Precision-based Boosting

Mohammad Hossein Nikravan, Marjan Movahedan, Sandra Zilles
Department of Computer Science, University of Regina, Regina, SK, Canada
nikravam@uregina.ca, marjan.movahedan@gmail.com, zilles@cs.uregina.ca

Abstract

AdaBoost is a highly popular ensemble classification method for which many variants have been published. This paper proposes a *generic refinement* of all of these AdaBoost variants. Instead of assigning weights based on the total error of the base classifiers (as in AdaBoost), our method uses class-specific error rates. On instance x it assigns a higher weight to a classifier predicting label y on x, if that classifier is less likely to make a mistake *when it predicts class* y. Like AdaBoost, our method is guaranteed to boost weak learners into strong learners. An empirical study on AdaBoost and one of its multi-class versions, SAMME, demonstrates the superiority of our method on datasets with more than 1,000 instances as well as on datasets with more than three classes.

Introduction

One of the most popular methods for ensemble classification is AdaBoost, introduced by Freund and Schapire (1997). The intuitive idea behind AdaBoost is: (i) train a simple base classifier h_1 and observe its performance on the training data; (ii) assign a higher weight to the training examples misclassified by h_1 and a lower weight to the correctly classified ones (so that the base classifier h_2 to be trained in the next iteration focuses more on those data points that h_1 misclassified); (iii) iterate this procedure T times, to create base classifiers $h_1, \ldots, h_T$; and (iv) form a final classifier from a weighted majority vote of the base classifiers.

If base classifier h_t has a lower weighted error on the training examples than base classifier $h_{t'}$, then h_t is assigned more weight in the final vote than $h_{t'}$. Moreover, when reweighting training examples after iteration t, weights are changed more substantially than after iteration t', because the examples misclassified by the stronger classifier h_t likely tend to be "more difficult" than many of those misclassified by the weaker classifier $h_{t'}$. The exact coefficients by which to reweight training examples between iterations and by which to weight base classifiers in the final vote were chosen so as to yield guarantees on the performance of AdaBoost.

In this paper, we follow an idea originally proposed by Aslam (2000) and refine the principle behind AdaBoost's weighting scheme. Consider a hypothetical situation in which several base classifiers are to be aggregated, based on their performance on a set of 100 training examples, of which 50 have class 1 and 50 have class 2. Classifier h_t predicts correct labels for 49 of the 50 instances in class 1, and for 25 of the 50 instances in class 2, for a total error of 26%. Classifier $h_{t'}$ predicts correct labels for 40 of the 50 instances in class 1, and for 40 of the 50 instances in class 2. Its total error is 20%. When aggregating all base classifiers to make a prediction on an unseen instance x, AdaBoost will assign more weight to $h_{t'}$, which appears more "trustworthy", given its lower error. The refined approach that we suggest though will first evaluate both h_t and $h_{t'}$ on x and then decide how to weight their votes. Suppose $h_t(x) = 2$ and $h_{t'}(x) = 1$. Our statistics on the training data suggest that, when h_t predicts class 2, it is incorrect in only $1/26 = 3.8\%$ of all cases. By comparison, when $h_{t'}$ predicts class 1, it is incorrect in $10/50 = 20\%$ of all cases. So, the chance of being incorrect *on this specific instance* x is more than 5 times higher when following $h_{t'}$ than when following h_t. Unlike what AdaBoost would do, the method we propose in this paper would assign a higher weight to the vote of h_t than to the vote of $h_{t'}$, when making a prediction on the specific instance x.

In short, our method does not simply assess the total error/accuracy of a base classifier, but its error/precision when it predicts class y, separately for each class y. We use the term *class-specific precision* to refer to the precision of a classifier on any one chosen class label. When handling data instance x, the weight our method assigns to a base classifier h hence increases with h's class-specific precision on the class $y = h(x)$ predicted by h on x. We thus call our method *Precision-based AdaBoost*, or *PrAdaBoost* for short. The same idea can be applied to virtually every existing variation of AdaBoost; e.g., we propose and test *PrSAMME*, the precision-based version of the multi-class AdaBoost adaptation SAMME (Hastie et al. 2009).

Following a forward-stagewise additive model (Friedman, Hastie, and Tibshirani 2000), the choice of weight parameters in AdaBoost is optimal when only a single weight parameter can be assigned to every individual base classifier. In this forward-stagewise model, we simply introduce one more degree of freedom by incorporating class-specific precision, which leads to a different optimal solution. Our formal analysis of PrAdaBoost shows that, as in the case

of AdaBoost, the PrAdaBoost algorithm trains and aggregates weak classifiers (which are only guaranteed to be better than random guessing) into arbitrarily strong (i.e., arbitrarily accurate) classifiers, when evaluated on the training data. Analogously to Freund and Schapire's (1996) result for AdaBoost, we obtain an upper bound on the training error that decreases exponentially quickly as a function of the number of iterations run by our algorithm.

In an empirical evaluation on 23 two-class UCI datasets, neither PrAdaBoost nor AdaBoost consistently outperform its competitor, but our results strongly suggest that PrAdaBoost tends to beat AdaBoost on larger datasets as well as on imbalanced datasets. In the multi-class setting, on 18 UCI datasets, overall PrSAMME is the clear winner over SAMME. In the sum, our evaluation suggests that, especially if the dataset has more than 1,000 data points, is imbalanced, or has more than three classes, precision-based boosting should be preferred over the traditional "error-based" boosting.

Related Work

Various methods for constructing ensemble-based systems have been developed (Rokach 2010). Adaptive boosting (Freund and Schapire 1997), bagging (Breiman 1996), and random forests (Breiman 2001) are some of the best-known ensemble-based classifiers. AdaBoost is among the most popular methods, not only because of its successes in practice, but also because of its solid theoretical foundation.

The theory behind AdaBoost is based on the greedy minimization of an exponential loss function (Schapire and Freund 2012; Friedman, Hastie, and Tibshirani 2000; Breiman 1999; Schapire and Singer 1999). Knowledge of this minimization procedure gives us insights into the convergence properties of the AdaBoost algorithm. In addition, it provides us with a means of modifying the loss function so as to obtain novel boosting-like methods for modified classification tasks (Schapire and Freund 2012). For example, Schapire and Singer (Schapire and Singer 1999) used the Hamming loss to develop a generalized multi-class version of AdaBoost. Their method, AdaBoost MH, minimizes the Hamming loss by decomposing the problem into several two-class classification problems.

Approaches targeting imbalances in the data or the misclassification costs led to the design of asymmetric learning methods (Nikolaou and Brown 2015), such as AdaCost (Fan et al. 1999), which reduces the cumulative misclassification cost by integrating a cost-adjustment function into the weight updating rule of AdaBoost, and of other variants of AdaBoost; see for instance (Wu and Nagahashi 2015) for further references and for a comparative study of various AdaBoost-like methods. The crucial difference between these methods and ours is that ours is not targeted at imbalanced data or imbalanced misclassification costs or any other kind of imbalance between the individual classes in the given data. Instead of targeting asymmetric learning settings, we improve AdaBoost *in its original task* by introducing more freedom in the choice of weight parameters to be selected in minimizing the exponential loss. The precision-based approach is applicable to all variations of AdaBoost – just as we test PrAdaBoost and PrSAMME, one could also implement PrAdaCost, PrAdaBoost.M2, etc.

The only existing method using an idea similar to ours is InfoBoost (Aslam 2000), which was never tested extensively in the literature. Like PrAdaBoost, it uses weight parameters based on class-specific precision, but it is built on an information-theoretic motivation. By contrast, our method employs the forward-stagewise additive model used to justify AdaBoost's weight parameters. In particular, we introduce another degree of freedom into this model, yielding a new optimal solution. We use this new optimal solution in our parameters, thus not only significantly improving AdaBoost but also beating InfoBoost by a large margin.

Algorithm 1: AdaBoost Scheme (Freund and Schapire 1997)

Input: A training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$,
BaseInducer, i.e., a weak learning algorithm,
Number of iterations $T \in \mathbb{N}$.
Initialize $D_1(x_i) = \frac{1}{n}$ for all $i \in \{1, \ldots, n\}$.
For $t = 1$ to T **do**

1. Call BaseInducer with the distribution D_t to obtain a hypothesis $h_t : X \to Y$.
2. Calculate the total error ε_t of h_t w.r.t. D_t, i.e.,

$$\varepsilon_t = \sum_{i=1}^{n} D_t(x_i) \mathbb{I}(h_t(x_i) \neq y_i).$$

   If $\varepsilon_t > 0.5$, then set T to $t-1$ and abort the loop.
3. Set β_t as a decreasing function of ε_t.
4. Redistribute weights as follows, for $1 \leq i \leq n$:

$$D_{t+1}(x_i) = \frac{D_t(x_i)}{Z_t} \times \begin{cases} e^{-\beta_t} & \text{if } h_t(x_i) = y_i \\ e^{\beta_t} & \text{otherwise} \end{cases}$$

   where Z_t is set such that $\sum_{i=1}^{n} D_{t+1}(x_i) = 1$.

end
Output: the final hypothesis

$$H(x) = \begin{cases} 1\,, & \text{if } \sum_{t=1}^{T} \beta_t h_t(x) \geq 0\,, \\ -1\,, & \text{otherwise}\,. \end{cases} \quad (1)$$

AdaBoost Framework

Suppose a training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of labeled examples is given, based on a binary target classifier. Here, each x_i represents a training instance, and $y_i \in \{-1, +1\}$ is the label of x_i.

The algorithmic framework that is used by AdaBoost (Freund and Schapire 1996) is displayed in Algorithm 1. On input of S, the boosting algorithm runs for T iterations, in each iteration t obtaining a new base classifier h_t from a weak learning algorithm. Initially, all training instances in S have the same weight, but in each iteration t, the weights are updated, see line 4. The parameter β_t used in the weight ad-

justment is defined to be a decreasing function of the error ε_t. The latter refers to the cumulative weight of the training instances misclassified by the base classifier h_t, with respect to the distribution D_t, see line 2.[1] After T iterations, the final aggregated classifier computes a weighted majority vote over all previously trained base classifiers, where the weight of classifier h_t equals β_t, i.e., decreases when the error ε_t of h_t increases, see Eqn. (1). In AdaBoost (Freund and Schapire 1996), the crucial parameter β_t is defined as

$$\beta_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t} .$$

AdaBoost follows a forward-stagewise additive model (Friedman, Hastie, and Tibshirani 2000), where the objective is to find a function H which minimizes the exponential loss $L(y, H) = e^{-yH(x)}$ over S, i.e., which minimizes

$$\sum_{i=1}^{n} L(y_i, H(x_i)) = \sum_{i=1}^{n} e^{-y_i H(x_i)} . \tag{2}$$

Here $H(x_i) \in \{-1, +1\}$ is the label H predicts for the training instance x_i and $y_i \in \{-1, +1\}$ is again the observed true label for x_i. Moreover, H is considered to be an additive function of the following form, where $\beta_t \in \mathbb{R}$ are coefficients and h_t are base classifiers for $t = 1, \ldots, T$:

$$H(x) = \sum_{t=1}^{T} \beta_t h_t(x) , \tag{3}$$

The greedy method implemented in AdaBoost approximates the optimal solution to (2) in an iterative way, starting with $H_0 = 0$, choosing β_t and h_t so as to minimize $\sum_{i=1}^{n} \exp(-y_i(H_{t-1}(x_i) + \beta_t h_t(x_i)))$, and then setting $H_t(x) = H_{t-1}(x) + \beta_t h_t(x)$ and $H = H_T$. Defining $\omega_{t,i} = \exp(-y_i H_{t-1}(x_i))$, the goal is thus to find

$$\operatorname*{argmin}_{\beta_t, h_t} \sum_{i=1}^{n} \omega_{t,i} \exp(-\beta_t y_i h_t(x_i))) \tag{4}$$

$$= \operatorname*{argmin}_{\beta_t, h_t} \sum_{y_i = h_t(x_i)} \omega_{t,i} e^{-\beta_t} + \sum_{y_i \neq h_t(x_i)} \omega_{t,i} e^{\beta_t} . \tag{5}$$

The optimal solution to this iterative procedure (thus approximating the minimizer of (2)) yields $\beta_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$, as used in AdaBoost, cf. (Friedman, Hastie, and Tibshirani 2000).

Precision-based Boosting

AdaBoost weights a classifier based on its error – a low error ε_t of h_t results in a high weight β_t assigned to h_t. This is possible, because the stagewise model shown above makes the coefficient β_t dependent on the iteration number t. We refine this approach by adding one more degree of freedom to the optimization problem described above. In iteration t, we will allow the stagewise model to select two values of β_t, one value $\beta_{t,1}$ to handle instances x_i on which h_t predicts 1, and one value $\beta_{t,-1}$ to handle those on which h_t predicts -1. Thus one can assign a higher weight to a classifier h_t when it predicts class y, in case it generally has a high precision on class y. Intuitively, this will create an advantage over AdaBoost when the precision of base classifiers on the two classes is not equal.

For this purpose, we extend Equation (4); the goal is to find $\beta_{t,1}$, $\beta_{t,-1}$, and $h_t(x)$ that minimize

$$\begin{aligned} &\sum_{y_i = h_t(x_i)} \omega_{t,i} e^{-\beta_{t,1}} \mathbb{I}(h_t(x_i) = 1) \\ &+ \sum_{y_i = h_t(x_i)} \omega_{t,i} e^{-\beta_{t,-1}} \mathbb{I}(h_t(x_i) = -1) \\ &+ \sum_{y_i \neq h_t(x_i)} \omega_{t,i} e^{\beta_{t,1}} \mathbb{I}(h_t(x_i) = 1) \\ &+ \sum_{y_i \neq h_t(x_i)} \omega_{t,i} + \ e^{\beta_{t,-1}} \mathbb{I}(h_t(x_i) = -1) \end{aligned} \tag{6}$$

While (4) eventually yields a β_t-value depending on the total error ε_t of h_t, our approach requires two error terms, namely $\varepsilon_{t,1}$ for the error of h_t when it predicts class 1, and $\varepsilon_{t,-1}$ for the error of h_t when it predicts class -1. Specifically, the error $\varepsilon_{t,y}$ is defined as the cumulative weight of the training instances on which h_t *incorrectly* predicts y, under the distribution D_t. Similarly, we use $r_{t,y}$ to refer to the cumulative weight of the training instances in class y.

Definition 1. *For $y \in \{1, -1\}$ and $t \in \{1, \ldots, T\}$, let*

$$\begin{aligned} \varepsilon_{t,y} &= \sum_{i=1}^{n} D_t(x_i) \mathbb{I}(y_i \neq h_t(x_i)) \mathbb{I}(h_t(x_i) = y) \text{ and} \\ r_{t,y} &= \sum_{i=1}^{n} D_t(x_i) \mathbb{I}(y_i = y) . \end{aligned}$$

In what follows, if $y = 1$, then $\bar{y} = -1$, and, if $y = -1$, then $\bar{y} = 1$. The proof of Lemma 1 is omitted.

Lemma 1. *If $\varepsilon_{t,y} \neq 0$ for all t, y, then (6) is minimized by*

$$\begin{aligned} h_t &= \operatorname*{argmin}_{h} \sum_{i=1}^{n} \omega_{t,i} \mathbb{I}(y_i \neq h(x_i)) \text{ and} \\ \beta_{t,y} &= \frac{1}{2} \ln \frac{r_{t,y} - \varepsilon_{t,\bar{y}}}{\varepsilon_{t,y}} . \end{aligned}$$

We correspondingly adapt the framework for AdaBoost to a new boosting algorithm called *PrAdaBoost*, which is short for *Precision-based AdaBoost*, shown in Algorithm 2. Intuitively, AdaBoost and PrAdaBoost differ only in one simple aspect. For a given training dataset S and a set of base classifiers, AdaBoost adopts the following strategy: *The higher the accuracy of a base classifier (statistically on S), the more one should trust its predictions.* By comparison, PrAdaBoost's strategy is: *The higher the class-specific precision of a base classifier h for class y (statistically on S), the more one should trust h when it predicts class y.*

A minor difference in forming the final classifier is that AdaBoost has a perfect base classifier h_t when $\beta_t = \infty$. In our case, $\beta_{t,y} = \infty$ does not imply $\varepsilon_t = 0$ – one can only

[1] Throughout the paper, the notation $\mathbb{I}(P) \in \{0, 1\}$ refers to the binary indicator of the truth of condition P.

Algorithm 2: PrAdaBoost

Input: A training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$,
BaseInducer, i.e., a weak learning algorithm,
Number of iterations $T \in \mathbb{N}$.
Initialize $D_1(x_i) = \frac{1}{n}$ for all $i \in \{1, \ldots, n\}$.
For $t = 1$ to T **do**

1. Call BaseInducer with the distribution D_t to obtain a hypothesis $h_t : X \to Y$.
2. If $\varepsilon_t > \min_y r_{t,y}$, set T to $t-1$ and abort the loop.
3. $\beta_{t,1} = \frac{1}{2} \ln \frac{r_{t,1} - \varepsilon_{t,-1}}{\varepsilon_{t,1}}$ and $\beta_{t,-1} = \frac{1}{2} \ln \frac{r_{t,-1} - \varepsilon_{t,1}}{\varepsilon_{t,-1}}$.
4. Redistribute weights as follows, for $1 \leq i \leq n$:

$$D_{t+1}(x_i) = \frac{D_t(x_i)}{Z_t} \times \begin{cases} e^{-\beta_{t,h_t(x_i)}} \text{ if } h_t(x_i) = y_i \\ e^{\beta_{t,h_t(x_i)}} \text{ if } h_t(x_i) \neq y_i \end{cases}$$

where Z_t is defined such that $\sum_{i=1}^n D_{t+1}(x_i) = 1$.

end
Output: the final hypothesis

$$H(x) = \begin{cases} h_{t_\infty}(x)\,, & \text{if } T_\infty(x) \neq \emptyset,\ t_\infty = \min T_\infty(x)\,, \\ 1\,, & \text{else if } \sum_{t=1}^T \beta_{t,h_t(x)} h_t(x) \geq 0\,, \\ -1\,, & \text{otherwise}\,. \end{cases}$$

$$T_\infty(x) = \{t \mid \beta_{t,h_t(x)} = \infty\} \tag{7}$$

rely fully on h_t when it predicts y. This is addressed by incorporating $\min(T_\infty(x))$ in the output of PrAdaBoost. Our formal derivation of $\beta_{t,y}$ above assumes that these values are finite, so that $T_\infty(x) = \emptyset$; but to ensure proper logic in our algorithm, we introduce the case $T_\infty(x) \neq \emptyset$.

Note that the abort condition in line 2, to ensure the β values are non-negative, is weaker than the corresponding one used in AdaBoost, but is irrelevant in the typical practical deployment of AdaBoost-type methods. It is given just for formal reasons, as the pseudocode allows any kind of base inducer. Formally, the loop is aborted when the total error of any base classifier is higher than the weight of either class under the current distribution. However, that abort condition is never met when using a standard decision stump learner, as it would favour a constant classifier with error $\varepsilon_t = r_{t,y}$ (i.e., the trivial classifier always predicting the class with the highest cumulative weight) over any other decision stump with error $\varepsilon_t > r_{t,y}$. Likewise, line 2 of AdaBoost's pseudocode can be dropped when using decision stumps. PrAdaBoost's runtime complexity is equal to AdaBoost's.

Formal Analysis

AdaBoost is known to boost a series of base classifiers h_t with $\varepsilon_t < \frac{1}{2}$ into a classifier with arbitrarily small training error; known upper bounds on the training error decrease exponentially quickly as a function of T (Freund and Schapire 1996). We will now establish a similar result for PrAdaBoost that gives, in some cases, much smaller bounds on the training error than the corresponding bounds for AdaBoost.

Theorem 1. *W.r.t. the uniform distribution D_1, the error ε_H of the hypothesis H output by PrAdaBoost is bounded by*

$$\begin{aligned} &\varepsilon_H = Pr_{x_i \sim D_1}[H(x_i) \neq y_i] \\ \leq\ &\prod_{t=1}^T 2\sqrt{\varepsilon_{t,1}(r_{t,1} - \varepsilon_{t,-1})} + 2\sqrt{\varepsilon_{t,-1}(r_{t,-1} - \varepsilon_{t,1})}\,. \end{aligned}$$

If the base classifiers h_t in Algorithm 2 satisfy $\varepsilon_t < 1/2$ for all $t \in \{1, \ldots, T\}$, this bound decreases exponentially quickly as a function of T.

Our proof of this theorem follows in spirit the corresponding standard proof for AdaBoost, cf. (Mohri, Rostamizadeh, and Talwalkar 2018).

Proof. Let $F(x) = \sum_{t=1}^T \beta_{t,h_t(x)} h_t(x)$. We first show, for all i, $\mathbb{I}(H(x_i) \neq y_i) \leq \exp(-y_i F(x_i))$. It suffices to show that $H(x_i) \neq y_i$ yields $-y_i F(x_i) \geq 0$. If $T_\infty(x_i)$ in Algorithm 2 is empty, then $F(x_i) = 0$ or $H(x_i) = \operatorname{sgn}(F(x_i))$. Thus, if $H(x_i) \neq y_i$, we get $-y_i F(x_i) \geq 0$. If $T_\infty(x_i) \neq \emptyset$, then $H(x_i) = y_i$ by Algorithm 2, so there is nothing to show.

Note that D_{T+1} can be written in terms of D_1 as follows:

$$\begin{aligned} D_{T+1}(x_i) &= D_1(x_i) \cdot \prod_{t=1}^T \frac{\exp(-y_i \beta_{t,h_t(x_i)} h_t(x_i))}{Z_t} \\ &= \frac{\exp(-y_i \sum_{t=1}^T \beta_{t,h_t(x_i)} h_t(x_i))}{n \cdot \prod_{t=1}^T Z_t} = \frac{\exp(-y_i F(x_i))}{n \cdot \prod_{t=1}^T Z_t} \end{aligned}$$

Now we use $\mathbb{I}(H(x_i) \neq y_i) \leq \exp(-y_i F(x_i))$ to obtain

$$\begin{aligned} \varepsilon_H &= \sum_{i=1}^n D_1(x_i) \mathbb{I}(H(x_i) \neq y_i) \leq \sum_{i=1}^n \frac{1}{n} \exp(-y_i F(x_i)) \\ &= \sum_{i=1}^n D_{T+1}(x_i) \prod_{t=1}^T Z_t = \prod_{t=1}^T Z_t \end{aligned}$$

Using line 4 of Algorithm 2, one can express Z_t as

$$\begin{aligned} Z_t &= \sum_{i=1}^n D_t(x_i) \exp(-y_i \beta_{t,h_t(x_i)} h_t(x_i)) \\ &= \sum_{i: y_i = h_t(x_i)} D_t(x_i) e^{-\beta_{t,y_i}} + \sum_{i: \bar{y}_i = h_t(x_i)} D_t(x_i) e^{\beta_{t,\bar{y}_i}} \\ &= (r_{t,1} - \varepsilon_{t,-1}) e^{-\beta_{t,1}} + (r_{t,-1} - \varepsilon_{t,1}) e^{-\beta_{t,-1}} \\ &\quad + \varepsilon_{t,1} e^{\beta_{t,1}} + \varepsilon_{t,-1} e^{\beta_{t,-1}} \\ &= (r_{t,1} - \varepsilon_{t,-1}) \sqrt{\frac{\varepsilon_{t,1}}{r_{t,1} - \varepsilon_{t,-1}}} \\ &\quad + (r_{t,-1} - \varepsilon_{t,1}) \sqrt{\frac{\varepsilon_{t,-1}}{r_{t,-1} - \varepsilon_{t,1}}} \\ &\quad + \varepsilon_{t,1} \sqrt{\frac{r_{t,1} - \varepsilon_{t,-1}}{\varepsilon_{t,1}}} + \varepsilon_{t,-1} \sqrt{\frac{r_{t,-1} - \varepsilon_{t,1}}{\varepsilon_{t,-1}}} \\ &= 2\sqrt{\varepsilon_{t,1}(r_{t,1} - \varepsilon_{t,-1})} + 2\sqrt{\varepsilon_{t,-1}(r_{t,-1} - \varepsilon_{t,1})} \end{aligned}$$

Thus one obtains the claimed upper bound

$$\varepsilon_H \leq \prod_{t=1}^T 2\big(\sqrt{\varepsilon_{t,1}(r_{t,1} - \varepsilon_{t,-1})} + \sqrt{\varepsilon_{t,-1}(r_{t,-1} - \varepsilon_{t,1})}\big)$$

To show that this bound drops exponentially quickly as a function of T, we use the known upper bound on the training error of AdaBoost, given by the formula

$$\prod_{t=1}^{T} 2\sqrt{\varepsilon_t(1-\varepsilon_t)}\,, \tag{8}$$

cf. (Freund and Schapire 1996; Mohri, Rostamizadeh, and Talwalkar 2018). This bound is known to drop exponentially quickly as a function of T if $\varepsilon_t < 1/2$ for all t. We will see that $2\big(\sqrt{\varepsilon_{t,1}(r_{t,1}-\varepsilon_{t,-1})}+\sqrt{\varepsilon_{t,-1}(r_{t,-1}-\varepsilon_{t,1})}\big) \leq 2(2\sqrt{\varepsilon_t(1-\varepsilon_t)})$, i.e., our bound is within a factor of 2 of the one known for AdaBoost and therefore also drops exponentially quickly as a function of T.

To verify that our bound is within a factor of 2 of the one for AdaBoost, it suffices to observe that

$$\begin{aligned}\varepsilon_{t,y}(r_{t,y}-\varepsilon_{t,\bar{y}}) &\leq \varepsilon_{t,y}(r_{t,y}-\varepsilon_{t,\bar{y}})+\varepsilon_t(r_{t,\bar{y}}-\varepsilon_{t,y})\\ &\leq \varepsilon_t(r_{t,y}-\varepsilon_{t,\bar{y}})+\varepsilon_t(r_{t,\bar{y}}-\varepsilon_{t,y})\\ &= \varepsilon_t(r_{t,y}+r_{t,\bar{y}}-\varepsilon_{t,\bar{y}}-\varepsilon_{t,y})\\ &= \varepsilon_t(1-\varepsilon_t)\end{aligned}$$

□

Our proof uses a crude approximation to show that our upper bound on PrAdaBoost's training error is at most twice as large as that of AdaBoost. Very often in fact, our bound on PrAdaBoost's training error (from Theorem 1) is smaller than the bound $\prod_{t=1}^{T} 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$ (Freund and Schapire 1996; Mohri, Rostamizadeh, and Talwalkar 2018) for AdaBoost – a claim that we also tested empirically (results not included here). Here we present a hypothetical situation in which a small difference between the bounds results in a large difference in the number of iterations after which the bounds can guarantee a certain target training accuracy.

Suppose the maximal error $\max_{t\leq T}\varepsilon_t$ incurred by both algorithms over T iterations is 0.48 and the target error to be reached is 0.02. Suppose further that the classes are never quite balanced throughout PrAdaBoost's iterations, e.g., $r_{t,1}\leq 0.4$ and $r_{t,-1}\geq 0.6$, and that the class-specific errors (which sum up to ε_t) are such that $\varepsilon_{t,1}\leq 0.285$ and $\varepsilon_{t,-1}\leq 0.195$ for all t. Under these conditions, the maximum possible value of $2\sqrt{\varepsilon_{t,1}(r_{t,1}-\varepsilon_{t,-1})}+2\sqrt{\varepsilon_{t,-1}(r_{t,-1}-\varepsilon_{t,1})}$ is roughly 0.97910 and occurs when $r_{t,1}=0.4$, $r_{t,-1}=0.6$, $\varepsilon_{t,1}=0.285$, $\varepsilon_{t,-1}=0.195$. The maximum possible value of $2\sqrt{\varepsilon_t(1-\varepsilon_t)}$ is roughly 0.99919, for $\varepsilon_t=0.48$. Making a pessimistic estimate for both algorithms, by considering the maximum possible values to occur not just in one but in all iterations, PrAdaBoost's bound evaluates to roughly 0.97910^T. This number is less than 0.02 if and only if $T\geq 186$. By comparison, AdaBoost's bound $\prod_{t=1}^{T} 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$, which is based only on ε_t, evaluates to roughly 0.99919^T and is below 0.02 only when $T\geq 4828$, which is almost 26 times the value obtained through PrAdaBoost's bound. Note that, for such upper-bound estimates on AdaBoost-type algorithms, it is standard practice to assume the maximum possible error under the given constraints occurs in every iteration, cf. (Freund and Schapire 1996; Mohri, Rostamizadeh, and Talwalkar 2018).

A Multi-class Variant

Precision-based scoring parameters can similarly be applied in variants of AdaBoost, such as AdaCost etc. We were particularly interested in studying the effect of precision-based boosting in multi-class classification; here we expected a more noticeable benefit of precision-based boosting due to the higher chance of base classifiers having large differences in precision for at least some of the classes.

The best-known multi-class AdaBoost variants are AdaBoost.M1, AdaBoost.M2 (Freund and Schapire 1996), AdaBoost.MH (Schapire and Singer 1999), and SAMME (Hastie et al. 2009). Since SAMME is simpler than AdaBoost.MH and was reported to outperform AdaBoost.M1 and AdaBoost.M2, we chose SAMME as a method for our analysis. Using the same approach as in Section , we formulate a forward stagewise additive model in a class-specific approach, but this time starting from the multi-class exponential loss function from which Hastie et al. (Hastie et al. 2009) derived SAMME. Minimizing the loss as done in Section for the binary case, one obtains optimal parameters of a class-specific flavor.

We adapt Definition 1 to a multi-class setting with K classes, by introducing a dual form of class error.

Definition 2. *Let $y\in\{1,\ldots,K\}$ and $t\in\{1,\ldots,T\}$. The dual class error $\varepsilon'_{t,y}$ is defined as the cumulative weight of the training instances in class y which are* incorrectly *predicted by h_t, under the distribution D_t.*

$$\varepsilon'_{t,y}=\sum_{i=1}^{n} D_t(x_i)\mathbb{I}(y_i\neq h_t(x_i))\mathbb{I}(y_i=y)$$

Then, the following scoring parameters are obtained with a calculation similar to that in Section :

Lemma 2. *Assuming $y\in\{1,\ldots,K\}$ and $t\in\{1,\ldots,T\}$, the following expression minimizes the precision-based version of SAMME's loss function.*

$$\beta_{t,y}=\frac{(K-1)^2}{K}\Big(\ln\frac{r_{t,y}-\varepsilon'_{t,y}}{\varepsilon_{t,y}}+\ln(K-1)\Big)$$

Using such $\beta_{t,y}$ as scoring parameters within SAMME, one obtains a method we call PrSAMME, analogously to the way PrAdaBoost was obtained from AdaBoost.

Empirical Analysis

We evaluated (Pr)AdaBoost on 23 binary UCI datasets (Lichman 2013) and (Pr)SAMME on 18 multi-class UCI datasets, using decision stumps trained in Matlab as base classifiers. We performed 10-fold cross validation on each dataset (except 'isolet', which has designated training and test portions), comparing two algorithms always on the same folds. All errors were calculated w.r.t. the uniform distribution over the input data. The datasets were preprocessed to remove all data points with missing attribute values.

Binary Classification

We ran PrAdaBoost and AdaBoost for T iterations, trying $T=30$, 50, and 100 (without attempting to tune T.) In each

case, we report (i) test error averages from 10-fold cross-validation over iterations 1 through T, and (ii) test error at iteration T. We also tried excluding the first 9 iterations from the average in (i), so as not to penalize a method for very early erroneous predictions, but since this did not change the trends the corresponding results are not reported here.

The results for $T = 50$ are given in Table 1, where statistically significant wins (using 2-tailed paired t-tests at the 95% confidence level) are shown in bold. Datasets are listed in decreasing order of size; from the top down to *german* are those with 1,000 or more instances. Note that not many of the results for iteration 50 alone are statistically significant, as they are based on only 10 result pairs (one per fold), whereas for iterations 1–50 we compare 50×10 result pairs.

PrAdaBoost tends to outperform AdaBoost (i) on larger datasets but also (ii) on datasets with class imbalance. Concerning (i), note that the datasets containing more than 1,000 instances are the top 14 in Table 1 – AdaBoost wins on only 4 of these 14 in terms of test error, while winning on 8 of the 9 small sets. This suggests that the training data in larger datasets become representative enough of the test data to give PrAdaBoost a crucial advantage over AdaBoost. As for (ii), note that PrAdaBoost takes class ratios into account explicitly via the $r_{1,y}$ values in iteration $t = 1$. When redistributing weights, class imbalance may decline, so it is hard to predict its effect for larger t. In our study, the datasets in which the size ratio between the larger and the smaller class is less than 1.5 are the following: *krvskp*, *banknote*, *messidor*, *crx*, *cleveland*, *house*, *sonarall*, and *promoters*. PrAdaBoost wins on only 2 out of these 8 datasets in terms of test error. By contrast, it wins in terms of test error on 9 out of 15 datasets with class ratio greater than 1.5. This suggests that PrAdaBoost provides much more of an advantage over AdaBoost for imbalanced datasets than for balanced ones.

Results for $T = 30$ showed similar trends, so that we do not display them here. For $T = 100$, we cannot make a meaningful comparison. While the errors incurred by AdaBoost keep getting slightly smaller, PrAdaBoost's errors appear to increase. The latter though seems to be due to numerical imprecision. PrAdaBoost adjusts the weights of individual training examples much more aggressively than AdaBoost, so that weights can become extremely small very quickly. In our experiments, this caused weights to be treated as zero when in fact they weren't, so that our test runs could not numerically perform the operations prescribed by PrAdaBoost. Efforts to resolve this problem, e.g., using smoothing or exact rational arithmetic, are left for future studies.

Comparison to InfoBoost

Aslam (2000) proposed InfoBoost as a first AdaBoost-type method that takes class-specific precision into account in its weighting scheme. The parameters used in InfoBoost differ from those used in PrAdaBoost, and appear substantially inferior, see Table 2. In terms of test error, PrAdaBoost significantly outperforms InfoBoost on 21 of the 23 tested datasets, ties on one dataset, and loses to InfoBoost on only one dataset (promoters). Moreover, PrAdaBoost mostly wins by a large margin, and it does not seem as if InfoBoost can compete with the original AdaBoost either.

	1-50		50 alone	
Name	AdaB	Pr	AdaB	Pr
bankfull	10.50	**10.33**	10.21	**10.03**
adult	15.98	**15.11**	14.69	**14.22**
creditcard	**18.06**	18.09	18.06	18.10
magic04	19.14	**17.98**	17.22	**16.04**
htru2	2.27	**2.20**	2.27	**2.16**
agaricus	0.17	**0.07**	0	0
spambase	8.98	**7.89**	7.10	6.56
krvskp	7.73	**7.15**	5.60	5.44
seismic	**6.58**	6.79	6.57	6.59
titanic	**22.22**	22.29	22.17	22.17
banknote	**2.88**	3.52	**0.43**	2.26
messidor	35.03	**33.45**	32.23	32.40
biodeg	18.68	**16.48**	17.44	14.98
german	26.19	**25.62**	24.30	25.50
breastc	**4.73**	5.60	**4.24**	6.59
crx	**13.45**	14.63	13.16	15.62
diabetes	**21.65**	24.22	**21.89**	25.96
ionosphere	9.91	**8.79**	8.57	7.12
cleveland	**17.30**	17.86	17.13	17.81
house	**3.38**	5.68	3.46	8.58
sonarall	**19.87**	20.91	**15.38**	22.11
promoters	**10.36**	15.31	8.72	17.09
hepatitis	**15.13**	16.78	13.75	17.50

Table 1: Test errors (%) of AdaBoost and PrAdaBoost, averaged over iterations 1–50 and at iteration 50 alone.

To show that this is not an issue of InfoBoost aggressively overfitting, we display training errors as well. Given InfoBoost's poor performance compared to PrAdaBoost over 50 iterations, we did not evaluate InfoBoost further.

Multi-Class Classification

Table 3 compares PrSAMME and SAMME in a 10-fold cross-validation and for 100 iterations, with bold entries again referring to statistically significant wins at the 95% confidence level (paired t-tests).

PrSAMME clearly outperforms SAMME (15 wins, 2 losses, 1 tie, when averaging over 100 iterations). Note that, in the few cases in which SAMME's errors are significantly less than PrSAMME's, their differences are still very small, whereas PrSAMME often beats SAMME by a large margin. PrSAMME tends to outperform SAMME more strongly when the number of classes increases. 13 of the 18 tested datasets have more than 3 classes. PrSAMME significantly outperforms SAMME on all of them. The datasets in which PrSAMME does not substantially beat SAMME (*splice*, *wine*, and *iris*) have only 3 classes each.

Stronger Base Classifiers

To see whether the trends in our results are specific to the use of decision stumps as base classifiers, we also tested both approaches using stronger base classifiers. Allowing trees with up to 5 splits as base classifiers, we tested the four largest two-class datasets and the four largest multi-class datasets: bankfull, adult, creditcard, magic04, connect4, sensorless, shuttle, and letter. For 'shuttle', SAMME and PrSAMME

	Training error		Test error	
Name	Info	Pr	Info	Pr
bankfull	18.34	**10.26**	18.38	**10.33**
adult	33.97	**15.01**	33.99	**15.11**
creditcard	34.85	**18.00**	34.87	**18.09**
magic04	45.51	**17.43**	45.77	**17.98**
htru2	8.04	**2.15**	8.10	**2.20**
agaricus	29.91	**0.07**	29.91	**0.07**
spambase	13.83	**6.97**	14.55	**7.89**
krvskp	17.87	**6.93**	18.02	**7.15**
seismic	13.27	**6.59**	13.86	**6.79**
titanic	32.75	**22.27**	32.90	**22.29**
banknote	2.93	**2.81**	3.62	3.52
messidor	42.62	**26.92**	45.52	**33.45**
biodeg	22.97	**12.89**	26.68	**16.48**
german	47.61	**21.42**	49.41	**25.62**
breast	4.27	**2.78**	6.85	**5.60**
crx	23.35	**10.34**	27.08	**14.63**
diabetes	24.23	**14.85**	29.85	**24.22**
ionosphere	5.64	**4.00**	12.30	**8.79**
cleveland	18.38	**12.02**	25.89	**17.86**
house	5.24	**4.27**	7.01	**5.68**
sonarall	7.08	**5.19**	25.19	**20.91**
promoters	2.19	**1.52**	**14.23**	15.31
hepatitis	9.34	**1.01**	25.30	**16.78**

Table 2: Training/test errors (%) of InfoBoost and PrAdaBoost, averaged over iterations 1 through 50.

	1–100		100 alone	
Name	SAMME	Pr	SAMME	Pr
connect4	29.58	**28.14**	27.75	**25.75**
sensorless	59.64	**45.64**	48.26	43.65
shuttle	5.54	**0.89**	2.98	0.09
letter	73.06	**66.30**	58.18	56.39
nursery	19.96	**14.96**	23.40	**14.27**
pendigit	44.63	**35.52**	29.34	**25.61**
isolet	79.91	**65.73**	68.44	52.34
satimage	25.25	**23.78**	**20.44**	22.44
splice	**6.80**	7.18	**5.33**	6.30
segment	24.97	**19.80**	17.01	16.75
yeast	68.53	**68.39**	68.53	68.39
vowel	68.84	**62.95**	63.74	**57.17**
vehicle	41.30	**40.31**	39.37	37.12
glass	47.62	**45.35**	48.98	43.10
seed	8.66	**6.94**	8.57	7.14
wine	4.85	4.72	3.40	3.46
iris	**5.95**	7.13	4.67	8.00
breastt	40.24	**37.26**	31.00	38.55

Table 3: Test errors (%) of SAMME and PrSAMME, averaged over iterations 1–100 and at iteration 100 alone; datasets are in decreasing order of size.

were on par; for the other seven datasets the comparison results were similar to those we reported for decision stumps, just with smaller error values throughout. In particular, Pr won significantly on six datasets and lost slightly only on creditcard. This suggests that, at least for larger datasets, precision-based methods are preferable even when using stronger base classifiers. Since most of the literature uses AdaBoost with decision stumps, we did not extend our study on stronger classifiers beyond the reported eight datasets.

Conclusions

We demonstrated that the AdaBoost framework can substantially benefit from using class-specific precision in its weighting scheme. In particular, we formally derived a weighting scheme that is provably optimal under additive forward stagewise modeling in a setting that generalizes the one on which AdaBoost is built. The resulting algorithm PrAdaBoost is guaranteed to convert a weak learner into a strong learner. It provably decreases its training error exponentially quickly, and the obtained error bound can often substantially improve on the known ones for AdaBoost. Similarly, virtually all known ensemble classification methods built on AdaBoost can be adapted to take class-based precision into account, as illustrated for SAMME. One interesting direction for future work would be to establish generalization error bounds for precision-based boosting.

Experiments on 23 two-class and 18 multi-class benchmark datasets showed PrAdaBoost and PrSAMME to be superior to AdaBoost and SAMME, resp., in many cases. Typically, larger datasets as well as datasets with more classes support our conclusion more strongly. Class imbalance also tends to be more easily handled by the precision-based approach. These trends seem plausible. Given multiple classes, the chances are higher that class-specific precision of any base classifier varies among classes, thus increasing the positive effect of the precision-based approach. Potential negative effects of class imbalance are possibly mitigated through the $r_{t,y}$ terms. The fact that PrAdaBoost does not beat AdaBoost on the smaller datasets tested, presumably has its reasons in the aggressiveness of PrAdaBoost's weight update technique. The latter may have adverse effects when the training data are not representative of the test data – which is more likely for small datasets. Regularization might help to mitigate PrAdaBoost's problems with overgeneralization on small datasets, and possibly also the numerical issues we encountered when running it for many iterations.

We do not claim that PrAdaBoost/PrSAMME are not outperformed by any existing boosting method, but that they tend to outperform their non-precision-based counterparts AdaBoost/SAMME on larger/multi-class datasets. In the same vein, we propose to consider replacing other existing AdaBoost-type methods by their corresponding precision-based variants when dealing with larger datasets.

In the sum, our experiments from both binary and multi-class settings suggest that the preferred choice, when running an AdaBoost-type method on datasets with more than 1,000 instances or with more than 3 classes, should be to use the precision-based approach that we propose, as it is expected to improve the predictive performance.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), through the Discovery Grants and Canada Research Chairs programs.

References

Aslam, J. A. 2000. Improving Algorithms for Boosting. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory (COLT))*, 200–207. Morgan Kaufmann.

Breiman, L. 1996. Bagging Predictors. *Machine Learning* 24(2): 123–140. doi:10.1007/BF00058655. URL http://dx.doi.org/10.1007/BF00058655.

Breiman, L. 1999. Prediction games and arcing algorithms. *Neural Computation* 11(7): 1493–1517.

Breiman, L. 2001. Random Forests. *Machine Learning* 45(1): 5–32. doi:10.1023/A:1010933404324. URL http://dx.doi.org/10.1023/A:1010933404324.

Fan, W.; Stolfo, S. J.; Zhang, J.; and Chan, P. K. 1999. AdaCost: misclassification cost-sensitive boosting. In *ICML*, 97–105.

Freund, Y.; and Schapire, R. E. 1996. Experiments with a new boosting algorithm. In *ICML*, 148–156.

Freund, Y.; and Schapire, R. E. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences* 55(1): 119–139.

Friedman, J.; Hastie, T.; and Tibshirani, R. 2000. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics* 28: 337–407.

Hastie, T.; Rosset, S.; Zhu, J.; and Zou, H. 2009. Multi-class AdaBoost. *Statistics and its Interface* 2: 349–360.

Lichman, M. 2013. UCI Machine Learning Repository. URL http://archive.ics.uci.edu/ml.

Mohri, M.; Rostamizadeh, A.; and Talwalkar, A. 2018. *Foundations of Machine Learning, 2nd ed.* MIT Press.

Nikolaou, N.; and Brown, G. 2015. Calibrating AdaBoost for Asymmetric Learning. In *Proceedings of the 12th International Workshop on Multiple Classifier Systems (MCS)*, 112–124.

Rokach, L. 2010. Ensemble-based classifiers. *Artificial Intelligence Review* 33(1-2): 1–39.

Schapire, R.; and Freund, Y. 2012. *Boosting: Foundations and Algorithms*. MIT Press.

Schapire, R. E.; and Singer, Y. 1999. Improved boosting algorithms using confidence-rated predictions. *Machine learning* 37(3): 297–336.

Wu, S.; and Nagahashi, H. 2015. Analysis of Generalization Ability for Different AdaBoost Variants Based on Classification and Regression Trees. *J. Electrical and Computer Engineering* 2015: 835357:1–835357:17.