Pruning Adaptive Boosting

*** ICML-97 Final Draft ***

Dragos D.Margineantu
Department of Computer Science
Oregon State University
Corvallis, OR 97331-3202
margindr@cs.orst.edu

Thomas G. Dietterich
Department of Computer Science
Oregon State University,
Corvallis, OR 97331-3202
tgd@cs.orst.edu

Abstract

The boosting algorithm AdaBoost, developed by Freund and Schapire, has exhibited outstanding performance on several benchmark problems when using C4.5 as the "weak" algorithm to be "boosted." Like other ensemble learning approaches, AdaBoost constructs a composite hypothesis by voting many individual hypotheses. In practice, the large amount of memory required to store these hypotheses can make ensemble methods hard to deploy in applications. This paper shows that by selecting a subset of the hypotheses, it is possible to obtain nearly the same levels of performance as the entire set. The results also provide some insight into the behavior of AdaBoost.

1 Introduction

The adaptive boosting algorithm AdaBoost (Freund & Schapire, 1995) in combination with the decision-tree algorithm C4.5 (Quinlan, 1993) has been shown to be a very accurate learning procedure (Freund & Schapire, 1996; Quinlan, 1996; Breiman, 1996b). Like all ensemble methods, AdaBoost works by generating a set of classifiers and then voting them to classify test examples. In the case of AdaBoost, the various classifiers are constructed sequentially by focusing the underlying learning algorithm (e.g., C4.5) on those training examples that have been misclassified by previous classifiers.

The effectiveness of such methods depends on constructing a diverse, yet accurate, collection of classifiers. If each classifier is accurate and yet the various classifiers disagree with each other, then the uncorrelated errors of the different classifiers will be removed by the voting process. AdaBoost appears to be especially effective at generating such collections of classifiers. We should expect, however, that there is an accuracy/diversity tradeoff. The more accurate two classifiers become, the less they can disagree with each other.

A drawback of ensemble methods is that deploying them in a real application requires a large amount of memory to store all of the classifiers. For example, in the Frey-Slate letter recognition task, it is possible to achieve very good generalization accuracy by voting 200 trees. However, each tree requires 295 Kbytes of memory, so an ensemble of 200 trees requires 59 Mbytes. Similarly, in an application of error-correcting output coding to the NETtalk task (Bakiri, 1991), an ensemble based on 127 decision trees requires 1.3 Mbytes while storing the 20,003-word dictionary itself requires only 590Kbytes, so the ensemble is much bigger than the data set from which it was constructed. This makes it very difficult to convince customers that they should use ensemble methods in place of simple dictionary lookup, especially compared to classifiers based on nearest-neighbor methods, which can also perform very well.

This paper addresses the question of whether all of the decision trees constructed by AdaBoost are essential for its performance. Can we discard some of those trees and still obtain the same high level of performance? We call this "pruning the ensemble." We introduce five different pruning algorithms and compare their performance on a collection of ten domains. The results show that in the majority of domains, the ensemble of decision trees produced by AdaBoost can be pruned quite substantially without seriously decreasing performance. In several cases, pruning even improves the performance of the ensemble. This suggests that pruning should be considered in any application of AdaBoost.

The remainder of this paper is organized as follows. First, we describe the AdaBoost algorithm. Then we introduce our five pruning algorithms and the experiments we performed with them. The paper concludes with a discussion of the results of the experiments.

Table 1: The AdaBoost.M1 algorithm. The formula $[\![E]\!]$ is 1 if E is true and 0 otherwise.

Input: a set S, of m labeled examples:
$S = < (x_i, y_i), i = 1, 2, \ldots, m >$,
labels $y_i \in Y = \{1, \ldots, k\}$
WeakLearn (a weak learner)
a constant T.

[1] **initialize** $w_1(i) = 1/m \ \forall\, i$
[2] **for** $t = 1$ **to** T **do**
[3] $\quad p_t(i) = w_t(i)/(\sum_i w_t(i)) \ \forall\, i$;
[4] $\quad h_t :=$ WeakLearn(p_t);
[5] $\quad \epsilon_t = \sum_i p_t(i)[\![h_t(x_i) \neq y_i]\!]$;
[7] $\quad$ **if** $\epsilon_t > 1/2$ **then**
$\qquad$ *restart with uniform weights*
[8] $\qquad w_t(i) = 1/m \ \forall\, i$
[9] $\qquad$ **goto** [3]
[10] $\quad \beta_t = \epsilon_t/(1 - \epsilon_t)$;
[11] $\quad w_{t+1}(i) = w_t(i)\beta_t^{1-[\![h_t(x_i)\neq y_i]\!]} \ \forall\, i$

Output: $h_f(x) = \operatorname*{argmax}_{y \in Y} \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) [\![h_t(x) = y]\!]$

2 The AdaBoost algorithm

Table 1 shows the AdaBoost.M1 algorithm. The algorithm maintains a probability distribution w over the training examples. This distribution is initially uniform. The algorithm proceeds in a series of T trials. In each trial, a sample of size m (the size of the training set) is drawn with replacement according to the current probability distribution. This sample is then given to the inner (weak) learning algorithm (in this case C4.5 Release 8 with pruning). The resulting classifier is applied to classify each example in the training set, and the training set probabilities are updated to reduce the probability for correctly-classified examples and increase the probability for misclassified examples. A classifier weight β is computed (for each trial), which is used in the final weighted vote. As recommended by Breiman (1996a), if a classifier has an error rate greater than 1/2 in a trial, then we reset the training set weights to the uniform distribution and continue drawing samples.

3 Pruning methods for AdaBoost

We define a pruning method as a procedure that takes as input a training set, the AdaBoost algorithm (including a weak learner), and a maximum memory size for the learned ensemble of classifiers. The goal of each pruning method will be to construct the best possible ensemble that uses no more than this maximum permitted amount of memory. In practice, we will specify the amount of memory in terms of the maximum number, M, of classifiers permitted in the ensemble. We have developed and implemented five methods for pruning AdaBoost ensembles. We describe them in order of increasing complexity. In any case where we compute the voted result of a subset of the classifiers produced by AdaBoost, we always take a weighted vote using the weights computed by AdaBoost.

3.1 Early Stopping

The most obvious approach is to use the first M classifiers constructed by AdaBoost. It may be, however, that classifiers produced later in the AdaBoost process are more useful for voting. Hence, the performance of early stopping will be a measure of the extent to which AdaBoost always finds the best next classifier to add to its ensemble at each step.

3.2 KL-divergence Pruning

A second strategy is to assume that all of the classifiers have similar accuracy and to focus on choosing diverse classifiers. A simple way of trying to find diverse classifiers is to focus on classifiers that were trained on very different probability distributions over the training data.

A natural measure of the distance between two probability distributions is the Kullback-Leibler Divergence (KL-distance; Cover & Thomas, 1991). The KL distance between two probability distributions p and q is

$$D(p||q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}.$$

For each pair of classifiers h_i and h_j, we can compute the KL-distance between the corresponding probability distributions p_i and p_j computed in line 3 of AdaBoost (Table 1).

Ideally, we would find the set U of M classifiers whose total summed pairwise KL-distance is maximized:

$$J(U) = \sum_{i,j \in U; i<j} D(p_i||p_j).$$

Because of the computational cost, we use a greedy algorithm to approximate this. The greedy algorithm begins with a set containing the first classifier constructed by AdaBoost: $U = \{h_1\}$. It then iteratively adds to U the classifier h_i that would most increase $J(U)$. This is repeated until U contains M classifiers.

3.3 Kappa Pruning

Another way of choosing diverse classifiers is to measure how much their classification decisions differ. Statisticians have developed several measures of agreement (or disagreement) between classifiers. The most widely used measure is the Kappa statistic, κ (Cohen, 1960; Agresti, 1990; Bishop, Fienberg, & Holland, 1975). It is defined as follows.

Given two classifiers h_a and h_b and a data set containing m examples, we can construct a contingency table where cell C_{ij} contains the number of examples x for which $h_a(x) = i$ and $h_b(x) = j$. If h_a and h_b are identical on the data set, then all non-zero counts will appear along the diagonal. If h_a and h_b are very different, then there should be a large number of counts off the diagonal. Define

$$\Theta_1 = \frac{\sum_{i=1}^{L} C_{ii}}{m}$$

to be the probability that the two classifiers agree (this is just the sum of the diagonal elements divided by m).

We could use Θ_1 as a measure of agreement. However, a difficulty with Θ_1 is that in problems where one class is much more common than the others, all reasonable classifiers will tend to agree with one another, simply by chance, so all pairs of classifiers will obtain high values for Θ_1. We would like our measure of agreement to be high only for classifiers that agree with each other much more than we would expect from random agreements.

To correct for this, define

$$\Theta_2 = \sum_{i=1}^{L} \left(\sum_{j=1}^{L} \frac{C_{ij}}{m} \cdot \sum_{j=1}^{L} \frac{C_{ji}}{m} \right)$$

to be the probability that the two classifiers agree by chance, given the observed counts in the table.

Then, the κ statistic is defined as follows:

$$\kappa = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2}.$$

$\kappa = 0$ when the agreement of the two classifiers equals that expected by chance, and $\kappa = 1$ when the two classifiers agree on every example. Negative values occur when agreement is weaker than expected by chance, but this rarely happens.

Our Kappa pruning algorithm operates as follows. For each pair of classifiers produced by AdaBoost, we compute κ on the training set. We then choose pairs of classifiers starting with the pair that has the lowest κ and considering them in increasing order of κ until we have M classifiers. Ties are broken arbitrarily.

3.4 Kappa-Error Convex Hull Pruning

The fourth pruning technique that we have developed attempts to take into account both the accuracy and the diversity of the classifiers constructed by AdaBoost. It is based on a plot that we call the Kappa-Error Diagram. The left part of Figure 1 shows an example of a Kappa-Error diagram for AdaBoost on the Expf domain. The Kappa-Error diagram is a scatterplot where each point corresponds to a pair of classifiers. The x coordinate of the pair is the value of κ for the two classifiers. The y coordinate of the pair is the average of their error rates. Both κ and the error rates are measured on the training data set.

The Kappa-Error diagram allows us to visualize the ensemble of classifiers produced by AdaBoost. In the left part of Figure 1, we see that the pairs of classifiers form a diagonal cloud that illustrates the accuracy/diversity tradeoff. The classifiers at the lower right are very accurate but also very similar to one another. The classifiers at the upper left have higher error rates, but they are also very different from one another.

It is interesting to compare this diagram with a Kappa-Error diagram for Breiman's bagging procedure (also applied to C4.5; see the right part of Figure 1). Bagging is similar to AdaBoost, except that the weights on the training examples are not modified in each iteration; they are always the uniform distribution so that each training set is a bootstrap replicate of the original training set. The right part of Figure 1 shows that the classifiers produced by bagging form a much tighter cluster than they do with AdaBoost. This is to be expected, of course, because each classifier is trained on a sample drawn from the same, uniform distribution. This explains visually why AdaBoost typically out-performs bagging: AdaBoost produces a more diverse set of classifiers. In most cases, the lower accuracy of those classifiers is evidentally compensated for by the improved diversity (and by the lower weight given to low-accuracy hypotheses in the weighted vote of AdaBoost).

How can we use the Kappa-Error diagram for pruning? One idea is to construct the convex hull of the points in the diagram. The convex hull can be viewed as a "summary" of the entire diagram, and it includes both the most accurate classifiers and the most diverse pairs of classifiers. We form the set U of classifiers by taking any classifier that appears in a classifier-pair corresponding to a point on the convex hull. The drawback of this approach is that we cannot adjust the size of U to match the desired maximum memory target M. Nonetheless, this strategy explicitly considers both accuracy and diversity in choosing its classifiers.

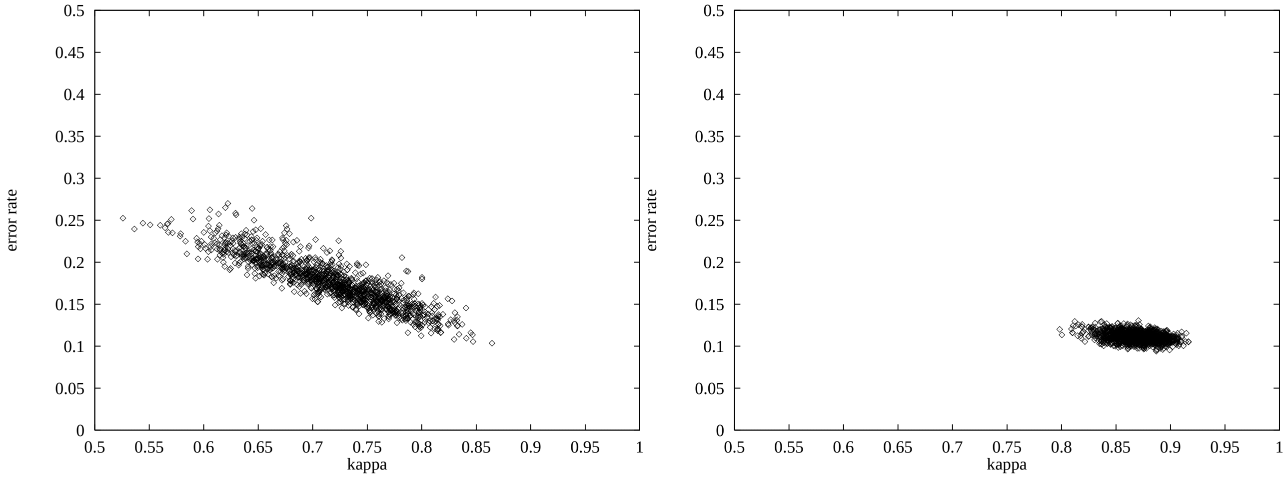

Figure 1: Kappa-Error diagrams for ADABOOST (left) and bagging (right) on the Expf domain.

3.5 Reduce-Error Pruning with Backfitting

The four methods we have discussed so far are each able to operate using only the training set. However, the last method requires that we subdivide the training set into a pruning set and a sub-training set. We train ADABOOST on the sub-training set and then use the pruning set to choose which M classifiers to keep. Reduce-Error Pruning is inspired by the decision-tree pruning algorithm of the same name. Our goal is to choose the set of M classifiers that give the best voted performance on the pruning set.

We could use a greedy algorithm to approximate this, but we decided to use a more sophisticated search method called backfitting (Friedman & Stuetzle, 1981). Backfitting proceeds as follows. Like a simple greedy algorithm, it is a procedure for constructing a set U of classifiers by growing U one classifier at a time. The first two steps are identical to the greedy algorithm. We initialize U to contain the one classifier h_i that has the lowest error on the pruning set (this is usually h_1, the first classifier produced by ADABOOST). We then add the classifier h_j such that the voted combination of h_i and h_j has the lowest pruning set error.

The differences between backfitting and the greedy algorithm become clear on the third iteration. At this point, backfitting adds to U the classifier h_k such that the voted combination of all classifiers in U has the lowest pruning set error. However, it then revisits each of its earlier decisions. First, it deletes h_i from U and replaces it with the classifier $h_{i'}$ such that the voted combination of $h_{i'}$, h_j, and h_k has lowest pruning set error. It then does the same thing with h_j. And then with h_k. This process of deleting previously-chosen classifiers and replacing them with the best classifier (chosen greedily) continues until none of the classifiers changes or until a limit on the number of iterations is reached. We employed a limit of 100 iterations.

In general, then, backfitting proceeds by (a) taking a greedy step to expand U, and (b) iteratively deleting each element from U and taking a greedy step to replace it until the elements of U converge. Then it takes another greedy step to expand U. This continues until U contains M classifiers.

4 Experiments and Results

We tested these five pruning techniques on ten data sets (see Table 2). Except for the Expf and XD6 data sets, all were drawn from the Irvine Repository (Merz & Murphy, 1996). Expf is a synthetic data set with only 2 features. Data points are drawn uniformly from the rectangle $x \in [-10, +10], y \in [-10, +10]$ and labeled according to the decision boundaries shown in Figure 2. The XD6 dataset contains examples generated from the propositional formula $(a_1 \wedge a_2 \wedge a_3) \vee (a_4 \wedge a_5 \wedge a_6) \vee (a_7 \wedge a_8 \wedge a_9)$. A tenth attribute a_{10} takes on random boolean random values. Examples are generated at random and corrupted with 10% class noise.

We ran ADABOOST on each data set to generate $T = 50$ classifiers, and evaluated each pruning technique with the target number of classifiers set to $10, 20, 30, 40,$ and 50 (no pruning). This corresponds to 80%, 60%, 40%, 20%, and 0% pruning. We also ran C4.5 on each data set—in the figures, we plot the resulting performance of C4.5 as 100% pruning. Performance was evaluated either by 10-fold cross-validation or by using a separate test set (as noted in Table 2). Where a separate test set was used,

Table 2: Data sets studied in this paper. "10-xval" indicates that performance was assessed through 10-fold cross-validation.

Name	# Class	Training Set Size	Test Set Size	Eval. Method
Auto	7	184	21	10-xval
Breast	2	629	70	10-xval
Chess	2	836	92	10-xval
Expf	12	1000	1000	test set
Glass	7	192	22	10-xval
Iris	3	135	15	10-xval
Letters	26	12000	4000	test set
Lymph	4	133	15	10-xval
Waveform	3	300	4700	test set
XD6	2	180	20	10-xval

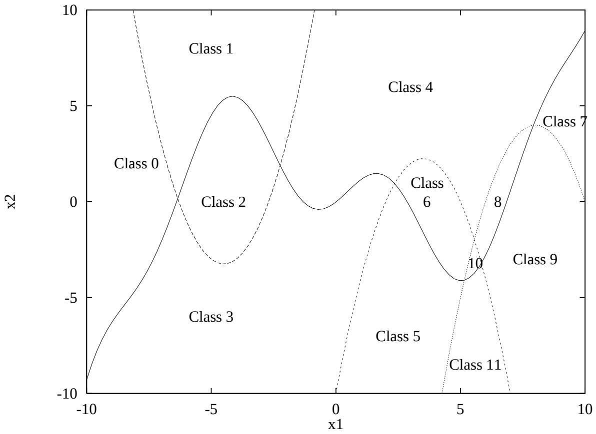

Figure 2: Decision boundaries for the Expf data set.

the experiment was repeated 10 times using 10 random train/test splits and the results were averaged. For Reduce-Error Pruning, we held out 15% of the training set to serve as a pruning set.

To obtain overall performance figures, define the *Gain* to be the difference in percentage points between the performance of full AdaBoosted C4.5 and the performance of C4.5 alone. In all of our 10 domains, this *Gain* was always positive. For any alternative method, we will define the *relative performance* of the method to be the difference between its performance and C4.5 divided by the *Gain*. Hence, a relative performance of 1.0 indicates that the alternative method obtains the same gain as AdaBoost. A relative performance of 0.0 indicates that the alternative method obtains the same performance as C4.5 alone.

Figure 3 shows the mean normalized performance of each pruning method averaged over the ten domains. A performance greater than 1.0 indicates that the pruned AdaBoost actually performed better than AdaBoost.

From the figure, we can see that Reduce-Error Pruning and Kappa pruning perform best at all levels of pruning (at least on average). The Convex Hull method is competitive with these at its fixed level of pruning. The KL-divergence and Early Stopping methods do not perform very well at all. This is true of the analogous plots for each individual domain as well (data not shown). The poor performance of early stopping shows that AdaBoost does not construct classifiers in decreasing order of quality. Pruning is "skipping" some of the classifiers produced by AdaBoost early in the process in favor of classifiers produced later.

Figure 4 shows the normalized performance of Reduce-Error Pruning on each of the ten domains. Here we see that for the Chess, Glass, Expf, and Auto data sets, pruning can improve performance beyond the level achieved by AdaBoost. This suggests that AdaBoost is exhibiting overfitting behavior in these domains. The figure also shows that for Chess, Glass, Expf, Iris, and Waveform, pruning as many as 80% of the classifiers still gives performance comparable to AdaBoost. However, in the Auto, Breast, Letter, Lympho, and XD6 domains, heavy pruning results in substantial decreases in performance. Even 20% pruning in the Breast domain hurts performance badly.

Figure 5 shows the results for Kappa Pruning. Pruning improves performance over AdaBoost for Breast, Chess, Expf, Glass, Iris, Lympho, and Waveform. The only data set that shows very bad behavior is Iris, which appears to be very unstable (as has been noted by other authors). Five domains (Chess, Expf, Glass, Iris, and Waveform) can all be pruned to 60% and still achieve a relative performance of 0.80. Hence, in many cases, significant pruning does not hurt performance very much.

Figure 6 shows the performance of Convex Hull pruning. The performance is better than the other pruning methods (at the corresponding level of pruning) for the Auto, Breast, Glass, Waveform, and XD6 and equal or worse for the other data sets.

5 Conclusions

From these experiments, we conclude that the ensemble produced by AdaBoost can be radically pruned (60–80%) in some domains. The best pruning methods were Kappa Pruning and Reduce-Error Pruning. The good performance of Reduce-Error Pruning is surprising, given that only a small hold-out set (15%) is used, and given that the training set is smaller as well. On the other hand, Reduce-Error Pruning takes the most direct approach to finding a subset of good classifiers. It does not rely on heuristics concerning diversity or accuracy.

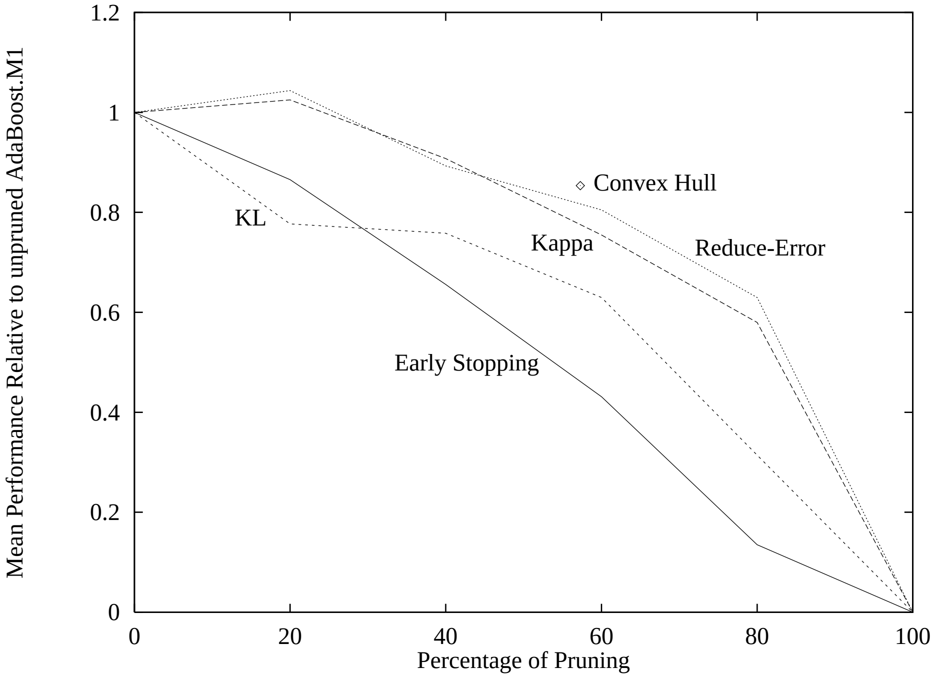

Figure 3: Relative performance of each pruning method averaged across the ten domains as a function of the amount of pruning. Note that the Convex Hull pruning appears as a single point, since the amount of pruning cannot be controlled in that method.

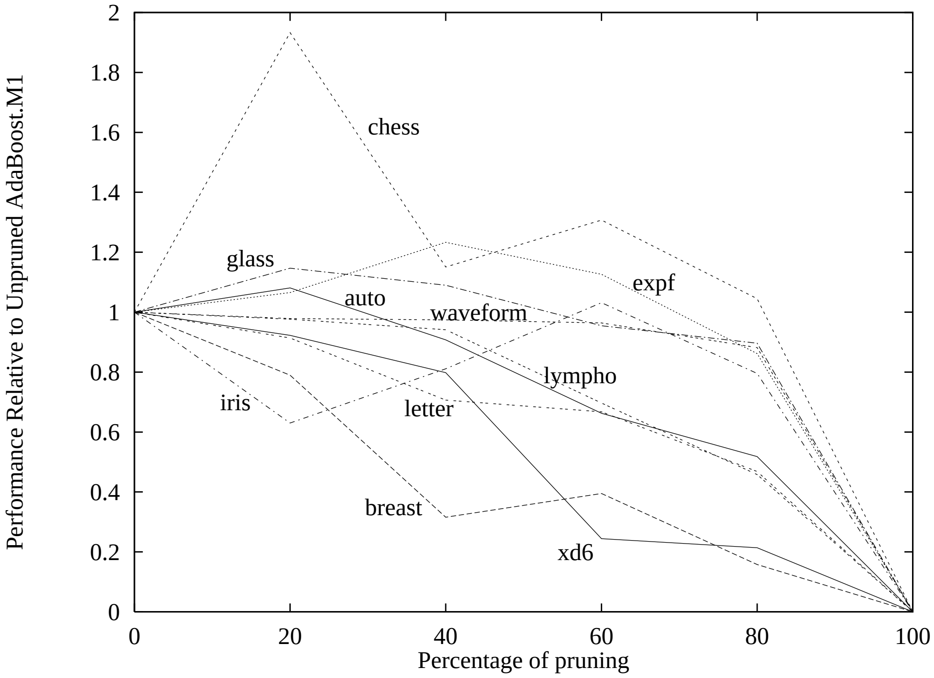

Figure 4: Normalized performance of Reduce-Error Pruning with various amounts of pruning.

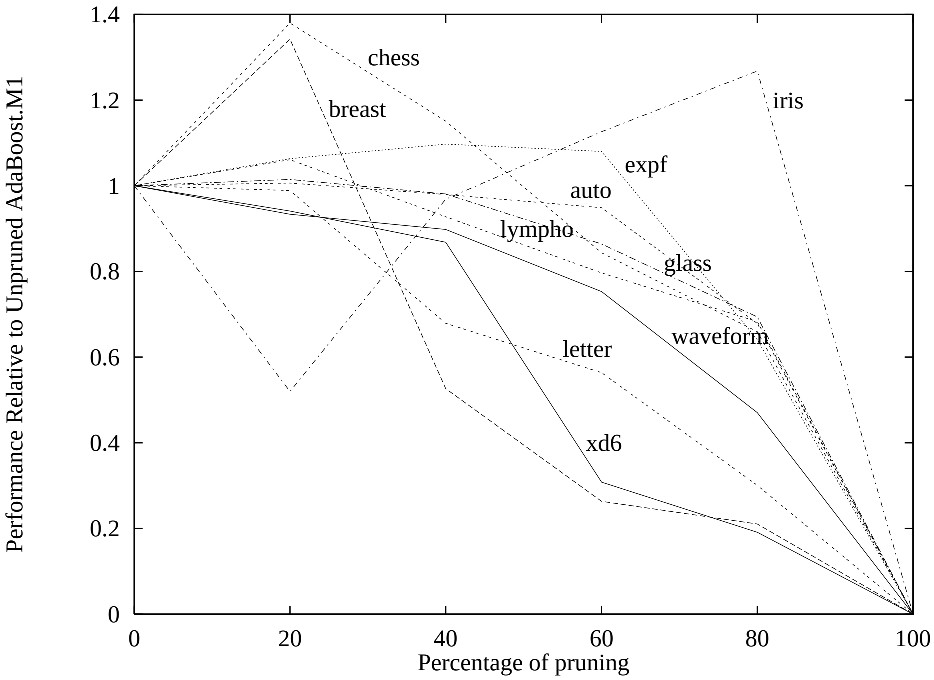

Figure 5: Relative performance of Kappa Pruning with various amounts of pruning.

The good performance of Kappa pruning is very attractive, because it does not require any hold-out set for pruning. It may be possible to improve Kappa pruning further by applying backfitting as we did with Reduce-Error Pruning. The Convex Hull method also gives acceptable performance in several domains, but it is less attractive because it does not permit control over the amount of pruning.

The results show that AdaBoost may be over-fitting; pruning by early stopping performs badly on every data set except Auto. Hence, some form of pruning should always be considered for AdaBoost. This raises the question of how much pruning should be performed in a new application. An obvious strategy is to select the amount of pruning through cross-validation. For most of the domains we have tested, the behavior of pruning is fairly smooth and stable, so cross-validation should work reasonably well. For Iris, however, it was very unstable, and it is doubtful that cross-validation could find the right amount of pruning.

Reduce-Error Pruning may not require cross-validation to determine the amount of pruning. Instead, it may also be possible to use the pruning data set to determine this.

The paper also introduced the Kappa-Error diagram as a way of visualizing the accuracy-diversity trade-off for voting methods. We showed that—as many people have suspected—Bagging produces classifiers with much less diversity than AdaBoost.

Acknowledgements

The authors gratefully acknowledge the support of the National Science Foundation under grant IRI-9626584. The authors also thank the reviewers for identifying a major error in our Kappa experiments which led us to correct and rerun them all.

References

Agresti, A. (1990). *Categorical Data Analysis*. John Wiley and Sons., Inc.

Bakiri, G. (1991). Converting English text to speech: A machine learning approach. Tech. rep. 91-30-2, Department of Computer Science, Oregon State University, Corvallis, OR.

Bishop, Y. M. M., Fienberg, S. E., & Holland, P. W. (1975). *Discrete multivariate analysis: Theory and practice*. MIT Press, Cambridge, Mass.

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Breiman, L. (1996b). Bias, variance, and arcing classifiers. Tech. rep., Department of Statistics, University of California, Berkeley.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Meas.*, *20*, 37–46.

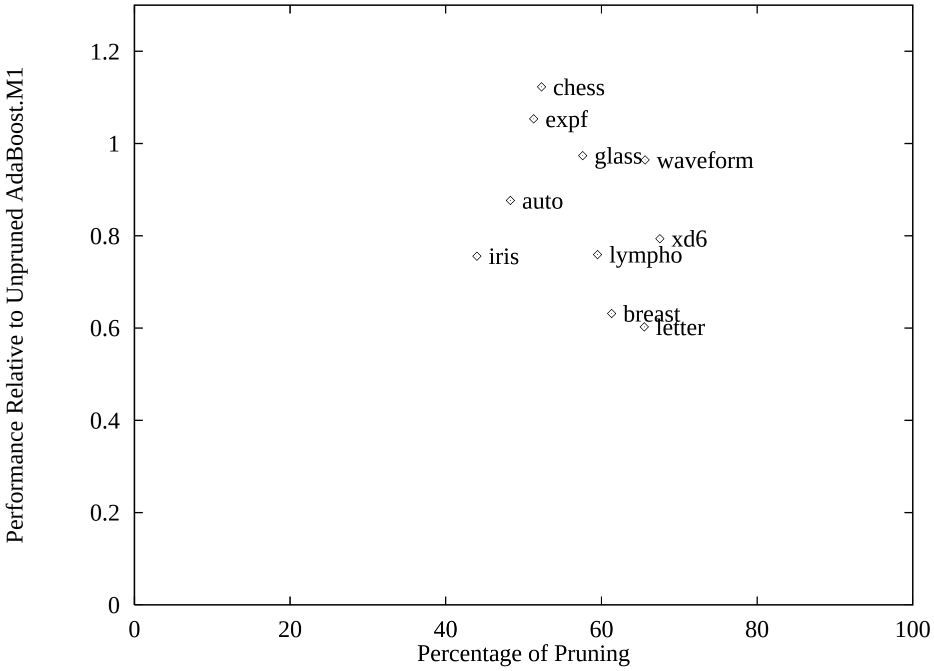

Figure 6: Normalized performance of Convex Hull Pruning.

Cover, T., & Thomas, J. (1991). *Elements of Information Theory.* J.Wiley and Sons, Inc.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the International Conference in Machine Learning*, pp. 148–156 San Francisco, CA. Morgan Kaufmann.

Freund, Y., & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. Tech. rep., AT&T Bell Laboratories, Murray Hill, NJ.

Friedman, J. H., & Stuetzle, W. (1981). Projection pursuit regression. *J. American Statistical Association*, *76*(376), 817–823.

Merz, C. J., & Murphy, P. M. (1996). UCI repository of machine learning databases. Tech. rep., U.C. Irvine, Irvine, CA. [http://www.ics.uci.edu/m̃learn/MLRepository.html].

Quinlan, J. R. (1993). *C4.5: Programs for Empirical Learning.* Morgan Kaufmann, San Francisco, CA.

Quinlan, J. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 725–730 Cambridge, MA. AAAI Press/MIT Press.