Using Validation Sets to Avoid Overfitting in AdaBoost *

Tom Bylander and **Lisa Tate**
Department of Computer Science, University of Texas at San Antonio,
San Antonio, TX 78249 USA
{bylander, ltate}@cs.utsa.edu

Abstract

AdaBoost is a well known, effective technique for increasing the accuracy of learning algorithms. However, it has the potential to overfit the training set because its objective is to minimize error on the training set. We demonstrate that overfitting in AdaBoost can be alleviated in a time-efficient manner using a combination of dagging and validation sets. Half of the training set is removed to form the validation set. The sequence of base classifiers, produced by AdaBoost from the training set, is applied to the validation set, creating a modified set of weights. The training and validation sets are switched, and a second pass is performed. The final classifier votes using both sets of weights. We show our algorithm has similar performance on standard datasets and improved performance when classification noise is added.

Introduction

AdaBoost was first introduced by Freund and Schapire (Freund & Schapire 1997). Since then AdaBoost has shown to do well with algorithms such as C4.5 (Quinlan 1996), Decision Stumps (Freund & Schapire 1996), and Naive Bayes (Elkan 1997). It has been shown that AdaBoost has the potential to overfit (Groves & Schuurmans 1998; Jiang 2000), although rarely with low noise data. However, it has a much higher potential to overfit in the presence of very noisy data (Dietterich 2000; Ratsch, Onoda, & Muller 2001; Servedio 2003). At each iteration, AdaBoost focuses on classifying the misclassified instances. This might result in fitting the noise during training. In this paper, we use validation sets to adjust the hypothesis of the AdaBoost algorithm to improve generalization, thereby alleviating overfitting and improving performance.

Validation sets have long been used in addressing the problem of overfitting with neural networks (Heskes 1997) and decision trees (Quinlan 1996). The basic concept is to apply the classifier to a set of instances distinct from the training set. The performance over the validation set can then be used to determine pruning, in the case of decision trees, or early stopping, in the case of neural networks.

*This research has been supported in part by the Center for Infrastructure Assurance and Security at the University of Texas at San Antonio.
Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

In both cases, the training set is used to find a small hypothesis space, and the validation set is used to select a hypothesis from that space. The justification is that in a large hypothesis space, minimizing training error will often result in overfitting. The training set can instead be used to establish a set of choices as training error is minimized, with the validation set used to reverse or modify those choices.

A validation set could simply be used for early stopping in AdaBoost. However, we obtain the most success by also performing *2-dagging* (Ting & Witten 1997), and modifying weights. Dagging is disjoint aggregation, where a training set is partitioned into subsets and training is performed on each set. Our algorithm partitions the original training set into 2 partitions, applies boosting on one partition and validation with the other partition. It switches the training and validation sets then applies boosting and validation again. In each boosting/validation pass, the sequence of weak hypotheses from boosting on the training partition are fit to the validation partition in the same order. The sequence is truncated if and when the average error of the training and validation set is 50% or more. The final weights are derived from averaging the error rates.

Our results show that a large improvement is due to the 2-dagging. When a training set contains classification noise, learning algorithms that are sensitive to noise and prone to overfitting can fit the noise. When the 2-dagging partitions the training set, a noisy example will be in only one of the subsets. Although the hypothesis created from that subset may overfit the noisy example, the hypothesis created from the other subset will not overfit that particular example. Aggregating the hypotheses counteracts the overfitting of each separate hypothesis. A further improvement is obtained by using the validation set to stop early and to modify the weights. While this might not be the best way to use validation sets for boosting, we do demonstrate that validation sets can be used to avoid overfitting efficiently.

Several techniques have been used to address overfitting of noisy data in AdaBoost. BrownBoost (Freund 1999), an adaptive version of Boost By Majority (Freund 1990), gives small weights to misclassified examples far from the margin. Using a "soft margin" (Ratsch, Onoda, & Muller 2001) effectively does not give preference to hypotheses that rely on a few examples with large weights from continuous misclassification. MadaBoost (Domingo & Watanabe 2000) pro-

vides an upper bound on example weights based on initial probability. SmoothBoost (Servedio 2003) smooths the distribution over the examples by weighting on the margin of error and creating an upper bound so that no one example can have too much weight. NadaBoost (Nakamura, Nomiya, & Uehara 2002) adjusts the weighting scheme by simply adding a threshold for the maximum allowable weight. In this paper, we address overfitting of noisy data by using a validation set to smooth the hypothesis weights.

The rest of this paper is organized as follows. First we describe the AdaBoost.M1 algorithm, used for multiclass datasets. We then present our AdaBoost.MV algorithm. Finally, we describe our experiments including a comparison with the BrownBoost algorithm and conclude with a discussion of the results.

AdaBoost.M1 Algorithm

Algorithm AdaBoost.M1
Input: train set of m examples $((x_1, y_1),...,(x_m, y_m))$
with labels $y_i \in Y = \{1, \ldots, k\}$
learning algorithm L
integer T specifying number of iterations
Initialize $D_1(i) = \frac{1}{m}$ for all i
Do for $t = 1, \ldots, T$

1. Call L providing it with the distribution D_t
2. Get back a hypothesis h_t: $X \rightarrow Y$
3. Calculate the error of h_t: $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
4. if $\epsilon_t \geq 1/2$, then set $T = t - 1$ and abort
5. Set $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$
6. Update distribution D_t:

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

where Z_t is a normalization constant.

Output the final classifier:

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$$

Figure 1: The AdaBoost.M1 algorithm

AdaBoost.M1 is a version of the AdaBoost algorithm (Freund & Schapire 1997) that handles multiple-label classification. The AdaBoost.M1 algorithm takes a set of training examples $((x_1, y_1),...,(x_m, y_m))$, where $x_i \in X$ is an instance and $y_i \in Y$ is the class label. It initializes the first distribution of weights D_1 to be uniform over the training examples. It calls a learning algorithm, referred to as L, iteratively for a predetermined T times. At each iteration, L is provided with the distribution D_t and returns a hypothesis h_t relative to the distribution. It calculates the error ϵ_t with respect to the distribution D_t and the hypothesis h_t. The β_t value, a function of the error, is used to update the weights in the distribution at every iteration t. If an example is misclassified it receives a higher weight. The Z_t value is the normalization factor. The β_t value is also used to calculate the final weight for each hypothesis h_t as $\log(1/\beta_t)$. The final hypothesis $h_{fin}(x)$ is a weighted plurality vote determined by the weights of the hypotheses.

Validation Algorithms

FitValidationSet Subroutine

Subroutine FitValidationSet
Input: validation set of m examples $((x_1, y_1),...,(x_m, y_m))$
with labels $y_i \in Y = \{1, \ldots, k\}$
hypotheses $\{h_1, \ldots, h_T\}$ from **AdaBoost.M1**
errors $\{\epsilon_1, \ldots, \epsilon_T\}$ from **AdaBoost.M1**
Initialize $D'_1(i) = \frac{1}{m}$ for all i
Do for $t = 1, \ldots, T$

1. Calculate the error of h_t: $\epsilon'_t = \sum_{i:h_t(x_i) \neq y_i} D'_t(i)$
2. if $(\epsilon_t + \epsilon'_t)/2 \geq 1/2$, then set $T = t - 1$ and abort
3. Set $\beta'_t = \frac{\epsilon'_t}{1 - \epsilon'_t}$
4. Update distribution D'_t:

$$D'_{t+1}(i) = \frac{D'_t(i)}{Z'_t} \times \begin{cases} \beta'_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

where $Z'_t = \sum D'_t(i)$

Output the errors ϵ'_t

Figure 2: The FitValidationSet algorithm

The inputs to the **FitValidationSet** subroutine (Figure 2) are the validation set, the classifiers from boosting on the training set, and the errors from boosting on the training set. FitValidationSet applies AdaBoost.M1's distribution updating scheme to the classifiers in sequential order. The first distribution D'_1 is initialized to be uniform. The validation error ϵ'_t is calculated for each classifier h_t with respect to distribution D'_t. The next distribution D'_{t+1} is determined using the same distribution update as AdaBoost.M1. FitValidationSet will stop early if $(\epsilon_t + \epsilon'_t)/2 \geq 1/2$, where ϵ_t is the error of h_t on the distribution D_t from AdaBoost.M1. In our algorithm, the validation set is used to adjust the weights away from fitting (maybe overfitting) the training set toward fitting the validation set. There are a variety of other reasonable sounding ways to use a validation set in boosting. Informally, our algorithm can be justified as follows.

We don't want to overfit either the training set or the validation set. We also don't want to forget the weights learned from either set, so our algorithm chooses a weight in between the two. For the method of adjustment, averaging the errors followed by computing a weight is more stable than computing two beta values or two weights and averaging them. It is natural to stop when the average error is 50% or

more. Stopping when either error is 50% is another possibility, but does not work quite as well empirically. As we mentioned earlier, we do not claim this is the best way to use a validation set, but we do show that a validation set can be efficiently and effectively applied.

AdaBoost.MV Algorithm

Algorithm AdaBoost.MV
Input: train set of m examples ((x_1, y_1),...,(x_m, y_m))
with labels $y_i \in Y = \{1, \ldots, k\}$
learning algorithm L
integer T specifying number of iterations
Divide the training set into two equally sized sets
N (train set) and V (validation set)
by performing a stratified, randomized split.
Do for k = 1 to 2

1. Call **AdaBoost.M1** providing it N, L, and T
2. Set $\{h_{k,1}, \ldots, h_{k,T}\}$to the classifiers from **AdaBoost.M1**
3. Set $\{\epsilon_{k,1}, \ldots, \epsilon_{k,T}\}$ to the errors from **AdaBoost.M1**.
4. Call **FitToValidate** providing it V, classifiers $\{h_{k,1}, \ldots, h_{k,T}\}$, and errors $\{\epsilon_{k,1}, \ldots, \epsilon_{k,T}\}$
5. Set $\{\epsilon'_{k,1}, \ldots, \epsilon'_{k,T}\}$ to the errors from **FitValidationSet**.
6. Set $\epsilon_{k,t}$ to $(\epsilon_{k,t} + \epsilon'_{k,t})/2$ for $t \in \{1, \ldots, T\}$
7. Exchange $N \leftrightarrow V$

Output the final classifier:

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{k=1}^{2} \sum_{t:h_t(x)=y} \log \frac{1 - \epsilon_{k,t}}{\epsilon_{k,t}}$$

Figure 3: The AdaBoost.MV algorithm

The **AdaBoost.MV** algorithm (Figure 3) inputs a set of training examples, a weak learning algorithm L, and a parameter T specifying the number of boosting iterations. It first performs a randomized, stratified split of the examples into two equally sized sets: a training set N and validation set V. The algorithm then performs two passes.

The first pass calls AdaBoost.M1 with the training set N, obtaining the classifiers and the error rates. Next, it calls FitValidationSet with the validation set V and the classifiers, obtaining another set of error rates. The weight for each classifier is calculated from the average of the corresponding two error rates. After N and V are switched, the second pass performs the same sequence of actions. If AdaBoost.M1 and FitValidationSet do not abort early, then there will be $2T$ base classifiers in the final hypothesis.

The two passes allow each example to be used once for boosting and once for validation. More than two passes could be performed by doing more than two folds or doing two folds multiple times. However, we did not want to do more work than the original AdaBoost.M1, and we wanted the number of base classifiers and boosting iterations to be as similar to AdaBoost.M1 as possible.

Table 1: Datasets: UCI Repository

Datasets	Examples	Classes	Attributes
anneal	798	6	38
balance	625	3	5
breast cancer	286	2	9
breast_w	698	2	9
horse colic	368	2	24
credit_a	690	2	15
credit_g	1000	2	20
diabetes	768	2	8
heart_c	303	5	14
heart_h	294	5	14
hepatitis	155	2	20
hypothyroid	3772	4	30
ionosphere	351	2	34
iris	150	3	4
krvskp	3196	2	36
labor	57	2	16
lymph	148	4	19
mushroom	8124	2	23
sick	3772	2	30
sonar	207	2	61
splice	3190	3	62
vote	435	2	16
waveform	5000	3	40

Experiments and Results

We compared AdaBoost.MV to AdaBoost.M1 and BrownBoost using stratified 10-fold cross-validation. We implemented AdaBoost.MV and BrownBoost in Weka (Witten & Frank 2000) and compared them to each other as well as AdaBoost.M1. Experiments were run with three different base classifiers: Decision Stump, C4.5, and Naive Bayes. We used 23 benchmark datasets from the UCI repository (Blake & Merz 1998). For each dataset, Table 1 lists the number of examples, the number of classes, and the number of attributes. Each combination was used to generate 100 classifiers. Because all three algorithms can stop early, the number of classifiers actually generated is sometimes less than the number requested, especially when created with the Naive Bayes classifier.

For each pair of algorithms, we count the number of times each algorithm had a lower error rate and also determined the relative error reduction. Table 2 provides a summary of these results.

We do not have the space to fully describe the BrownBoost algorithm, but we describe the choices made in our implementation. Our version of BrownBoost searches for the largest value of c (to 0.1 accuracy) that results in exiting the BrownBoost loop before 100 base classifiers are generated. We did not implement confidence-rated predictions.

Table 2: Summary of Results

Noise Added	Base Classifiers	MV vs. M1: Better - Worse - Tie	MV vs. BB: Better - Worse - Tie	BB vs. M1: Better - Worse - Tie
No	Decision Stump	12-10-1	12-9-2	10-6-7
	Naive Bayes	14-7-2	14-7-2	12-8-3
	C4.5	12-10-1	14-8-1	4-16-3
10%	Decision Stump	13-6-4	9-10-4	13-6-4
	Naive Bayes	17-6-0	16-6-1	10-6-7
	C4.5	18-2-3	22-1-0	7-12-4
20%	Decision Stump	11-11-1	14-7-2	9-6-8
	Naive Bayes	18-5-0	17-6-0	11-6-6
	C4.5	18-5-0	18-4-1	14-7-2
Noise Added	**Base Classifiers**	**MV vs. M1: Average Error Reduction**	**MV vs. BB: Average Error Reduction**	**BB vs. M1: Average Error Reduction**
No	Decision Stump	2.85%	2.96%	-0.12%
	Naive Bayes	8.98%	8.15%	0.90%
	C4.5	2.94%	10.21%	-8.09%
10%	Decision Stump	2.65%	0.65%	2.65%
	Naive Bayes	11.39%	8.94%	2.69%
	C4.5	18.50%	21.04%	-3.22%
20%	Decision Stump	5.11%	1.55%	2.81%
	Naive Bayes	13.26%	12.67%	0.67%
	C4.5	21.94%	22.19%	-0.32%

For multi-class problems, we simply used "steps" of -1 and 1 for incorrectly and correctly classified examples, respectively.

We also ran experiments with 10% and 20% classification noise in the training set. Additional experiments were performed to determine how much of the improvement is due to dagging vs. weight adjustment.

Without Noise Added

Figure 4 shows scatter plots of the error rates of all 23 datasets for each type of base classifier comparing AdaBoost.M1 and AdaBoost.MV. Each point corresponds to one dataset. The points below the diagonal line are when the error rate is higher for AdaBoost.M1 than AdaBoost.MV. T was set to 100 for AdaBoost.M1 and set to 50 for AdaBoost.MV (both result in up to 100 base classifiers). No noise was added to the datasets. The figure shows that when using Decision Stumps as our base classifier, AdaBoost.MV does as well as AdaBoost.M1.

In this case (see Table 2), AdaBoost.MV performed better than AdaBoost.M1 on 12 of the datasets, worse on 10 datasets, and tied on 1 dataset (12-10-1). AdaBoost.MV does similar using C4.5 (12-10-1) and better using Naive Bayes (14-7-2). Compared with AdaBoost.M1, AdaBoost.MV has a relative error reduction of about 3% with Decision Stumps, 9% with Naive Bayes, and 3% with C4.5 on average.

When we compare AdaBoost.MV with BrownBoost, AdaBoost.MV performs at least as well as BrownBoost with Decision Stumps (12-9-2), Naive Bayes (14-7-2) and C4.5 (14-8-1). However, when looking at the average error reduction, AdaBoost.MV reduces the error of BrownBoost by about 3% with Decision Stumps, 8% with Naive Bayes, and 10% with C4.5.

With Noise Added

When noise is added to the training set using the AdaBoost.M1 algorithm, there is an increased potential for overfitting. At each iteration AdaBoost.M1 focuses on misclassified examples, which are typically the added noise.

When 10% classification noise is added to the training sets (no noise in test sets), AdaBoost.MV performs better than AdaBoost.M1 with Decision Stumps (13-6-4, error reduction 3%) and Naive Bayes (17-6-0, error reduction 11%). Where we see the most improvement is with C4.5. AdaBoost.MV not only improves the error for 18 of the 23 datasets (18-2-3), but also the relative error reduction is 18% on average. Figure 5 shows a scatter plot of these results.

When 20% classification noise is added to the training sets, AdaBoost.MV with Decision Stumps does not clearly outperform AdaBoost.M1 (11-11-1, error reduction 5%). However, AdaBoost.MV does much better with Naive Bayes (18-5-0, error reduction 13%) and C4.5 (18-5-0, error reduction 22%).

We also compared AdaBoost.MV with BrownBoost using 10% and 20% classification noise. We found that AdaBoost.MV with Decision Stumps did as well as BrownBoost with 10% noise (9-10-4, error reduction 1%) and 20% noise (14-7-2, error reduction 2%). With Naive Bayes, AdaBoost.MV showed an improvement with both 10% noise (16-6-1, error reduction 9%) and 20% noise (17-6-0, error reduction 13%). Where we see the biggest difference is with

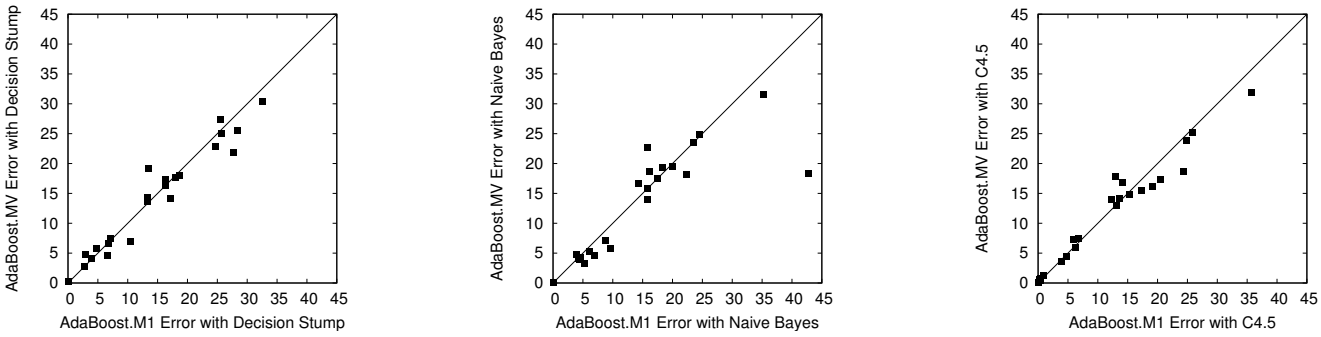

Figure 4: Error of AdaBoost.MV vs. AdaBoost.M1 run with 100 classifiers, using Decision Stumps (*left*), Naive Bayes (*middle*), and C4.5 (*right*).

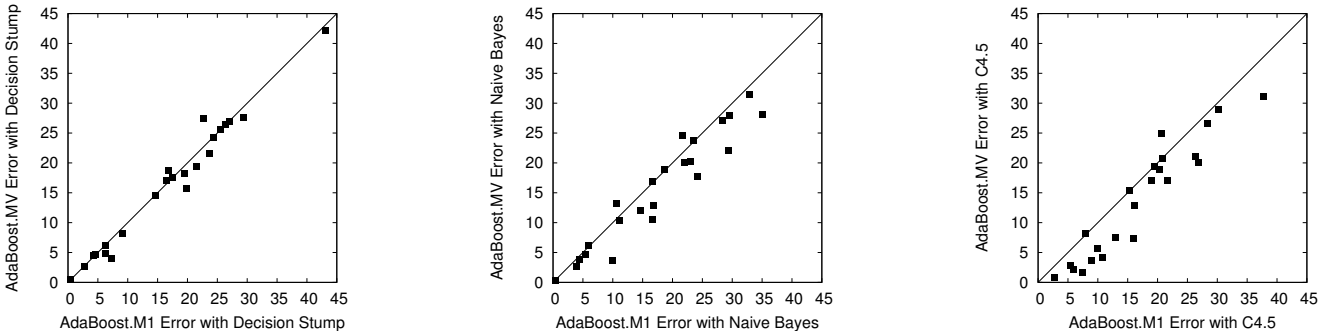

Figure 5: Error of AdaBoost.MV vs. AdaBoost.M1 run with 100 classifiers with 10% noise added, using Decision Stumps (*left*), Naive Bayes (*middle*), and C4.5 (*right*).

C4.5. AdaBoost.MV performs much better than BrownBoost with 10% noise (22-1-0, error reduction 21%) and 20% noise (18-4-1, error reduction 22%).

The Effects of 2-Dagging and Early Stopping

As previously mentioned, a gain in accuracy might be achieved simply by early stopping. In this set of experiments, we determined which elements of AdaBoost.MV led to the performance gains. We compared three versions of AdaBoost.MV:

1. 2-Dagging is used, but not early stopping or weight adjustment. This is implemented by skipping steps 4-6 of the AdaBoost.MV algorithm in Figure 3.
2. 2-Dagging and early stopping are used, but not weight adjustment. The FitToValidate subroutine is used only to determine the number of classifiers.
3. The full AdaBoost.MV algorithm is used, with dagging, early stopping, and weight adjustment.

The relative error reduction of these versions vs. AdaBoost.M1 was determined over the 23 datasets for the three types of base classifiers, with and without 10% classification noise. Figure 6 shows bar graphs of average error reduction when no noise is added (left) and 10% classification noise is added (right).

These bar graphs show that a large part of the error reduction is due to dagging alone, though it leads to a performance decrease for Decision Stumps with noise. Adding early stopping generally leads to a small performance decrease. However, AdaBoost.MV, which performs the combination of dagging, early stopping, and weight adjustment consistently provides an increase in performance over the two other versions.

Conclusion

Our results (Table 2) show that AdaBoost.MV alleviates some of the overfitting by AdaBoost.M1. It is evident that there is more improvement in performance with more complex base classifiers. There is little or no improvement with Decision Stumps, but a large improvement with C4.5. Comparing AdaBoost.MV with BrownBoost, the same pattern occurs with the largest improvement using C4.5.

An important aspect of our experiments is that the improvement is achieved using the same number of base classifiers. Because each base classifier in AdaBoost.MV is trained on half of the training data, we expect AdaBoost.MV to be more efficient than AdaBoost.M1. Timings of our experiments showed this to be true. For example, in the experiments run using C4.5 with 10% noise added to the training sets, the average runtime for BrownBoost is about 5 times greater than the runtime for AdaBoost.MV, 8 to 10 times greater on some large datasets such as mushroom and sick. AdaBoost.M1 has a runtime of about 2 to 3 times greater

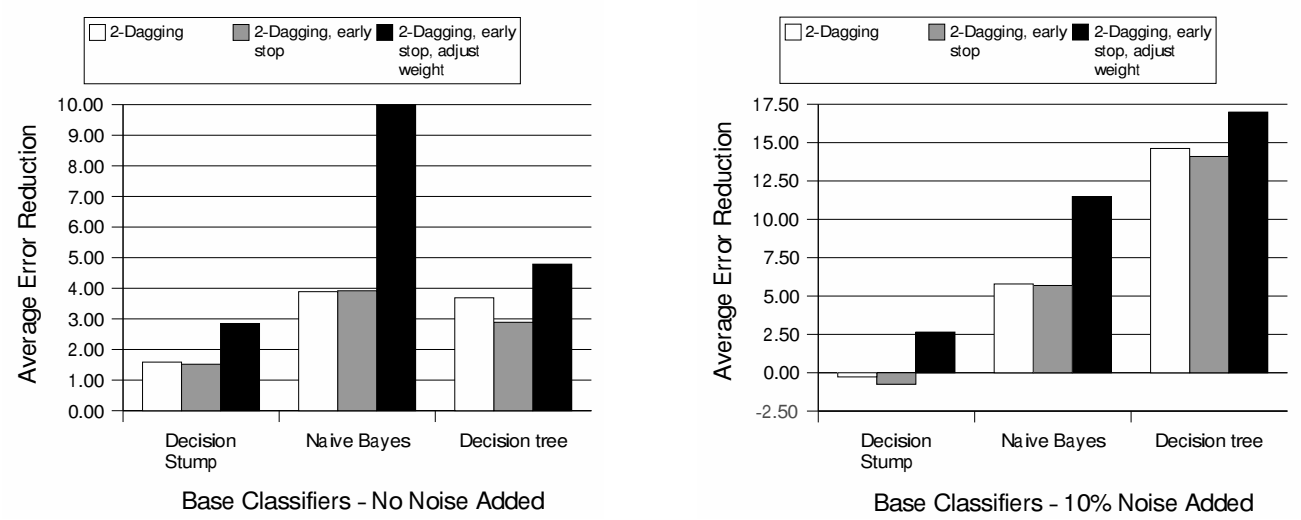

Figure 6: Average error reduction by using AdaBoost.MV with dagging only, dagging and early stopping, and dagging, early stopping, and adjusting the weights with the validation set, no noise (*left*), 10% classification noise (*right*)

than AdaBoost.MV.

In the future, we will continue to look for ways to better use our techniques. We will also look for ways to apply dagging and validation to other learning algorithms.

References

Blake, C. L., and Merz, C. J. 1998. UCI repository of machine learning databases.

Dietterich, T. 2000. Ensemble methods in machine learning. *Lecture Notes in Computer Science* 1857:1–15.

Domingo, C., and Watanabe, O. 2000. Scaling up a boosting-based learner via adaptive sampling. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 317–328.

Elkan, C. 1997. Boosting and naive bayesian learning. Technical report, Department of Computer Science and Engineering, University of California, San Diego, CA.

Freund, Y., and Schapire, R. E. 1996. Experiments with a new boosting algorithm. In *Proc. Thirteenth Int. Conf. on Machine Learning*, 148–156.

Freund, Y., and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. of Computer and System Sciences* 55(1):119–139.

Freund, Y. 1990. Boosting a weak learning algorithm by majority. In *COLT: Proceedings of the Workshop on Computational Learning Theory*.

Freund, Y. 1999. An adaptive version of the boost by majority algorithm. In *COLT: Proceedings of the Workshop on Computational Learning Theory*.

Groves, A. J., and Schuurmans, D. 1998. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proc. Fifteenth Int. Conf. on Artificial Intelligence*, 692–699.

Heskes, T. 1997. Balancing between bagging and bumping. In *Advances in Neural Information Processing Systems*, 466.

Jiang, W. 2000. Process consistency for adaboost. Technical report, Department of Statistics, Northwestern University.

Nakamura, M.; Nomiya, H.; and Uehara, K. 2002. Improvement of boosting algorithm by modifying the weighting rule. In *Annals of Mathematics and Artificail Intelligence*, volume 41, 95–109.

Quinlan, J. R. 1996. Bagging, boosting, and C4.5. In *Proc. Thirteenth National Conf. on Artificial Intelligence*, 725–730.

Ratsch, G.; Onoda, T.; and Muller, K. R. 2001. Soft margins for adaboost. *J. of Machine Learning* 42(3):287–320.

Servedio, R. A. 2003. Smooth boosting and learning with malicious noise. *J. of Machine Learning Research* 4:633–648.

Ting, K. M., and Witten, I. H. 1997. Stacking bagged and dagged models. In *Proc. 14th International Conference on Machine Learning*, 367–375. Morgan Kaufmann.

Witten, I. H., and Frank, E. 2000. *Data Mining: Practical machine learning tools with Jave implementations*. San Francisco, California: Morgan Kaufmann.