*i*Boost: Boosting Using an *i*nstance-Based Exponential Weighting Scheme

Stephen Kwek and Chau Nguyen

Computational Learning Group, Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
{kwek,cnguyen}@cs.utsa.edu

Abstract. Recently, Freund, Mansour and Schapire established that using exponential weighting scheme in combining classifiers reduces the problem of overfitting. Also, Helmbold, Kwek and Pitt that showed in the prediction using a pool of experts framework an instance based weighting scheme improves performance. Motivated by these results, we propose here an instance-based exponential weighting scheme in which the weights of the base classifiers are adjusted according to the test instance x. Here, a competency classifier c_i is constructed for each base classifier h_i to predict whether the base classifier's guess of x's label can be trusted and adjust the weight of h_i accordingly. We show that this instance-based exponential weighting scheme enhances the performance of AdaBoost.

1 Introduction

Recent research in classification problems has placed an emphasis on ensemble methods that construct a set of base classifiers instead of a single classifier. An unlabeled instance is then classified by taking a vote of the base classifiers' predictions of its class label. Ensemble methods like Bagging [1] and AdaBoost [5] have been shown to outperform the individual base classifiers when the *base inducer* that produces the base classifiers is unstable. An inducer is said to be *unstable* if a slight change in the training examples results in a very different classifier being constructed. Further, the idea of ensemble methods also gives rise to the use of error-correcting output code technique in enhancing accuracy in multi-class classification problems.

In ensemble methods, the vote of each base classifier either receives the same weight (e.g. Bagging and Arcing) or is weighted according to the estimated error rate (e.g. AdaBoost). Based on a recent work of Helmbold, Pitt and the first author [6] we propose an instance-based approach of assigning weights to the base classifiers. Instead of assigning a weight to a base classifier that is fixed for all instances, we attempt to assign a weight that is based upon how well we think the base classifier is going to predict on the label of the test instance. Intuitively, given an unlabeled instance x, if there is some indication that the base classifier's

T. Elomaa et al. (Eds.): ECML, LNAI 2430, pp. 245–257, 2002.

prediction is correct then we should increase its weight. Otherwise, we should reduce its weight. This paper proposes a new version of boosting algorithm *i*Boost that uses such *i*nstance-based weighting scheme. We demonstrate that *i*Boost enhances the performance AdaBoost. Another difference between *i*Boost and AdaBoost is that *i*Boost adopts the exponential weighting scheme, as the work of Helmbold et. al. [6] is based on the weighted majority framework (see discussion in Section 3.1). We believe that the technique employed here can be applied to ensemble methods in general.

2 The AdaBoost Algorithm

Before elaborating further, we shall define some notations that we use in this proposal. A *labeled instance* is a pair $\langle x, y\rangle$ where x is an element from some *instance space* X and y comes from a set Y of nominal values. We assume a probability distribution $\mathcal{D}$ over the space of labeled instances. A *sample* S is a set of labeled instances $S = \{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, ...\langle x_m, y_m\rangle\}$ drawn from the independent and identical probability distribution $\mathcal{D}$. A *classifier* or *hypothesis* is a mapping from X to Y. An *inducer* or *learner* takes a sample S as training data and constructs a classifier. In ensemble methods, multiple base classifiers are created by calling a *base inducer* over different training examples.

Boosting was originally introduced by Schapire to address the question, posed by Kearns and Valiant [8], of whether a weak PAC (Probably Approximately Correct [17]) learner (which outputs a hypothesis that is slightly better than random guess) can be turned into a strong PAC learner of arbitrary accuracy [1]. Schapire [14] came up with the first provable polynomial time algorithm to 'boost' a weak learner into a strong learner. A year later, Freund [3] presented a much more efficient boosting algorithm. Unfortunately, the strong theoretical results assume the availability of a (polynomially) large training sample (depending on the desired accuracy and confidence). However, in most practical applications, the size of training sample is very limited which severely curb the usefulness of both algorithms.

In 1995, Freund and Schapire [5] attempt to resolve this difficulty by introducing AdaBoost (**Ada**ptive **Boost**ing) as shown in Figure 1. The input to the algorithm is a set of labeled examples $S = \{(x_1, y_1), ..., (x_m, y_m)\}$. As in Bagging, AdaBoost calls a base inducer repeatedly in a sequence of iterations $t = 1, ..., T$ to produce a set of weak base classifiers $\{h_1, ..., h_T\}$. However, unlike Bagging where the training sets are drawn uniformly in each iteration, AdaBoost maintains a distribution or a set of weights over the training set. The weight of this distribution on training example i on the tth iteration is denoted by $D_t(i)$. Initially, all the weights are set equally to $1/m$. On each iteration, the weights of those incorrectly classified examples are increased while the weights of those correctly classified are decreased. The base inducer's task is to construct a base classifier that minimizes the error $\epsilon_t = Pr_{i\sim D_t}[h_t(x_i) \neq y_i]$. The effect of the

[1] Assuming 'polynomial' number of labeled examples are available and the target concept to be learned is in the hypothesis class.

Given: $S = (x_1, y_1), ..., (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
and the number of iterations T

Initialize $D_1(i) = 1/m$
for $t = 1, ..., T$
Train base learner using distribution D_t
Get base classifier $h_t : X \to R$
Compute $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$
Update:

$$D_{t+1}(i) = \frac{D_t(i) \left(\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \right)^{(y_i h_t(x_i))}}{Z_t}$$

where Z_t is a normalization factor that ensures D_{t+1} is a probability distribution.

Output the final classifier:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right) \text{ where } \alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right).$$

Fig. 1. The boosting algorithm AdaBoost for binary-class problem.

reweighting is that in the next iteration, the base inducer is forced to concentrate more on those difficult examples.

For each labeled example $\langle x_i, y_i \rangle$ that is incorrectly labeled by h_t, we increase its weight $D_{t+1}(i)$ by a factor of $\sqrt{(1-\epsilon_t)/\epsilon_t}$ (of $D_t(i)$). Those that are correctly labeled have their weights reduced by a factor of $\sqrt{\epsilon_t/(1-\epsilon_t)}$. The *final* or *combined classifier* H is a weighted majority vote of the predictions made by the T base classifiers where the prediction of each h_t is assigned a weight of

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

3 *i*Boost

3.1 Inspiration

The inspiration behind the theme for this proposal arises from earlier computational learning theory (COLT) research work on learning from a pool of experts. In [11], Littlestone and Warmuth studied the problem of making on-line prediction using a pool of experts. In their model, the learner faces a (possibly infinite) sequence of trials, with a boolean prediction to be made in each trial. The goal of the learner is to make minimum number of mistakes. The learner is allowed to make his prediction by observing how a given pool of experts predict. The

underlying assumption is that at least one of these experts will perform well but the learner does not know which one.

They propose the weighted majority algorithm that works as follows. A weight is associated with each expert, and is initially set to one. The learner predicts negative if the sum of the weights of all the experts that predict negative is greater than that of the experts that predict positive, otherwise the learner predicts positive. When a mistake is made, the learner simply multiplies the weights of those experts that predict wrongly by some fixed non-negative constant β smaller than one. They showed that if the best expert makes at most η mistakes, then on any sequence of possibly infinite trials, the weighted majority algorithm makes at most $O(\log |\mathcal{E}| + \eta)$ mistakes. Various alternative weighting schemes have also been proposed.

In the weighted majority algorithm and its variants, the weight of an expert is β^m where m is the number of errors made by the expert so far. Such weighting scheme is know more generally as exponential gradient descent where the number of mistakes or the error rate appear in the exponent of the weight. Although most work are in the online learning model, Freund et. al. [4] recently established theoretical result in the offline batch model. In their work, the final prediction is made by taking the weighted average of all hypotheses, weighted exponentially w.r.t. their training errors. They showed that this weighting scheme can protect against overfitting problem commonly encountered by those algorithms that predict with the best (single) hypothesis, and hence is more stable.

Out of curiosity, we decided to investigate whether adapting an exponential weighting scheme in AdaBoost improves prediction accuracy. Unfortunately, we found a decrease in prediction accuracy. This preliminary study and an earlier work [6] of the first author (see discussion below), suggest that a better exponential weighting scheme variant of AdaBoost may have to take into consideration of the unlabeled test instance in assigning the hypotheses' weights.

Going back to the framework of prediction using a pool of experts, suppose the experts' predictions and the actual outcome depend on some input (instance) in each trial. Notice that the weighted majority and its variations do not make use of information specific to the input when calculating the weights of the experts. This may turn out to be a missed opportunity as this information may help to determine which experts are likely to predict correctly.

To illustrate this, consider the following example with a boolean instance space $\{0,1\}^n$ and two experts, E_0 and E_1 that always give opposite predictions. If E_0 makes at most a small number, m_0, of mistakes when one crucial component of the instance is set to 0, and E_1 makes at most m_1 mistakes when that crucial component is set to 1, then the weighted majority and similar schemes can be forced to make a mistake on almost every point in the instance space (more precisely, $2^n - |m_0 - m_1|$ mistakes). This remains true even if the learner uses table lookup to remember all the previous mistakes. However, if the learner uses E_0's predictions when the crucial input component was set to 0 and E_1's predictions otherwise, then the learner makes at most $m_0 + m_1$ mistakes when it maintains a table of labeled counterexamples. In other words, although neither expert is

very competent, they become competent collectively if we consider restricting the use of each expert to the appropriate subset of the instance space.

Unfortunately, those algorithms that employ the weighted majority techniques do not take advantage of the above situation. Hence, Helmbold, Pitt and the first author [6] proposed a theoretical framework to capture this notion of 'the whole is often greater than the sum of its parts' by considering the regions in which the experts are competent in. Within this framework, we established various positive theoretical results. The next natural step is to perform experimental work to empirically verify that the notion of competency regions is superior to simply taking a weighted vote of the experts. In trying to do so, we realize that the on-line prediction using a pool of experts bares great similarity to the ensemble methods in that both use a simple weighting scheme where each expert, or base classifier, has the same weight for all instances. Further, the base-classifiers in AdaBoost are trained using different distributions hence we expect their competency regions to be different. Thus, we decide to apply the idea of competency region to AdaBoost and also adopt the exponential weighting scheme.

3.2 The Algorithm *i*Boost

We started by experimenting with AdaBoost but with the weight of each base classifier adjusted according to whether there is evidence suggesting that it may predict well for that specific test instance (in addition to its overall training error ϵ_t). The experiment was implemented on top of the open source machine learning software Weka 3 [18] provided by the University of Waikato in New Zealand. We call our algorithm, which is shown in Figure 2, *i*Boost (**i**nstance-based **boost**ing). We use the decision tree inducer J48 in the Weka software package (which implements a version of C4.5) to construct the base classifiers.

We modify AdaBoost so that in the tth iteration, after the base decision tree classifier h_t has been constructed, we label each sample as '$\surd$' or '$\times$' depending on whether it is labeled correctly by h_t. We then use the decision tree inducer J48 to learn from this newly labeled sample a *competency predictor* c_t for predicting whether h_i's prediction can be trusted on a given unlabeled instance. Following the spirit of the weighted majority paradigm and the work of Freund et. al. [4], we adopt the exponential weighting scheme by setting the initial weight of each base classifier h_t to $e^{-\epsilon_t}$.

Given an unlabeled instance x, the weight h_t of the base classifier does not depend solely on h_t's estimated error rate, ϵ_t, but also on whether $c_t(x)$ is '$\surd$' or '$\times$'. If $c_t(x) = \surd$ then there is evidence that the prediction of h_t on x's label can be trusted. Thus, h_t's weight should be increased. In this case, we treat c_t as another expert that makes the same prediction on x as h_t. It receives a weight of $e^{-\epsilon'_t}$ where ϵ'_t is c'_t's estimated error rate on S. Thus, *i*Boost sets $h_t(x)$'s weight α_t to $\eta e^{-\epsilon_t} + (1-\eta)e^{-\epsilon'_t}$. Here, η is a parameter between 0 and 1 that places the relative importance of the two experts c_t and h_t. Unless explicitly stated otherwise, we shall assume throughout this paper that $\eta = 0.5$.

On the other hand, if $c_t(x) = \times$ then it suggests that h_t may not be competent in predicting x's label. Here, we need to reduce the weight of h_t. We do this by

Given: $S = \{(x_1, y_1), ..., (x_m, y_m) : x_i \in X, y_i \in Y = \{-1, +1\}\}$ and the number of iterations T

Same code as the original AdaBoost but with the following addition

Create a sample S_t from S by labeling each example x_i as "$\surd$" or "$\times$" depending on whether $h_t(x_i) = y_i$.
Train competency predictor c_t using S_t.
Compute c_t''s estimated error rate, ϵ_t', on S.

Output the final classifier:

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

where

$$\alpha_t(x) = \begin{cases} \eta e^{-\epsilon_t} + (1-\eta)e^{-\epsilon_t'} & \text{if } c_t(x) = \text{`}\surd\text{'} \\ \max\left(0.0001e^{-\epsilon_t}, \eta e^{-\epsilon_t} - (1-\eta)e^{-\epsilon_t'}\right) & \text{if } c_t(x) = \text{`}\times\text{'} \end{cases}$$

Fig. 2. *i*Boost for binary class problems

treating c_t as another expert that predicts $\neg h_t(x)$, the opposite of h_t's prediction. As before, c_t gets a weight of $e^{-\epsilon_t'}$, and the overall weight of h_t is set to $\eta e^{-\epsilon_t} - (1-\eta)e^{-\epsilon_t'}$. We avoid using negative weights because our preliminary investigation indicates that having negative weights decreases prediction accuracy. Thus, If this modified weight is zero or negative, we set it to $0.0001e^{-\epsilon_t}$. We choose not to set the weight to 0 in the latter situation just in case all the c_t's predict '$\times$'.

3.3 Previous Related Work

The idea for adjusting weights based on the unlabeled test instance has been pioneered earlier in neural network research by Jordan and Jacobs [7]. In their work, the predictions of the experts (i.e., base classifiers) are combined by a tree architecture neural network. The weights in the neural network is obtained by gating networks that take the test instance as input. Thus, the test instance effectively determined the weights.

A closely related idea is that of RegionBoost proposed by Maclin [12]. In his work, he aimed to predict, for each base classifier, the probability that the test instance is misclassified. He proposed to replace the base classifier's estimated (overall) error rate by this probability in assigning weight to the base classifier. He studied the usage of k-Nearest neighbor and neural network approach in estimating this probability. Using UCI data sets, he illustrated that RegionBoost has

a slight edge over AdaBoost. *i*Boost is very similar to RegionBoost in the sense that both try to establish a better assignment of base-classifiers' weights according to the test instance. However, the exponential weighting scheme is motivated by the work of Freund et. al. [4] and earlier COLT work on weighted majority weighting scheme. More importantly, the improvements obtained by *i*Boost are larger than that of RegionBoost. Further, while RegionBoost is even more susceptible to overfitting than AdaBoost [12], *i*Boost seems to better reduce the problem of overfitting (see Section 4). This is not surprising since Freund et. al. [4] have established theoretically that (weighted) averaging experts' prediction using exponential weighting scheme protect against the overfitting problem.

Besides taking a weighted vote, another popular way of combining base classifiers' predictions is stacking [19]. In stacking, an inducer is used to learn how to combine the predictions of the base-classifiers. In this stacking framework, Todorovski and Dzeroski [16] proposed constructing a decision tree that selects a (single) base classifier to use. As in their work, *i*Boost invokes a decision tree inducer to learn to combine the base classifiers' predictions. However, *i*Boost constructs multiple decision trees, one for each base classifier. Instead of selecting a single predictor, we take a weighted vote of the predictions. Further, their work is for combining heterogeneous classifiers produced by different inducers while *i*Boost is concerned about combining homogeneous classifiers obtained by bootstrap sampling (or reweighting) and using the same inducer.

4 Results

Improve Prediction Accuracy. At first we ran the algorithm through 32 UC Irvine data sets for classification problems over 10 iterations (See Table 1). Our *i*Boost algorithm beats AdaBoost on 19 data sets, loses to AdaBoost on 6 data sets and draws on 6 data sets. The average improvement based on 10-fold cross-validation is about 0.6% which is not extremely impressive. However, upon closer inspection, there are significant number of data sets where AdaBoost predicts with error less than 5% and probably very close to the actual noise rate. This leaves very little room for improvement. Nevertheless, on the 12 data sets where AdaBoost achieves accuracy in the 90% range (See Table 1(C)), we still manage to squeeze in an improvement of 0.31% after 100 iterations, with *i*Boost winning AdaBoost on 6 data sets and losing on 2 data sets.

In the 70^{+}% range, the improvement seems to be more noticeable. See Table 1. Among the 10 data sets, *i*Boost consistently improves over the performance of AdaBoost, except for one or two data sets. Further, the difference in their average performance widens from 0.80% at 10 iterations to 1.14% at 100 iterations. We perform Student's one-sided t-test on the improvements obtained at 50 iterations where AdaBoost has the best performance. The level of significance is tabulated in Table 2A which shows that for many data sets, the confidence that *i*Boost outperforms AdaBoost is near or above 90%. The improvements are slightly more significance if we allow the mixing rate η to varies in incremental of 0.1. Here, for 6 out of the 10 data sets, the best performance is still

Table 1. The performance of *i*Boost vs. AdaBoost. Those numbers in bold face indicate the winner between AdaBoost and *i*Boost with the same number of iterations

Data Sets in the 70^+% range								
	Iterations = 10		Iterations = 25		Iterations = 50		Iterations = 100	
Data set	Ada	*i*Boost	Ada	*i*Boost	Ada	*i*Boost	Ada	*i*Boost
A1 Balance-Scale	78.08	**78.24**	76.00	**77.60**	75.36	**76.80**	74.56	**76.16**
A2 Breast-Cancer	69.58	**72.38**	70.28	**73.78**	70.28	**74.81**	70.28	**74.48**
A3 Credit (G)	72.60	**74.90**	75.00	**75.70**	74.40	**76.00**	74.10	**74.70**
A4 Diabetes	73.31	**73.70**	73.70	**73.96**	72.79	**73.57**	73.57	**74.58**
A5 Glass	73.83	**74.77**	**76.17**	75.23	77.57	**78.04**	78.04	**79.44**
A6 Heart-C	**77.89**	77.23	78.55	**79.87**	78.55	**79.87**	79.54	**80.63**
A7 Heart-H	76.19	**76.87**	77.21	77.21	77.89	**78.57**	**78.23**	77.99
A8 Heart-Statlog	80.00	**80.37**	81.85	81.48	82.59	82.59	81.11	**81.95**
A9 Sonar	79.33	**80.29**	**84.62**	83.65	84.62	**85.10**	83.65	**84.72**
A10 Vehicle	76.48	**76.60**	**78.25**	77.54	**78.72**	78.37	**77.42**	77.31
Average	75.73	**76.53**	77.16	**77.60**	77.28	**78.37**	77.05	**78.19**
Win	1	9	4	5	1	8	2	8

Data Sets in the 80^+% range								
	Iterations = 10		Iterations = 25		Iterations = 50		Iterations = 100	
Data set	Ada	*i*Boost	Ada	*i*Boost	Ada	*i*Boost	Ada	*i*Boost
B1 Audiology	84.07	**84.96**	84.96	84.96	84.96	84.96	84.96	**85.50**
B2 Autos	82.44	**83.90**	84.39	**84.88**	85.37	**87.32**	86.34	**86.93**
B3 Horse Colic	83.97	83.97	82.07	**84.78**	81.79	**85.87**	81.79	**85.43**
B4 Credit (A)	84.93	**85.80**	86.38	**87.10**	85.94	**86.52**	87.10	**87.59**
B5 Heart-Statlog	80.00	**80.37**	**81.85**	81.48	82.59	**82.59**	81.11	**81.95**
B6 Hepatitis	**83.87**	81.94	**85.16**	81.94	84.52	**85.16**	**85.81**	85.46
B7 Lymphography	**83.78**	81.08	82.43	82.43	**84.46**	83.11	83.78	83.78
B8 Waveform	81.08	**81.54**	82.92	**84.06**	84.58	**84.60**	**84.94**	84.90
Average	**83.02**	82.94	83.39	**83.95**	84.28	**85.02**	84.48	**85.18**
Win	2	5	2	4	1	5	2	5

Data Sets in the 90^+% range								
	Iterations = 10		Iterations = 25		Iterations = 50		Iterations = 100	
Data set	Ada	*i*Boost	Ada	*i*Boost	Ada	*i*Boost	Ada	*i*Boost
C1 Anneal	99.67	**99.78**	99.67	99.67	99.67	99.67	99.67	99.67
C2 Breast-W	**95.85**	95.57	95.99	**96.28**	95.99	**96.28**	96.42	**96.57**
C3 Hypothyroid	99.73	99.73	99.68	99.68	99.71	99.71	99.71	99.71
C4 Ionosphere	93.73	93.73	**95.16**	94.59	**94.87**	94.30	94.87	94.87
C5 Iris	94.67	94.67	**94.67**	94.00	94.00	**94.67**	94.67	94.67
C6 Kr-Vs-Kp	**99.50**	99.47	**99.50**	99.47	99.53	99.53	99.47	**99.50**
C7 Segment	98.57	**98.61**	98.70	98.70	98.57	98.57	**98.79**	98.74
C8 Sick	**99.15**	99.07	**99.02**	98.99	**99.15**	99.05	**98.99**	98.91
C9 Soybean	92.83	**92.97**	92.39	**94.00**	92.53	**94.14**	92.39	**94.29**
C10 Vote	94.94	**95.17**	94.94	**95.86**	94.94	**95.40**	94.94	**95.40**
C11 Vowel	92.22	**93.03**	95.25	**96.06**	96.26	**96.57**	96.77	**97.17**
C12 Zoo	95.05	95.05	95.05	**96.04**	95.05	**96.04**	95.05	**96.04**
Average	96.33	**96.40**	96.67	**96.94**	96.69	**96.99**	96.81	**97.13**
Win	3	5	4	5	2	6	2	6

$\eta = 0.5$ (i.e., placing equal importance in both base classifiers and competency predictors.).

The relative performance of *i*Boost and AdaBoost on the 80^+% data sets is somewhat in between the performance of those 70^+% and 90^+% data sets (see Table 1(B)). Although the average improvements on the 80^+% and 90^+% data

Table 2. (A) Test of significance using Student t-test. The average improvement is computed based only on those data sets that *i*Boost performs better in both cases where $\eta = 0.5$ and η is optimized. These data sets are marked with an asterisk (B) Using exponential weighting scheme alone (i.e. $\eta = 1$) does not increase accuracy

	$\eta = 0.5$		best η		
data set	-Δerror	t-test	η	-Δerror	t-test
A1	1.44	99.2% *	0.5	1.44	99.2% *
A2	4.53	96.0% *	0.5	4.53	96.0% *
A3	1.60	99.6% *	0.5	1.60	99.6% *
A4	0.78	89.2% *	0.5	0.78	89.2% *
A5	0.47	69.8% *	0.1	0.93	82.7% *
A6	1.32	94.9% *	0.6	1.65	97.5% *
A7	0.68	65.0% *	0.5	0.66	65.0% *
A8	0.00	50%	0.7	0.37	83.0%
A9	0.48	62.2% *	0.5	0.48	62.2% *
A10	-0.35	25.0%	0.9	-0.12	38.8%
Average *	1.41	84.5%		1.51	86.0%

(A)

	accuracy	
data set	$\eta = 1$	$\eta = 0.5$
A1	75.36	**76.80**
A2	69.93	**74.81**
A3	74.28	**76.00**
A4	72.79	**73.57**
A5	77.10	**78.04**
A6	78.22	**79.87**
A7	77.89	**78.57**
A8	82.22	**82.59**
A9	85.10	85.10
A10	**78.61**	78.37
Average:	77.14	**78.37**

(B)

sets are smaller that that of the $70^{+}\%$ data set, the t-test results yield similar level of confidence [2]. Lastly, the improvement is also more pronounced if we consider only those binary class data sets[2].

Competency Predictors Help. It is possible that the improvements obtained by *i*Boost is not so much due to the use of competency predictor but rather to the usage of exponential weighting scheme. To show that this is not the case, we compare *i*Boost with boosting using exponential weighting scheme but without adjusting the weights using the competency predictors. We can do this by simply setting $\eta = 1$. Table 2(B) shows the results obtained for those data sets having 70% accuracy where *i*Boost continues to outperform. The result indicates that the improvement is due to the use of competency predictors.

Margin Distribution. The performance of AdaBoost has been studied in terms of the margins of the training examples [15]. The margin of an example (x, y) is defined to be

$$margin(x, y) = \frac{y \sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} |\alpha_t|}.$$

The magnitude of the margin can be interpreted as a measure of confidence in the prediction. It ranges from -1 to +1 and is positive if and only if AdaBoost predicts correctly. Schapire et. al. [15] proved that larger margins of the training set translate into a superior upper bound on the generalization error. Figure 3

[2] Details omitted due to page limitation. More details will be provided in a full version of this paper.

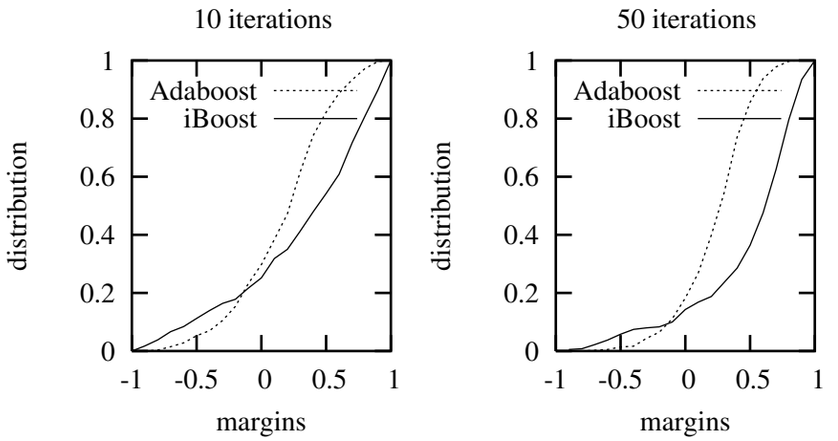

Fig. 3. The margin distributions of *i*Boost vs. AdaBoost on the Colic data set

shows the cumulative distributions of the margins for both AdaBoost and *i*Boost on the H. Colic data set at the 10th iterations and 50th iterations. The negative margins near zero tend to increase and many become positive. Further, The positive margins increase further towards one. We believe that this shift in margin gives rise to the improvement in accuracy. The observation is very typical on almost all data sets which *i*Boost performs well.

Minimize Overfitting. Optiz and Maclin [13] illustrated that AdaBoost suffers from overfitting in certain situations. They illustrated this by using artificial data sets with one-sided noise. The data sets are defined over two relevant attributes and four irrelevant attributes. The target concept is a simple linear halfspace based on the relevant attributes. Points are generated within the target concept with 10% of the points being mislabeled. They showed that AdaBoost often produced significant increase in error (i.e. overfit). Maclin's RegionBoost algorithm worsens this overfitting problem [12]. However, Our experiment (see figure 4) with 250 randomly generated data points indicates that *i*Boost tends not to overfit as badly. This is not surprising since Freund et. al. [4] had shown theoretically that the exponential weighting scheme reduces overfitting.

Bias-Variance Analysis. Another way to understand how *i*Boost improves prediction accuracy is to decompose the estimated error rate into the bias and variance components [2,9]. The bias component measures how closely the learning algorithm's average guess (over all possible training sets of the same size as the given labeled sample) matches the target, while the variance component measures how much the learning algorithm's guess fluctuates for different training sets of a given size. Experiments performed using Dietterich and Kong's definition[3] of bias and variance [10] suggest that most of *i*Boost's improvements are due to the reduction of the variance component, while the bias component

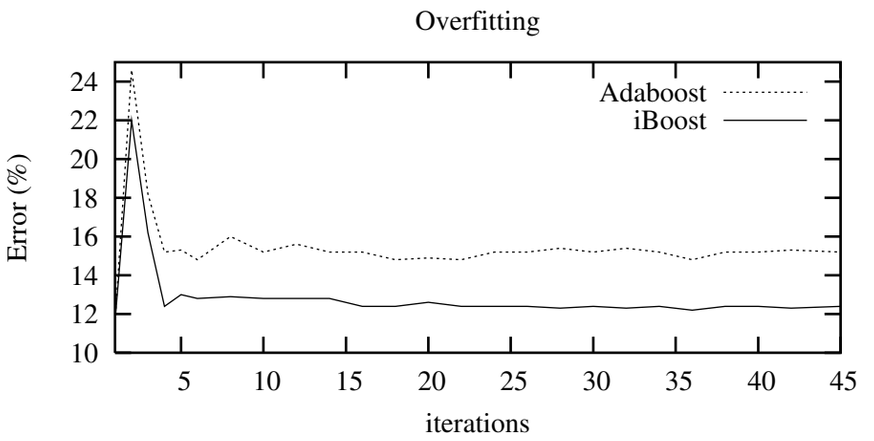

Fig. 4. *i*Boost seems to minimize overfitting

remain fairly stable. This is somewhat surprising since in AdaBoost, the bias component is much larger than the variance component. However, this is somewhat unfortunate since it places a limitation of how much *i*Boost can outperform AdaBoost. Due to page limitation, we shall defer our discussion to a later full version of this paper.

5 Conclusion

In this paper, we introduce a variant of boosting, *i*Boost, that employs the exponential weighting scheme and the use of competency predictors in predicting whether the base classifiers can be trusted. The latter allows us to adjust the weights of the base classifiers by taking into consideration of how confident we are with their predictions on the unlabeled test instance.

Although *i*Boost allows us to adjust the relative importance of the base classifiers and competency predictors, our experiments suggest that often it is best to place equal emphasis on both (i.e. setting $\eta = 0.5$). This also implies that the improvement is not solely due to the use of exponential weighting scheme (i.e. ignoring the competency predictors totally). However, due to the use of the exponential weighting scheme, which has been shown theoretically to reduce overfitting, *i*Boost tends to overfit less as compared to AdaBoost and RegionBoost. Thus, on the UCI benchmark data sets, *i*Boost exhibits a significant improvement in accuracy over AdaBoost. Bias-Variance decomposition of the error rates obtained by AdaBoost and *i*Boost suggests that the improvement of *i*Boost is due to the reduction of the variance component of the error rate. An inspection of the margin distributions on those data sets where *i*Boost performs

well indicate that *i*Boost does so by shifting the positive margins and negative margins close to zero towards 1.

Acknowledgement

This work is partially supported by NSF grant CCR-0208935. We like to thank the reviewers for valuable comments and references to some relevant work. We also want to thank Tom Bylander for helpful suggestions. This project would have taken a longer time to complete if not for the open source code of Weka.

References

1. L. Breiman. Bagging predictors. In *Machine Learning*, volume 24, pages 123–140, 1996. 245
2. T. G. Dietterich and E. B. Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University, 1995. 254
3. Y. Freund. Boosting a weak learning algorithm by majority. *Inform. Comput.*, 121(2):256–285, September 1995. Also appeared in COLT90. 246
4. Yoav Freund, Yishay Mansour, and Robert Schapire. Why averaging classifiers can protect against overfitting. In *Proc. of the 8th International Workshop on Artificial Intelligence and Statistics*, 2001. 248, 249, 251, 254
5. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996. 245, 246
6. David Helmbold, Stephen Kwek, and Leonard Pitt. Learning when to trust which experts. In *Computational Learning Theory: EuroColt '97*, pages 134–149. Springer-Verlag, 1997. 245, 246, 248, 249
7. Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. CBCL Paper 83, M. I. T. Center for Biological and Computational Learning, August 1993. 250
8. Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994. 246
9. Ron Kohavi and David H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In *Proc. 13th International Conference on Machine Learning*, pages 275–283. Morgan Kaufmann, 1996. 254
10. Eun Bae Kong and Thomas G. Dietterich. Error-correcting output coding corrects bias and variance. In *Proc. 12th International Conference on Machine Learning*, pages 313–321. Morgan Kaufmann, 1995. 254
11. N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inform. Comput.*, 108(2):212–261, 1994. 247
12. Richard Maclin. Boosting classifiers regionally. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 700–705, Menlo Park, July 26–30 1998. AAAI Press. 250, 251, 254
13. D. Optiz and R. Marlin. Popular ensemble methods: An empirical study. CBCL Paper UMD CS TR 98-1, University of Maryland, 19938. 254

14. Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990. 246
15. Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proc. 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1997. 253
16. Ljupčo Todorovski and Sašo Džeroski. Combining classifiers with meta decision trees. In *Machine Learning Journal*, 2002. to appear. 251
17. L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984. 246
18. I. H. Witten and E. Frank. Nuts and bolts: Machine learning algorithms in java,. In *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.*, pages 265–320. Morgan_Kaufmann, 2000. 249
19. D. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–260, 1992. 251