Journal of Japanese Society for Artificial Intelligence, 14(5):771-780, September, 1999.
(In Japanese, translation by Naoki Abe.)

A Short Introduction to Boosting

Yoav Freund Robert E. Schapire
AT&T Labs – Research
Shannon Laboratory
180 Park Avenue
Florham Park, NJ 07932 USA
www.research.att.com/∼{yoav, schapire}
{yoav, schapire}@research.att.com

Abstract

Boosting is a general method for improving the accuracy of any given learning algorithm. This short overview paper introduces the boosting algorithm AdaBoost, and explains the underlying theory of boosting, including an explanation of why boosting often does not suffer from overfitting as well as boosting's relationship to support-vector machines. Some examples of recent applications of boosting are also described.

Introduction

A horse-racing gambler, hoping to maximize his winnings, decides to create a computer program that will accurately predict the winner of a horse race based on the usual information (number of races recently won by each horse, betting odds for each horse, etc.). To create such a program, he asks a highly successful expert gambler to explain his betting strategy. Not surprisingly, the expert is unable to articulate a grand set of rules for selecting a horse. On the other hand, when presented with the data for a specific set of races, the expert has no trouble coming up with a "rule of thumb" for that set of races (such as, "Bet on the horse that has recently won the most races" or "Bet on the horse with the most favored odds"). Although such a rule of thumb, by itself, is obviously very rough and inaccurate, it is not unreasonable to expect it to provide predictions that are at least a little bit better than random guessing. Furthermore, by repeatedly asking the expert's opinion on different collections of races, the gambler is able to extract many rules of thumb.

In order to use these rules of thumb to maximum advantage, there are two problems faced by the gambler: First, how should he choose the collections of races presented to the expert so as to extract rules of thumb from the expert that will be the most useful? Second, once he has collected many rules of thumb, how can they be combined into a single, highly accurate prediction rule?

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 short paper overviews some of the recent work on boosting, focusing especially on the AdaBoost algorithm which has undergone intense theoretical study and empirical testing. After introducing AdaBoost, we describe some of the basic underlying theory of boosting, including an explanation of why it often tends not to overfit. We also describe some experiments and applications using boosting.

Background

Boosting has its roots in a theoretical framework for studying machine learning called the "PAC" learning model, due to Valiant [46]; see Kearns and Vazirani [32] for a good introduction to this model. Kearns and Valiant [30, 31] were the first to pose the question of whether a "weak" learning algorithm which performs just slightly better than random guessing in the PAC model can be "boosted" into an arbitrarily accurate "strong" learning algorithm. Schapire [38] came up with the first provable polynomial-time boosting algorithm in 1989. A year later, Freund [17] developed a much more efficient boosting algorithm which, although optimal in a certain sense, nevertheless suffered from certain practical drawbacks. The first experiments with these early boosting algorithms were carried out by Drucker, Schapire and Simard [16] on an OCR task.

AdaBoost

The AdaBoost algorithm, introduced in 1995 by Freund and Schapire [23], solved many of the practical difficulties of the earlier boosting algorithms, and is the focus of this paper. Pseudocode for AdaBoost is given in Fig. 1. The algorithm takes as input a training set $(x_1, y_1), \ldots, (x_m, y_m)$ where each x_i belongs to some *domain* or *instance space* X, and each *label* y_i is in some label set Y. For most of this paper, we assume $Y = \{-1, +1\}$; later, we discuss extensions to the multiclass case. AdaBoost calls a given *weak* or *base learning algorithm* repeatedly in a series of rounds $t = 1, \ldots, T$. 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 $D_t(i)$. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set.

The weak learner's job is to find a *weak hypothesis* $h_t : X \to \{-1, +1\}$ appropriate for the distribution D_t. The goodness of a weak hypothesis is measured by its *error*

$$\epsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right] = \sum_{i: h_t(x_i) \neq y_i} D_t(i).$$

Notice that the error is measured with respect to the distribution D_t on which the weak learner was trained. In practice, the weak learner may be an algorithm that can use the weights D_t on the training examples. Alternatively, when this is not possible, a subset of the training examples can be sampled according to D_t, and these (unweighted) resampled examples can be used to train the weak learner.

Relating back to the horse-racing example, the instances x_i correspond to descriptions of horse races (such as which horses are running, what are the odds, the track records of each horse, etc.)

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

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

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

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$.
- Update:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

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

Output the final hypothesis:

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

Figure 1: The boosting algorithm AdaBoost.

---

and the labels y_i give the outcomes (i.e., the winners) of each race. The weak hypotheses are the rules of thumb provided by the expert gambler where the subcollections that he examines are chosen according to the distribution D_t.

Once the weak hypothesis h_t has been received, AdaBoost chooses a parameter α_t as in the figure. Intuitively, α_t measures the importance that is assigned to h_t. Note that $\alpha_t \geq 0$ if $\epsilon_t \leq 1/2$ (which we can assume without loss of generality), and that α_t gets larger as ϵ_t gets smaller.

The distribution D_t is next updated using the rule shown in the figure. The effect of this rule is to increase the weight of examples misclassified by h_t, and to decrease the weight of correctly classified examples. Thus, the weight tends to concentrate on "hard" examples.

The *final hypothesis* H is a weighted majority vote of the T weak hypotheses where α_t is the weight assigned to h_t.

Schapire and Singer [42] show how AdaBoost and its analysis can be extended to handle weak hypotheses which output real-valued or *confidence-rated* predictions. That is, for each instance x, the weak hypothesis h_t outputs a prediction $h_t(x) \in \mathbb{R}$ whose sign is the predicted label (-1 or $+1$) and whose magnitude $|h_t(x)|$ gives a measure of "confidence" in the prediction. In this paper, however, we focus only on the case of binary ($\{-1, +1\}$) valued weak-hypothesis predictions.

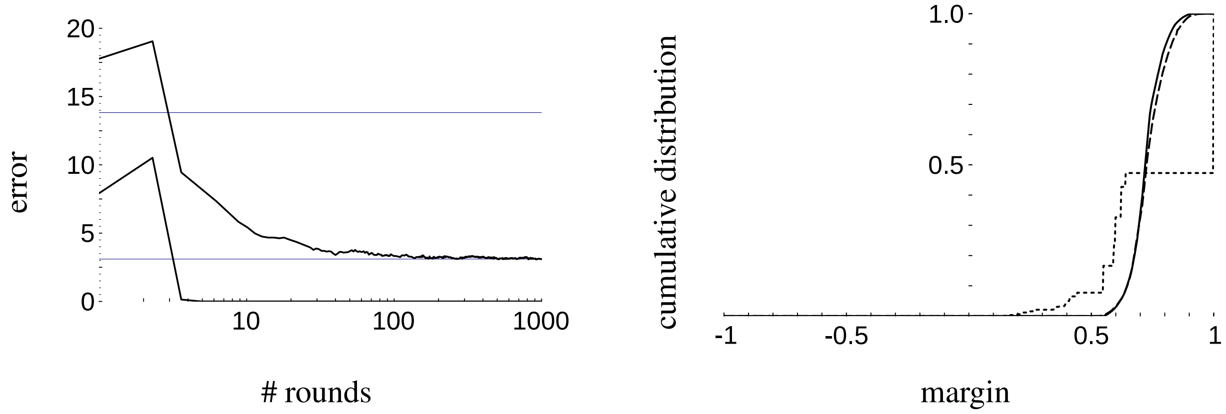

Figure 2: Error curves and the margin distribution graph for boosting C4.5 on the letter dataset as reported by Schapire et al. [41]. *Left*: the training and test error curves (lower and upper curves, respectively) of the combined classifier as a function of the number of rounds of boosting. The horizontal 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.

Analyzing the training error

The most basic theoretical property of AdaBoost concerns its ability to reduce the training error. Let us write the error ϵ_t of h_t as $\frac{1}{2} - \gamma_t$. Since a hypothesis that guesses each instance's class at random has an error rate of $1/2$ (on binary problems), γ_t thus measures how much better than random are h_t's predictions. Freund and Schapire [23] prove that the training error (the fraction of mistakes on the training set) of the final hypothesis H is at most

$$\prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \le \exp\left(-2\sum_t \gamma_t^2\right). \tag{1}$$

Thus, if each weak hypothesis is slightly better than random so that $\gamma_t \ge \gamma$ for some $\gamma > 0$, then the training error drops exponentially fast.

A similar property is enjoyed by previous boosting algorithms. However, previous algorithms required that such a lower bound γ be known a priori before boosting begins. In practice, knowledge of such a bound is very difficult to obtain. AdaBoost, on the other hand, is *adaptive* in that it adapts to the error rates of the individual weak hypotheses. This is the basis of its name — "Ada" is short for "adaptive."

The bound given in Eq. (1), 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 weak learning algorithm (which can always generate a hypothesis with a weak edge for any distribution) into a strong learning algorithm (which can generate a hypothesis with an arbitrarily low error rate, given sufficient data).

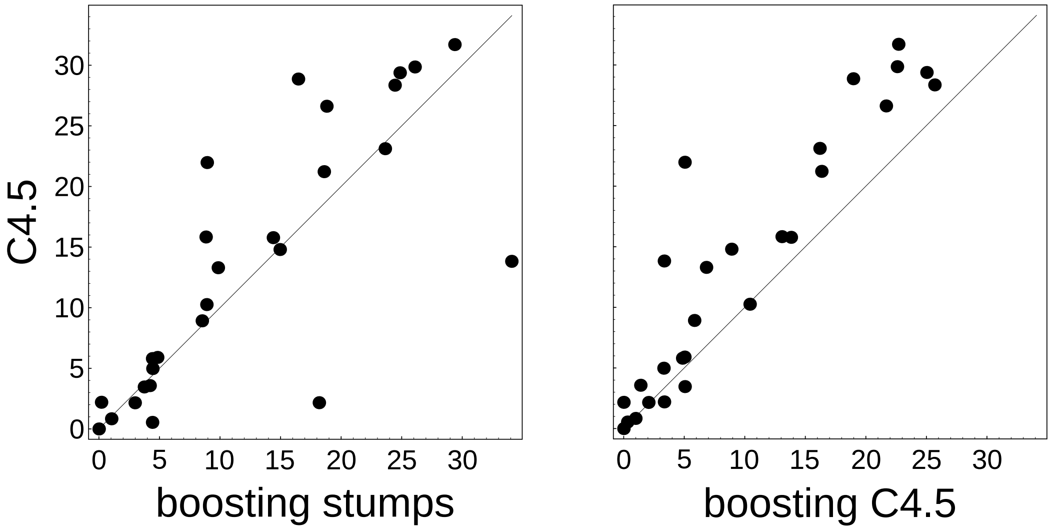

Figure 3: Comparison of C4.5 versus boosting stumps and boosting C4.5 on a set of 27 benchmark problems as reported by Freund and Schapire [21]. Each point in each scatterplot shows the test error rate of the two competing algorithms on a single benchmark. The y-coordinate of each point gives the test 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.

Generalization error

Freund and Schapire [23] showed how to bound the generalization error of the final hypothesis in terms of its training error, the sample size m, the VC-dimension d of the weak hypothesis space and the number of boosting rounds T. (The VC-dimension is a standard measure of the "complexity" of a space of hypotheses. See, for instance, Blumer et al. [5].) Specifically, they used techniques from Baum and Haussler [4] to show that the generalization error, with high probability, is at most

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

where $\hat{\Pr}\left[\cdot\right]$ 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 [9, 15, 36] 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. 2 shows the training and test curves of running boosting on top of Quinlan's C4.5 decision-tree learning algorithm [37] on the "letter" dataset.

In response to these empirical findings, Schapire et al. [41], following the work of Bartlett [2], gave an alternative analysis in terms of the *margins* of the training examples. The margin of

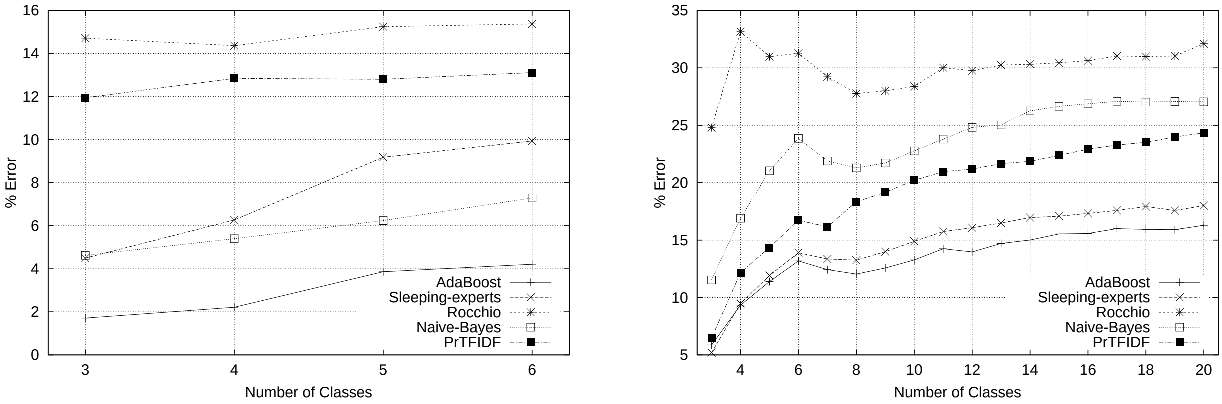

Figure 4: Comparison of error rates for AdaBoost and four other text categorization methods (naive Bayes, probabilistic TF-IDF, Rocchio and sleeping experts) as reported by Schapire and Singer [43]. The algorithms were tested on two text corpora — Reuters newswire articles (left) and AP newswire headlines (right) — and with varying numbers of class labels as indicated on the x-axis of each figure.

---

example (x, y) is defined to be

$$\frac{y \sum_t \alpha_t h_t(x)}{\sum_t \alpha_t}. \tag{2}$$

It is a number in $[-1, +1]$ which is positive if and only if H correctly classifies the example. Moreover, 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 error is at most

$$\hat{\Pr}\left[\text{margin}(x, y) \leq \theta\right] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right) \tag{3}$$

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. 2 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.

Attempts (not always successful) to use the insights gleaned from the theory of margins have been made by several authors [7, 27, 34].

The behavior of AdaBoost can also be understood in a game-theoretic setting as explored by Freund and Schapire [22, 24] (see also Grove and Schuurmans [27] and Breiman [8]). In particular, boosting can be viewed as repeated play of a certain game, and AdaBoost can be shown to be a special case of a more general algorithm for playing repeated games and for approximately solving a game. This also shows that boosting is closely related to linear programming and online learning.

Relation to support-vector machines

The margin theory points to a strong connection between boosting and the support-vector machines of Vapnik and others [6, 12, 47]. To clarify the connection, suppose that we have already found the weak hypotheses that we want to combine and are only interested in choosing the coefficients α_t. One reasonable approach suggested by the analysis of AdaBoost's generalization error is to choose the coefficients so that the bound given in Eq. (3) is minimized. In particular, suppose that the first term is zero and let us concentrate on the second term so that we are effectively attempting to maximize the *minimum* margin of any training example.[1] To make this idea precise, let us denote the vector of weak-hypothesis predictions associated with the example (x, y) by $\mathbf{h}(x) \doteq \langle h_1(x), h_2(x), \ldots, h_N(x) \rangle$ which we call the *instance vector* and the vector of coefficients by $\boldsymbol{\alpha} \doteq \langle \alpha_1, \alpha_2, \ldots, \alpha_N \rangle$ which we call the *weight vector*. Using this notation and the definition of margin given in Eq. (2) we can write the goal of maximizing the minimum margin as

$$\max_{\boldsymbol{\alpha}} \min_i \frac{(\boldsymbol{\alpha} \cdot \mathbf{h}(x_i)) y_i}{||\boldsymbol{\alpha}|| \; ||\mathbf{h}(x_i)||} \tag{4}$$

where, for boosting, the norms in the denominator are defined as:

$$||\boldsymbol{\alpha}||_1 \doteq \sum_t |\alpha_t|, \qquad ||\mathbf{h}(x)||_\infty \doteq \max_t |h_t(x)| \, .$$

(When the h_t's all have range $\{-1, +1\}$, $||\mathbf{h}(x)||_\infty$ is simply equal to 1.)

In comparison, the explicit goal of support-vector machines is to maximize a minimal margin of the form described in Eq. (4), but where the norms are instead Euclidean:

$$||\boldsymbol{\alpha}||_2 \doteq \sqrt{\sum_t \alpha_t^2}, \qquad ||\mathbf{h}(x)||_2 \doteq \sqrt{\sum_t h_t(x)^2} \, .$$

Thus, SVM's use the ℓ_2 norm for both the instance vector and the weight vector, while AdaBoost uses the ℓ_∞ norm for the instance vector and ℓ_1 norm for the weight vector.

When described in this manner, SVM and AdaBoost seem very similar. However, there are several important differences:

- **Different norms can result in very different margins.** The difference between the norms ℓ_1, ℓ_2 and ℓ_∞ may not be very significant when one considers low dimensional spaces. However, in boosting or in SVM, the dimension is usually very high, often in the millions or more. In such a case, the difference between the norms can result in very large differences

[1] Of course, AdaBoost does not explicitly attempt to maximize the minimal margin. Nevertheless, Schapire et al.'s [41] analysis suggests that the algorithm does try to make the margins of all the training examples as large as possible, so in this sense, we can regard this maximum minimal margin algorithm as an illustrative approximation of AdaBoost. In fact, algorithms that explicitly attempt to maximize minimal margin have not been experimentally as successful as AdaBoost [7, 27].

in the margin values. This seems to be especially so when there are only a few relevant variables so that $\boldsymbol{\alpha}$ can be very sparse. For instance, suppose the weak hypotheses all have range $\{-1, +1\}$ and that the label y on all examples can be computed by a majority vote of k of the weak hypotheses. In this case, it can be shown that if the number of relevant weak hypotheses k is a small fraction of the total number of weak hypotheses then the margin associated with AdaBoost will be much larger than the one associated with support vector machines.

- **The computation requirements are different.** The computation involved in maximizing the margin is mathematical programming, i.e., maximizing a mathematical expression given a set of inequalities. The difference between the two methods in this regard is that SVM corresponds to *quadratic programming*, while AdaBoost corresponds only to *linear programming*. (In fact, as noted above, there is a deep relationship between AdaBoost and linear programming which also connects AdaBoost with game theory and online learning [22].)

- **A different approach is used to search efficiently in high dimensional space.** Quadratic programming is more computationally demanding than linear programming. However, there is a much more important computational difference between SVM and boosting algorithms. Part of the reason for the effectiveness of SVM and AdaBoost is that they find linear classifiers for extremely high dimensional spaces, sometimes spaces of infinite dimension. While the problem of overfitting is addressed by maximizing the margin, the computational problem associated with operating in high dimensional spaces remains. Support vector machines deal with this problem through the method of *kernels* which allow algorithms to perform low dimensional calculations that are mathematically equivalent to inner products in a high dimensional "virtual" space. The boosting approach is instead to employ *greedy search*: from this perspective, the weak learner is an oracle for finding coordinates of $\mathbf{h}(x)$ that have a non-negligible correlation with the label y. The reweighting of the examples changes the distribution with respect to which the correlation is measured, thus guiding the weak learner to find different correlated coordinates. Most of the actual work involved in applying SVM or AdaBoost to specific classification problems has to do with selecting the appropriate kernel function in the one case and weak learning algorithm in the other. As kernels and weak learning algorithms are very different, the resulting learning algorithms usually operate in very different spaces and the classifiers that they generate are extremely different.

Multiclass classification

So far, we have only considered binary classification problems in which the goal is to distinguish between only two possible classes. Many (perhaps most) real-world learning problems, however, are *multiclass* with more than two possible classes. There are several methods of extending AdaBoost to the multiclass case.

The most straightforward generalization [23], called AdaBoost.M1, is adequate when the weak learner is strong enough to achieve reasonably high accuracy, even on the hard distributions created by AdaBoost. However, this method fails if the weak learner cannot achieve at least 50% accuracy when run on these hard distributions.

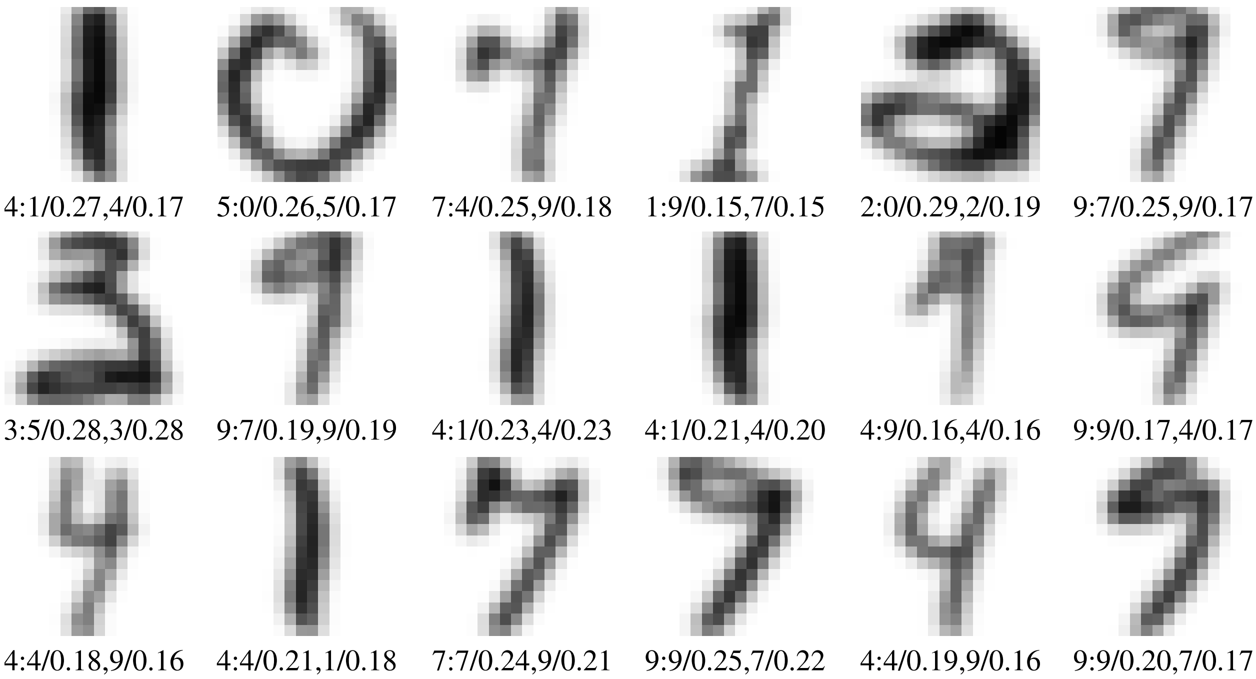

Figure 5: A sample of the examples that have the largest weight on an OCR task as reported by Freund and Schapire [21]. These examples were chosen after 4 rounds of boosting (top line), 12 rounds (middle) and 25 rounds (bottom). Underneath each image is a line of the form $d{:}\ell_1/w_1{,}\ell_2/w_2$, where d is the label of the example, ℓ_1 and ℓ_2 are the labels that get the highest and second highest vote from the combined hypothesis at that point in the run of the algorithm, and w_1, w_2 are the corresponding normalized scores.

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 [42] 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 [23] algorithm AdaBoost.M2 (which is a special case of Schapire and Singer's [42] 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 weak learning algorithm. A different technique [39], which incorporates Dietterich and Bakiri's [14] method of error-correcting output codes, achieves similar provable bounds to those of AdaBoost.MH and AdaBoost.M2, but can be used with any weak learner which can handle simple, binary labeled data. Schapire and Singer [42] give yet another method of combining boosting with error-correcting output codes.

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 weak learner and so can be flexibly combined with *any* method for finding weak hypotheses. Finally, it comes with a set of theoretical guarantees given sufficient data and a weak learner that can reliably provide only moderately accurate weak hypotheses. 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 weak 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 weak learner. Consistent with theory, boosting can fail to perform well given insufficient data, overly complex weak hypotheses or weak hypotheses which are too weak. Boosting seems to be especially susceptible to noise [13] (more on this later).

AdaBoost has been tested empirically by many researchers, including [3, 13, 15, 29, 33, 36, 45]. For instance, Freund and Schapire [21] tested AdaBoost on a set of UCI benchmark datasets [35] using C4.5 [37] as a weak learning algorithm, as well as an algorithm which finds the best "decision stump" or single-test decision tree. Some of the results of these experiments are shown in Fig. 3. As can be seen from this figure, even boosting the weak decision stumps can usually give as good results as C4.5, while boosting C4.5 generally gives the decision-tree algorithm a significant improvement in performance.

In another set of experiments, Schapire and Singer [43] used boosting for text categorization tasks. For this work, weak hypotheses were used which 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. 4. 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. Boosting has also been applied to text filtering [44], "ranking" problems [19] and classification problems arising in natural language processing [1, 28].

The generalization of AdaBoost by Schapire and Singer [42] provides an interpretation of boosting as a gradient-descent method. A potential function is used in their algorithm to associate a cost with each example based on its current margin. Using this potential function, the operation of AdaBoost can be interpreted as a coordinate-wise gradient descent in the space of linear classifiers (over weak hypotheses). Based on this insight, one can design algorithms for learning popular classification rules. In recent work, Cohen and Singer [11] showed how to apply boosting to learn rule lists similar to those generated by systems like RIPPER [10], IREP [26] and C4.5rules [37]. In other work, Freund and Mason [20] showed how to apply boosting to learn a generalization of decision trees called "alternating trees."

A nice property of AdaBoost is its ability to identify *outliers*, i.e., examples that are either mislabeled in the training data, or which 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. 5 taken from an OCR experiment conducted by Freund and Schapire [21].

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 [13]. Friedman et al. [25] suggested a variant of AdaBoost, called "Gentle AdaBoost" which puts less emphasis on outliers. In recent work, Freund [18] suggested another algorithm, called "BrownBoost," which takes a more radical approach that de-emphasizes outliers when it seems clear that they are "too hard" to classify correctly. This algorithm is an adaptive version of Freund's [17] "boost-by-majority" algorithm. This work, together with Schapire's [40] work on "drifting games," reveal some interesting new relationships between boosting, Brownian motion, and repeated games while raising many new open problems and directions for future research.

References

[1] Steven Abney, Robert E. Schapire, and Yoram Singer. Boosting applied to tagging and PP attachment. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[2] Peter L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, March 1998.

[3] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, to appear.

[4] Eric B. Baum and David Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[5] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[6] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[7] Leo Breiman. Arcing the edge. Technical Report 486, Statistics Department, University of California at Berkeley, 1997.

[8] Leo Breiman. Prediction games and arcing classifiers. Technical Report 504, Statistics Department, University of California at Berkeley, 1997.

[9] Leo Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[10] William Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[11] William W. Cohen and Yoram Singer. A simple, fast, and effective rule learner. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[13] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, to appear.

[14] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.

[15] Harris Drucker and Corinna Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, pages 479–485, 1996.

[16] Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):705–719, 1993.

[17] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[18] Yoav Freund. An adaptive version of the boost by majority algorithm. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, 1999.

[19] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. In *Machine Learning: Proceedings of the Fifteenth International Conference*, 1998.

[20] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Machine Learning: Proceedings of the Sixteenth International Conference*, 1999. to appear.

[21] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[22] Yoav Freund and Robert E. Schapire. Game theory, on-line prediction and boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 325–332, 1996.

[23] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[24] Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, to appear.

[25] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. Technical Report, 1998.

[26] Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 70–77, 1994.

[27] Adam J. Grove and Dale Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[28] Masahiko Haruno, Satoshi Shirai, and Yoshifumi Ooyama. Using decision trees to construct a practical parser. *Machine Learning*, 34:131–149, 1999.

[29] Jeffrey C. Jackson and Mark W. Craven. Learning sparse perceptrons. In *Advances in Neural Information Processing Systems 8*, pages 654–660, 1996.

[30] Michael Kearns and Leslie G. Valiant. Learning Boolean formulae or finite automata is as hard as factoring. Technical Report TR-14-88, Harvard University Aiken Computation Laboratory, August 1988.

[31] Michael Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the Association for Computing Machinery*, 41(1):67–95, January 1994.

[32] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[33] Richard Maclin and David Opitz. An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 546–551, 1997.

[34] Llew Mason, Peter Bartlett, and Jonathan Baxter. Direct optimization of margins improves generalization in combined classifiers. Technical report, Deparment of Systems Engineering, Australian National University, 1998.

[35] C. J. Merz and P. M. Murphy. UCI repository of machine learning databases, 1998. www.ics.uci.edu/$\sim$mlearn/MLRepository.html.

[36] J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730, 1996.

[37] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[38] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[39] Robert E. Schapire. Using output codes to boost multiclass learning problems. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 313–321, 1997.

[40] Robert E. Schapire. Drifting games. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, 1999.

[41] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, October 1998.

[42] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91, 1998. To appear, *Machine Learning*.

[43] Robert E. Schapire and Yoram Singer. BoosTexter: A boosting-based system for text categorization. *Machine Learning*, to appear.

[44] Robert E. Schapire, Yoram Singer, and Amit Singhal. Boosting and Rocchio applied to text filtering. In *SIGIR '98: Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval*, 1998.

[45] Holger Schwenk and Yoshua Bengio. Training methods for adaptive boosting of neural networks. In *Advances in Neural Information Processing Systems 10*, pages 647–653, 1998.

[46] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[47] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.