a boosting algorithm. Furthermore, it does not appear to minimize any convex loss function. Like AdaBoost, the algorithm on each round puts more weight on the harder examples. However, unlike AdaBoost, it has a very interesting behavior in which it can "give up" on the very hard examples. This property might make the algorithm more robust to noise by eventually ignoring outliers and noise-corrupted examples rather than "spinning its wheels" on them as AdaBoost does. Unfortunately, unlike AdaBoost, the boost-by-majority algorithm is not *adaptive* in the sense that it requires prior knowledge about the number of rounds and the degree to which the weak learning assumption holds. Nevertheless, Freund [9] proposed making it adaptive by passing to a kind of limit in which time is moving continuously rather than in discrete steps.

The resulting algorithm, called *BrownBoost*, is somewhat more challenging to implement, but preliminary experiments suggest that it might be more resistent to noise and outliers. See Table 2.

Summarizing, we have seen that, under appropriate conditions, AdaBoost provably converges in its accuracy to the best possible, even in the presence of noise and even without the weak learning condition. On the other hand, AdaBoost's per-

formance can be very poor when the weak hypotheses are insufficiently expressive. Noise can be a real problem for AdaBoost, and various approaches have been proposed for handling it, including a form of boosting which operates in continuous time.

7 Conclusions

This chapter has attempted to bring together several different approaches that have been proposed for understanding AdaBoost. These approaches are reflective of broader trends within machine learning, including the rise of methods based on margin maximization, loss minimization, and regularization. As we have seen, these different approaches are based on varying assumptions, and attempt to capture different aspects of AdaBoost's behavior. As such, one can argue as to which of these is most realistic or explanatory, a perspective that is likely to depend on individual taste and experience. Furthermore, direct experimental comparison of the different approaches is especially difficult due to the looseness of the various bounds and theoretical predictions when applied to actual data.

For the most part, the different perspectives that have been presented do not subsume one another, each having something to say about AdaBoost that is perhaps not captured by the others. But taken together, they form a rich and expansive theory for understanding this one algorithm. Perhaps someday a single overriding theory will emerge that encompasses all of the others.

Acknowledgements. Support for this research was generously provided by NSF under award #1016029.

References

1. Bartlett, P.L., Traskin, M.: AdaBoost is consistent. Journal of Machine Learning Research **8**, 2347–2368 (2007)
2. Baum, E.B., Haussler, D.: What size net gives valid generalization? Neural Computation **1**(1), 151–160 (1989)
3. Bickel, P.J., Ritov, Y., Zakai, A.: Some theory for generalized boosting algorithms. Journal of Machine Learning Research **7**, 705–732 (2006)
4. Breiman, L.: Prediction games and arcing classifiers. Neural Computation **11**(7), 1493–1517 (1999)
5. Breiman, L.: Population theory for boosting ensembles. Annals of Statistics **32**(1), 1–11 (2004)
6. Dietterich, T.G.: An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine Learning **40**(2), 139–158 (2000)
7. Frean, M., Downs, T.: A simple cost function for boosting. Tech. rep., Department of Computer Science and Electrical Engineering, University of Queensland (1998)

8. Freund, Y.: Boosting a weak learning algorithm by majority. Information and Computation **121**(2), 256–285 (1995)
9. Freund, Y.: An adaptive version of the boost by majority algorithm. Machine Learning **43**(3), 293–318 (2001)
10. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences **55**(1), 119–139 (1997)
11. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: A statistical view of boosting. Annals of Statistics **28**(2), 337–407 (2000)
12. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics **29**(5) (2001)
13. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, second edn. Springer (2009)
14. Jiang, W.: Process consistency for AdaBoost. Annals of Statistics **32**(1), 13–29 (2004)
15. Kalai, A.T., Servedio, R.A.: Boosting in the presence of noise. Journal of Computer and System Sciences **71**(3), 266–290 (2005)
16. Long, P.M., Servedio, R.A.: Martingale boosting. In: 18th Annual Conference on Learning Theory (2005)
17. Long, P.M., Servedio, R.A.: Adaptive martingale boosting. In: Advances in Neural Information Processing Systems 21 (2009)
18. Long, P.M., Servedio, R.A.: Random classification noise defeats all convex potential boosters. Machine Learning **78**, 287–304 (2010)
19. Lugosi, G., Vayatis, N.: On the Bayes-risk consistency of regularized boosting methods. Annals of Statistics **32**(1), 30–55 (2004)
20. Maclin, R., Opitz, D.: An empirical evaluation of bagging and boosting. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence, pp. 546–551 (1997)
21. Mannor, S., Meir, R., Zhang, T.: Greedy algorithms for classification — consistency, convergence rates, and adaptivity. Journal of Machine Learning Research **4**, 713–742 (2003)
22. Mason, L., Bartlett, P., Baxter, J.: Direct optimization of margins improves generalization in combined classifiers. In: Advances in Neural Information Processing Systems 12 (2000)
23. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Functional gradient techniques for combining hypotheses. In: Advances in Large Margin Classifiers. MIT Press (2000)
24. Mease, D., Wyner, A.: Evidence contrary to the statistical view of boosting. Journal of Machine Learning Research **9**, 131–156 (2008)
25. Onoda, T., Rätsch, G., Müller, K.R.: An asymptotic analysis of AdaBoost in the binary classification case. In: Proceedings of the 8th International Conference on Artificial Neural Networks, pp. 195–200 (1998)
26. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
27. Rätsch, G., Onoda, T., Müller, K.R.: Soft margins for AdaBoost. Machine Learning **42**(3), 287–320 (2001)
28. Reyzin, L., Schapire, R.E.: How boosting the margin can also boost classifier complexity. In: Proceedings of the 23rd International Conference on Machine Learning (2006)
29. Rosset, S., Zhu, J., Hastie, T.: Boosting as a regularized path to a maximum margin classifier. Journal of Machine Learning Research **5**, 941–973 (2004)
30. Schapire, R.E., Freund, Y.: Boosting: Foundations and Algorithms. MIT Press (2012)
31. Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S.: Boosting the margin: A new explanation for the effectiveness of voting methods. Annals of Statistics **26**(5), 1651–1686 (1998)
32. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Machine Learning **37**(3), 297–336 (1999)
33. Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B (Methodological) **58**(1), 267–288 (1996)
34. Vapnik, V.N., Chervonenkis, A.Y.: On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and its Applications **XVI**(2), 264–280 (1971)
35. Vapnik, V.N., Chervonenkis, A.Y.: Theory of Pattern Recognition. Nauka (1974). (In Russian)

36. Wang, L., Sugiyama, M., Jing, Z., Yang, C., Zhou, Z.H., Feng, J.: A refined margin analysis for boosting algorithms via equilibrium margin. Journal of Machine Learning Research **12**, 1835–1863 (2011)
37. Wyner, A.J.: On boosting and the exponential loss. In: Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (2003)
38. Zhang, T.: Statistical behavior and consistency of classification methods based on convex risk minimization. Annals of Statistics **32**(1), 56–134 (2004)
39. Zhang, T., Yu, B.: Boosting with early stopping: Convergence and consistency. Annals of Statistics **33**(4), 1538–1579 (2005)
40. Zhao, P., Yu, B.: Stagewise Lasso. Journal of Machine Learning Research **8**, 2701–2726 (2007)