, , 1–19 ()

Improving Bagging Performance by Increasing Decision Tree Diversity

BERNHARD PFAHRINGER bernhard@ai.univie.ac.at

Austrian Research Institute for AI, Vienna, Austria

IAN H. WITTEN ihw@cs.waikato.ac.nz

Department of Computer Science, University of Waikato, Hamilton, New Zealand

Editor: Philip Chan

Abstract. Ensembles of decision trees often exhibit greater predictive accuracy than single trees alone. *Bagging* and *boosting* are two standard ways of generating and combining multiple trees. Boosting has been empirically determined to be the more effective of the two, and it has recently been proposed that this may be because it produces more diverse trees than bagging. This paper reports empirical findings that strongly support this hypothesis. We enforce greater decision tree diversity in bagging by a simple modification of the underlying decision tree learner that utilizes randomly-generated decision stumps of predefined depth as the starting point for tree induction. The modified procedure yields very competitive results while still retaining one of the attractive properties of bagging: all iterations are independent. Additionally, we also investigate a possible integration of bagging and boosting. All these ensemble-generating procedures are compared empirically on various domains.

Keywords: Bagging, Boosting, Sampling, Diversity, Decision Trees

1. Introduction

Methods of machine learning like bagging and boosting that operate by creating ensembles of classifiers and combining the result have attracted considerable interest because of their empirical success. These methods are ideal for use in domains where the utmost is sought in predictive performance, with little regard for the intelligibility of the results of induction. Bagging, in particular, enjoys absolute

simplicity and an inherently parallel nature which encourages efficient implementation on multiprocessors—an advantage not shared by boosting. However, boosting seems empirically to give better results, and explanations have been sought for this difference in terms of a decomposition into bias and variance[4, 2]. Furthermore, it has recently been noticed that boosting tends to produce a more diverse set of classifiers than bagging, and this has been cited as a factor in increased performance [11].

The present paper explores the proposition that the higher performance of boosting can be explained by the greater diversity of the classifiers that it produces. We report an empirical comparison of bagging and boosting, together with some variants of bagging which are designed specifically to increase the diversity of the classifiers involved. All experiments use decision tree induction as the underlying weak learning method. We have chosen a reasonable mix of small- and medium-size databases available at the UCI repository [12] for this study.

The next section describes the various ensemble-generating procedures that we explore. Section 3 explains the experimental design and reports on the results obtained. In Section 4, we review the Kappa-Error diagram of [11] as a means of comparison of ensembles, and define a new syntactic similarity measure for decision trees that overcomes some of the deficiencies of the Kappa-Error diagram for this purpose. Finally, Section 5 summarizes and elaborates upon our findings, pointing out interesting directions for further research.

2. Algorithms

We begin by reviewing the standard definitions of both bagging and boosting, and discuss some implementational details of boosting that address problems we encountered when using rather a strong "weak learner." Then we introduce several variants of bagging that are aimed at increasing the diversity of the decision trees produced. It turns out in Section 3 that despite the fact that members of the ensembles are generated independently, these variants often perform very well compared to boosting.

2.1. Bagging

In an important discovery, Breiman [3] noticed that some kind of classifiers, including decision trees, are rather unstable: small changes in the training set can cause classifiers induced from them to be quite different. The so-called *bagging* procedure takes advantage of this effect to induce a set of different classifiers from a given single set of training data. Examples are drawn at random, with replacement, to yield sets of the same size as the original training set, but which contain on average only about 63.2% of the training examples, the remainder being duplicates. This method is reminiscent of the statistical procedure of "bootstrapping" [6].

Input: m examples, desired number of iterations T

1. **initialize** $D(i) = 1/m$;
2. **for** $t = 1$ **to** T **do**
3. $\quad L_t = draw(m, D)$
4. $\quad h_t = induce\_tree(L_t)$
5. **return** $h = \sum_{t=1}^{T} h_t/T$

Output: an ensemble of T equally-weighted decision trees

Figure 1. The bagging algorithm.

Figure 1 gives pseudo-code for our implementation of bagging. The procedure $draw(m, D)$ draws m examples randomly from the data set, with replacement, according to a specified probability distribution D. The procedure *induce_tree* uses a rational reconstruction of the well-known decision tree inducer C4.5 [13] to induce and prune decision trees. Our reconstruction deviates from the original C4.5 implementation in just two points. First, missing values are handled differently: they are treated as a distinct value, instead of using the weighting procedure adopted by C4.5. Second, during pruning the "lifting" operation—replacing a node by its most heavily populated subtree—is not considered. In the experiments reported below, we use C4.5's default settings for all essential parameters.

Note that the probability distribution used to sample the training examples remains the same—we call the sampling process "blind." This is one of the major practical benefits of bagging: all iterations are independent, therefore they can be executed in parallel. This is a valuable asset, particularly for large datasets.

2.2. Boosting

The method of *boosting* [15], unlike bagging, is a purely sequential procedure. The key idea is to force the induction algorithm to concentrate its effort on examples that it misclassified in previous iterations. For this reason boosting can be called "informed" in the sense that sampling takes previous performance into account. The probability distribution used to sample the training examples is updated after each iteration, increasing the weight of misclassified examples and decreasing the weight of correctly classified ones.

Figure 2 gives pseudo-code for our implementation of boosting. The reweighting of the distribution $D(i)$ (in line 9) is governed by the factor α_t, which is determined by the performance of the current classifier with respect to the *current* probability distribution as calculated in line 5. The same weight α_t also determines the contribution of the associated hypothesis h_t to the final ensemble h. Therefore it is not necessarily true (as claimed in [11], for example) that boosting assigns smaller weights to hypotheses with lower accuracy. A good hypothesis for the *current* probability distribution might have a considerable error rate on the *original* training set represented by the initial probability distribution.

Unlike both [4] and [11], our implementation does not resample and restart in cases where ϵ_t exceeds 0.5. In preliminary experiments we found that crossing this threshold is a very good indicator that boosting is failing. Failure may be due to excessive noise in the training data, or to inadequate attributes, or to a bias of the underlying (weak) learner that is strongly maladapted to the learning task. In fact, during the experiments reported below we have never encountered this problem.

Another interesting question is how to handle cases where $\epsilon_t = 0$, which occurs whenever a single tree fits the data perfectly. When the original boosting framework was set up, it was presumably not envisaged that this might actually occur in real

Input: m examples, the number of iterations T

1. **initialize** $D_0(i) = 1/m$;
2. **for** $t = 1$ **to** T **do**
3. $\quad$ /* $L_t = draw(m, D_{t-1})$ */ /*unnecessary for weight-handling learners*/
4. $\quad$ $h_t = induce\_tree(L_t)$
5. $\quad$ $\epsilon_t = P_{i \sim D_{t-1}}[y_i \neq h_t(x_i)]$
6. $\quad$ **if** $\epsilon_t > 0.5$ **then** *abort* **fi**
7. $\quad$ $\alpha_t = ln((1 - \epsilon_t)/\epsilon_t)/2$
8. $\quad$ $Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$
9. $\quad$ $D_t(i) = \frac{D_{t-1} exp(-y_i \alpha_t h_t(x_i))}{Z_t}$
10. **return** $h = \frac{\sum_{t=1}^{T} \alpha_t h_t}{\sum_{t=1}^{T} \alpha_t}$

Output: an ensemble of T weighted decision trees

Figure 2. The ADAboost algorithm.

experiments. According to the standard ADAboost description, such a classifier would be assigned infinite voting power, and boosting would effectively be stopped. We have devised the following simple solution to handle this situation in a way that makes reasonable use of this obviously good tree while retaining the ability to continue boosting. If all training examples were weighted equally (as they are when boosting begins), $alpha_t$ can be rewritten as

$$alpha_t = \frac{log(N_{correctly\ classified}) - log(N_{wrongly\ classified})}{2} \quad (1)$$

By abusing the statistical idea of a Laplace correction, we can imagine that even a "perfect" tree might still make one prediction error, giving

$$alpha_{perfect\ tree} = \frac{log(N_{examples} + 1) - log(1)}{2} \quad (2)$$

This yields a reasonable voting weight for a perfect tree which ignores the current probability distribution.

A second problem is how to reweight the training examples for the next round of boosting. Since all examples are correctly classified, there is no principled way

of increasing the weight of some of the examples and decreasing the weight of the remainder. If the weak learner does not use the weights directly, but instead receives a sample drawn according to the weight distribution, there may not be a problem at all: usually (except for very skewed distributions) the next sample will be sufficiently different to lead to a different classifier. However, a deterministic learner that uses the weights directly would induce exactly the same classifier in every iteration that followed. A reasonable solution is to pertubate the weight distribution with Gaussian noise. This can be done on either the current or the initial weight distribution, and for our experiments we chose the latter.

The original ADAboost formulation supplies the weak learner with a *sample* of equal size drawn from the training set according to the probability distribution generated at each iteration. This ensures that it applies to any kind of weak learner. However, if the weak learner can handle weights directly, there is no need to sample: instead *all* examples can be supplied, with their weights. Like [14, 2], we have chosen this alternative.[1] Therefore we have commented out line 3 of the ADAboost description given in Figure 2.

The superior performance of boosting over bagging has been attributed to a reduction of both variance and (statistical) bias, whereas bagging seems to reduce solely variance. This paper investigates the hypothesis that the better performance of boosting is related to boosting being able to construct a greater variety of different trees.

2.3. S-bagging

Our new version of bagging, which we call "S-bagging" (S for "stump"), increases the diversity of the decision trees involved by randomly generating decision stumps of depth one, two or three as starting points for tree induction. Although attributes are chosen at random in these stumps, the cutpoints for numerical attributes are selected to optimize the information gain. The algorithm is depicted in Figure 3. The procedure $draw\_attrs(f)$ draws f attributes at random without replacement from the set of all available attributes. These attributes define the decision stump, which will be further refined by standard tree induction and pruning.

Input: m examples, the number of iterations T, number of fixed attributes f

1. **initialize** $D(i) = 1/m$;
2. **for** $t = 1$ **to** T **do**
3. $\quad L_t = draw(m, D)$
4. $\quad FA_t = draw\_attrs(f)$
5. $\quad h_t = induce\_tree(L_t, FA_t)$
6. **return** $h = \sum_{t=1}^{T} h_t/T$

Output: an ensemble of T equally weighted decision trees

Figure 3. The S-bagging algorithm, which bags trees created by extending randomly-generated stumps.

The idea of fixing some root structure at random is inspired by a discovery in heuristic search reported in [8]: the likelihood of making a bad heuristic choice is largest close to the root of a search tree. Because we do not search for the single best tree, but instead construct a bagging ensemble, replacing the heuristic choice by an even less informed random choice for the first few tree levels does not have such disastrous consequences. Indeed, it allows a wider variety of trees to be constructed that are still, on the whole, reasonably accurate. We will discuss the question of choosing a good stump-depth later (Section 3.2).

2.4. Bagging Random Trees

An obvious variant of the S-bagging idea is to go to the extreme and simply bag completely random trees. Preliminary experiments show that the increase in diversity comes at too great a cost in terms of predictive accuracy. On average, random tree ensembles perform worse than the single tree induced by C4.5.

However, a simple modification, which we call "R-bagging" (R for "random"), yields interesting results. The procedure is shown in Figure 4, and the crucial twist, which is hidden in procedure *induce_random_tree*, is that it undertakes extensive pruning. Recall that the standard version of C4.5 considers two differ-

Input: m examples, the number of iterations T

1. **initialize** $D_0(i) = 1/m$;
2. **for** $t = 1$ **to** T **do**
3. $\quad L_t = draw(m, D)$
4. $\quad h_t = induce\_random\_tree(L_t)$
5. **return** $h = \sum_{t=1}^{T} h_t/T$

Output: an ensemble of T equally weighted decision trees

Figure 4. The R-bagging algorithm, which bags extensively pruned random trees.

ent pruning operations: (a) turning subtrees into leaves and (b) lifting the most populated subtree up to replace the current node; both operations are actually performed only if the resulting tree yields a better error estimate. Instead of (b), *induce_random_tree* considers lifting *all* subtrees, not just the most populated one, because they are the result of random rather than heuristic choices. This more extensive pruning improves results considerably, but at considerable computational cost, for in comparison to standard C4.5 the complexity of pruning is increased by a factor proportional to the average branching factor of the tree. Moreover, compared to pruning with no lifting of any subtrees there is an additional factor proportional to the average tree depth, because every example has to be reclassified that many times during pruning.

2.5. B-boosting: Bagged boosting

For comparison purposes we have also investigated one way of directly combining bagging and boosting: simply bag a few boosting iterations as illustrated in Figure 5. In a sense, this procedure retains the advantages of both methods: it exhibits independence at the level of the outer loop, and it produces quite diverse trees in the inner loop where the boosting algorithm tries to fit the bagging samples. For the experiments reported below we have fixed both the number of bagging operations T_{bag} and the number of boosting operations T_{boost} to 10, resulting in ensembles

of size 100. Given a fixed number of available iterations, this is a rather *ad hoc* allocation, and other choices would no doubt prove superior. The final hypothesis is simply the normalized sum of all boosting hypotheses.

Input: m examples, the number of bagging iterations T_{bag}, the number of boosting iterations T_{boost}

1. **initialize** $D_0(i) = 1/m$;
2. **for** $t = 1$ **to** T_{bag} **do**
3. $\quad L_t = draw(m, D_0)$
4. $\quad h_t = boost(L_t, T_{boost})$
5. **return** $h = \sum_{t=1}^{T_{bag}} h_t / T_{bag}$

Output: an ensemble of $T_{bag} \cdot T_{boost}$ weighted decision trees.

Figure 5. The bagged-boosting algorithm, which bags a few boosting iterations.

We have not yet tried the inverse of this procedure, namely boosted bagging or B-bagging, which is left for future research.

3. Experiments

This section reports experimental results for the algorithms described above. Our question is: given a pre-specified ensemble size, what algorithm constructs the best ensemble? Based on experimental findings reported in the literature, we have fixed the size at 100. This seems a reasonable choice for decision trees, where further iterations generally yield only marginal gains (unlike other kinds of weak learner, where even thousands of iterations may make a significant difference [15]). Presumably, decision tree inducers are not so "weak" after all.

To estimate predictive accuracy, stratified five-fold cross-validation was repeated five times. The rationale is this. Single cross-validation can be rather unstable [1], therefore one should average at least a few. Results of five-fold cross-validation do not seem to differ greatly from those of the more commonly chosen ten-fold cross-validation. But of course five-fold cross-validation is more efficient, having fewer

iterations and also fewer examples in each—which makes a difference for algorithms whose complexity is non-linear. This is important because our experiments involved inducing a quarter of a million trees. For the two largest domains, however, we did not perform cross-validation but instead used the prespecified split into training and test sets that was supplied with these datasets.

3.1. Algorithms and datasets

The algorithms we used are those discussed above. In summary,

C4.5 Like the standard C4.5 except that missing values are handled differently and subtree lifting is disabled.

BAG Standard bagging (Figure 1) as described in Section 2.1.

BOOST Standard boosting (Figure 2) as described in Section 2.2.

SB1 S-bagging (Figure 3) as described in Section 2.3, with random initial stumps of depth 1.

SB2 S-bagging with random initial stumps of depth 2.

SB3 S-bagging with random initial stumps of depth 3.

RB R-bagging (Figure 4) as described in Section 2.4.

BB B-boosting (Figure 5) as described in Section 2.5.

All datasets used for experiments are from the UC Irvine repository [12]. We chose some small and medium-sized datasets that exhibit a good mix with respect to the following characteristics: number of classes, number of examples, number of attributes, and proportion of categorical and numerical attributes. Table 1 summarizes these characteristics, along with the testing procedure—either five five-fold cross-validations, or the use of a prespecified test set.

Table 1. Characteristics of the used datasets.

Dataset	#Ex	#C	#Cat	#Num	Test
Audiology	226	24	69	0	5×5 cv
Breast	699	2	0	9	5×5 cv
Colic	368	2	14	8	5×5 cv
Credit	690	2	9	6	5×5 cv
Diabetes	768	2	0	8	5×5 cv
DNA	3175	3	60	0	5×5 cv
German	1000	2	13	7	5×5 cv
Glass	214	6	0	9	5×5 cv
Iris	150	3	0	4	5×5 cv
Labor	57	2	8	8	5×5 cv
Lymph	148	4	18	0	5×5 cv
Sonar	208	2	0	60	5×5 cv
Soybean	683	19	35	0	5×5 cv
Vote	435	2	16	0	5×5 cv
Letter	20000	26	0	16	4000
Satimage	6435	6	0	36	2000

3.2. Results

Tables 3 and 4 give the average predictive error rates, with errorbars showing the measured standard deviation, for all domains and methods. Not surprisingly, no clear single winner emerges. But we can observe some interesting global trends.

First, as has been observed by others [2], standard bagging is *uniformly* better than C4.5—its performance is never worse. On the other hand, the improvement is rather modest most of the time. Bagging's impotence is evident from the Table 2, which summarizes the rankings of each method. Standard bagging never wins in any domain. Also, on average it ranks worse than any other combination method. Boosting's advantage over bagging is also evident with boosting being ranked better on average and winning 10 times (out of 16) in a pairwise comparison

to bagging. Furthermore, it is interesting to identify the domains where boosting fails. For instance, overfitting might reasonably explain boosting's performance in the `Audiology` domain, as there is ample opportunity for overfitting given such a small dataset together with a large number of both attributes and classes. Bad boosting performance in the `Colic` domain has been observed before [14] too, but apparently no explanation for this failure is known.

Table 2. Ranking summary: how many times is a method the Nth-best method for some domain.

Method	1st	2nd	3rd	4th	5th	6th	7th	8th	Mean
C4				1		2	2	11	7.4
BAG		1	1	1	3	5	5		5.6
BOOST	2	2		3	2	2	2	3	4.9
SB1	1	2	3	2	5	2	1		4.1
SB2	4	3	5	3		1			2.7
SB3	2	6	3	3	1	1			2.9
BB	1	2	3	2	3	3	2		4.3
RB	6		1	1	2		4	2	4.2

Second, even though R-bagging (RB) wins six (out of sixteen) times, we would not propose it as a useful ensemble constructor because its performance is very unstable. It performs exceptionally badly in several of the remaining domains—sometimes even worse than the baseline represented by a single decision tree.

Third, S-bagging using initial stumps of depth two (SB2) performs quite well. Never worse than the baseline, it wins four times and is close to the best in most other domains. This is reflected in the best average ranking of all methods. Bagged boosting comes quite close, particularly for the larger datasets, outperforming SB2 in three of the four domains with more than 1000 examples. It is not clear whether this is due to dataset size *per se*, or to a combination of size and a predominance of numerical attributes. C4.5 is a rather poor choice for truly numerical input, and may provide inadequate bias as a weak learner—in which case one might expect the limited boosting that is represented in bagged boosting to address this deficiency more effectively than the other methods. It is also interesting to compare bagged

boosting to standard boosting. Bagged boosting seems to have a kind of moderating influence on standard boosting: where plain boosting performs badly, bagging improves it, but it damages performance in domains where boosting excels. Bagged boosting may present a reasonable alternative for noisy domains, where ADAboost sometimes performs badly [2, 14].

Initial stumps of depth one and three perform less well in S-bagging than stumps of depth two, on average. These choices may produce too little and too great diversity, respectively. However, we do not think that this indicates that depth two is necessarily superior in general. On the contrary, it is likely that the optimal choice of depth is determined by the number of examples, together with the number of (numerical) attributes. For instance, in the `Letter` domain, depth three comfortably outperforms depth two, and in further experiments (not detailed here) we found depths five and six to be even better for this domain.

4. Measuring diversity

One possible explanation for the perceived differences in predictive behavior of the various ensemble-generating procedures such as bagging and boosting is that the trees which are induced differ in the amount of diversity they exhibit. To visualize these differences we have used two techniques: the Kappa-Error diagram of [11] and a similar DTSim-Error diagram.

4.1. Kappa

Margineantu and Dietterich [11] introduce a diagram based on the well-known Kappa statistic that quantifies the similarity of two classifiers by comparing their predictions. The Kappa statistic is an extensional measure of comparison and can be applied to any categorical classifier. It is defined as

$$\kappa = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2} \tag{3}$$

Θ_1 measures the agreement between the two classifiers:

$$\Theta_1 = \frac{\sum_{i=1}^{L} C_{ii}}{m} \tag{4}$$

where m is the total number of examples, L the number of possible classes, and C_{ii} the number of examples that are assigned to class i by both classifiers. Θ_2 is a correction that takes into account the likelihood of arbitrary agreements:

$$\Theta_2 = \sum_{i=1}^{L}(\sum_{j=1}^{L} \frac{C_{ij}}{m} \cdot \sum_{j=1}^{L} \frac{C_{ji}}{m}) \tag{5}$$

If two classifiers produce exactly the same predictions, $\kappa = 1$. On the other hand, if their predictions coincide purely at a chance level, then $\kappa = 0$. Negative values are possible, just as negative correlation coefficients may occur in linear regression.

The Kappa-Error diagram plots the average predictive error of a pair of classifiers (vertically) against their κ value (horizontally). Ideally, when combining classifiers in an ensemble, one would like the individual classifiers to be simultaneously maximally distinct and maximally correct [9]: a pair of such classifiers would appear at the lower lefthand corner of the diagram. Obviously this is too much to expect: a trade-off is involved because maximally correct classifiers would simply fit the training data exactly and therefore be identical to each other as far as the Kappa statistic was concerned. This trade-off is reflected by a sloping line in the Kappa-Error diagram below which it is not possible to go: pairs of classifiers on the line cannot be made to perform better (moved downwards) without also becoming less similar to each other (moving to the right), and *vice versa*. Pairs of good classifiers cluster just above this line.

Figure 5 gives Kappa-Error diagrams for the `Breast` domain for the seven ensemble-generating procedures used in the experiments. It is important to note the different scales employed for the y-axis: bagging and all s-bagging variants have a much smaller range of predictive error than r-bagging, boosting, or bagged-boosting. This is necessary for detecting any distinction between the former four methods at all. Clearly there is a considerable difference between bagging (top left) and boosting (top right). However, some of the other diagrams—particularly those for bagging and S-bagging using initial stumps of depth one (center left)—are almost identical. We conclude that the Kappa-Error diagram only partially explains differences in the predictive performance of ensembles.

4.2. DTSim

On comparing the trees in a bagging ensemble with those in an S-bagging ensemble, it is immediately clear that there is a huge difference in the actual tree structures. Of course, this is obvious from the way in which these trees are constructed. But no extensional measure like Kappa can be expected to capture differences that are purely structural and are not reflected in classification performance. In order to portray this finer distinction, we introduce a structural similarity measure, DTSim, that is specific to decision trees.

DTSim is defined as

$$\text{DTSim}(t_1, t_2) = \frac{|a(t_1) \cap a(t_2)|}{min(|a(t_1)|, |a(t_1)|)} \tag{6}$$

Here, $a(t)$ denotes the set of all "abstract paths" in a decision tree t. An abstract path is the multiset of attributes tested along a specific path from the tree's root to a leaf, along with the class assigned at that leaf. We ignore the actual values of attributes and numerical cut-points. Thus DTSim measures the percentage of abstract paths that are shared by two trees. Note that the measure reaches its maximum value of 1 not only when the two trees are identical, but also for different trees that might test quite different numerical cutpoints for the same attributes—so long as they both reach the same conclusions.

As an example, consider the two simple decision trees t_1 and t_2 depicted in Figure 6. They have the following abstract paths and consequent DTSim value.

$$\begin{aligned} a(t_1) &= \{\ \{\text{A2,A9,+}\}, \{\text{A2,A9,–}\}, \{\text{A9,A9,+}\}, \{\text{A9,A9,–}\}\ \} \\ a(t_2) &= \{\ \{\text{A2,A9,+}\}, \{\text{A2,A9,–}\}, \{\text{A2,A5,+}\}, \{\text{A2,A5,–}\}, \{\text{A2,–}\}\ \} \\ \text{DTSim}(t_1, t_2) &= \tfrac{2}{\min(4,5)} = 0.5 \end{aligned}$$

The definition of DTSim may seem *ad hoc*, and more sophisticated similarity measures could easily be defined for decision trees—based, for example, on edit distance. However, even such a simple measure allows one to visualize differences that are not apparent in Kappa-Error diagrams. Moreover, simple measures pay off in efficient implementation, which can be an issue given the quadratic growth of pairwise comparison.

The DTSim-Error diagram is constructed in an analogous manner to the Kappa-Error diagram by plotting average error rates of pairs of trees against their DTSim values. The comments about trading error performance against similarity no longer apply: it is perfectly possible for two trees that are quite different structurally to yield the same performance. Figure 7 shows DTSim-Error diagrams for the `Breast` domain. It is once again important to note the different scales employed for the y-axis. Aside from the absence of a tradeoff line, the most prominent difference from the Kappa-Error diagrams in Figure 5 is the clearly-visible distinction between bagging and S-bagging using initial stumps of depth two (top left and center right, respectively). The latter diagram exhibits a larger agglomeration of tree pairs in the lower left corner. These are rather dissimilar trees (their DTSim values lie between 0 and 0.2) but they nevertheless all enjoy a low average error rate.

The reason for the improved discrimination of the DTSim measure over Kappa is the fact that many trees make (almost) identical predictions on the training set, but are nevertheless rather dissimilar in their structure. In this case the Kappa values would be close to 1, whereas the DTSim values would approach 0. Structurally different but high-performance trees might exploit redundancies in the attributes supplied, causing their ensemble behaviour to generate more robust predictions.

5. Conclusions and Discussion

We have devised various modifications of bagging that are aimed at increasing the variety of the induced classifiers, and presented an empirical comparison of their performance. Attention has been confined to a single type of weak learner, namely a decision tree inducer. We have also defined a syntactic similarity measure decision trees, DTSim, which proves useful for visualizing how tree diversity interacts with predictive accuracy for the various ensemble-generating procedures. We have shown that there is indeed a link between the degree of syntactic diversity of bagged classifiers and the prediction quality of the resulting ensemble. Some of the bagging variants we have devised improve considerably over standard bagging: they approach and sometimes even exceed the performance of boosting.

But many open questions remain. Lower-level issues include the optimal choice of parameters of each bagging variant. For instance, the best size of the initial stump in S-bagging is probably a function of both the number of attributes and the number of examples. Finding a good way to allocate iterations in bagged boosting between the outer and inner loops is another interesting practical problem.

The evident success of R-bagging in some domains, together with poor performance in others, suggests that a closer look might be rewarding. R-bagging's successes could be explained by a Bayesian interpretation, summing performance over all possible trees—this would suggest that the incorporation of sophisticated priors might improve upon our simple-minded policy of equal weighting. Generally, all the procedures compared in this paper use either an equal-weight policy or some local form of weight computation. More advanced schemes that take into account all classifiers when deriving weights include "stacked generalization" [17, 16] and "vector support networks" [5]. Such schemes might be able to enhance bagging's performance even more, further narrowing the gap between bagging and boosting.

Other higher-level issues involve the recently-proposed idea of pruning ensembles [11]. It can be quite demanding—particularly of memory—to retain a great number of large classifiers, and it might be possible to decrease this burden by pruning ensembles without prejudicing performance, or perhaps to actually improve performance as one does when pruning decision trees. It would be interesting to compare the use of Kappa and the Kappa-Error diagram for pruning with an analogous use of DTSim and the DTSim-Error diagram.

Pruning ensembles can be viewed as an *a posteriori* way of re-simplifying the structures that are learned. But a more direct approach might be feasible for some combinations of domains and learners. As discussed in [4], ensembles seem to address the problems of bias and variance simultaneously—albeit to a different degree depending on the specific ensemble generator. Now if a learner is adjustable to a rich bias space, one could envision ensemble-guided bias adjustment leading to single classifiers that perform well. Indeed, for good classifiers small ensemble sizes might suffice to reduce the potential residual variance. Is it too much to hope that this will give back what has been sacrificed in the quest for improved predictions, namely intelligibility of the induced structure?

Acknowledgments

Most of the ideas and experiments reported in this paper were conceived while the first author was visiting the Computer Science Department of the University of Waikato courtesy to the following grant: "Schrödingerstipendium Nr.J01269-MAT" of the Austrian "Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)". Eibe Frank and Yong Wang provided valuable comments and general help for preparing this document. Kai Ming Ting was very influential in numerous discussions on ensemble-generating learning procedures. The Austrian Research Institute for Artificial Intelligence is sponsored by the Austrian Federal Ministry of Science and Transport. The second author acknowledges generous support from the New Zealand Marsden Fund.

Notes

1. We have also experimented with sampling, but our experimental results failed to show a clear difference between bagging and boosting. This accords with results reported in [7], who also employed sampling. However, even using sampling we observed the same qualitative improvement with more diverse trees that is reported in later sections.

References

1. Bailey T.L., Elkan C.: Estimating the Accuracy of Learned Concepts, in Bajcsy R.(ed.), Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, pp.895-901, 1993.
2. Bauer E., Kohavi R.: An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants, submitted to the Special Issue on Integrating Multiple Learned Models, Machine Learning Journal, 1997.
3. Breiman L.: Bagging Predictors, Machine Learning, 24(2), 1996.
4. Breiman L.: Bias, Variance and Arcing Classifiers, University of California, Statistics Department, Berkeley, CA, Technical Report 460, 1996.
5. Cortes C., Vapnik V.: Support-Vector Networks, Machine Learning, 20(3), 1995.

6. Efron B., Tibshirani R.: An Introduction to the Bootstrap, Chapman & Hall, New York, 1994.
7. Freund Y., Schapire R.E.: Experiments with a New Boosting Algorithm, in Saitta L.(ed.), Proceedings of the Thirteenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, pp.148-156, 1996.
8. Harvey W.D., Ginsberg M.L.: Limited Discrepancy Search, in Mellish C.S.(ed.), Proceedings of the 14th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, pp.607-613, 1995.
9. Krogh A., Vedelsby J.: Neural Network Ensembles, Cross Validation, and Active Learning, in Tesauro G., et al., Advances in Neural Information Processing Systems 7, MIT Press, Cambridge, MA, 1995.
10. Maclin R., Opitz D.: An Empirical Evaluation of Bagging and Boosting, in Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI Press/MIT Press, Menlo Park, CA, pp.546-551, 1997.
11. Margineantu D.D., Dietterich T.G.: Pruning Adaptive Boosting, in Proceedings of the Fourteenth International Conference on Machine Learning, Morgan Kaufmann, 1997.
12. Merz, C.J., Murphy, P.M.: UCI Repository of machine learning databases, University of California, Department of Information and Computer Science, Irvine, CA, 1996.
13. Quinlan J.R.: C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.
14. Quinlan J.R.: Bagging, Boosting, and C4.5, Proceedings of the AAAI'96 Conference, 1996.
15. Schapire R.E., Freund Y., Bartlett P., Lee W.S.: Boosting the Margin: A new Explanation for the Effectiveness of Voting Methods, in Proceedings of the Fourteenth International Conference on Machine Learning, Morgan Kaufmann, 1997.
16. Ting K.M., Witten I.H.: Stacked Generalization: When does it Work?, in Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Francisco, CA, pp.866-875, 1997.
17. Wolpert D.H.: Stacked Generalization, Neural Networks, 5(2), 241-260, 1992.

Table 3. Predictive error rates — I

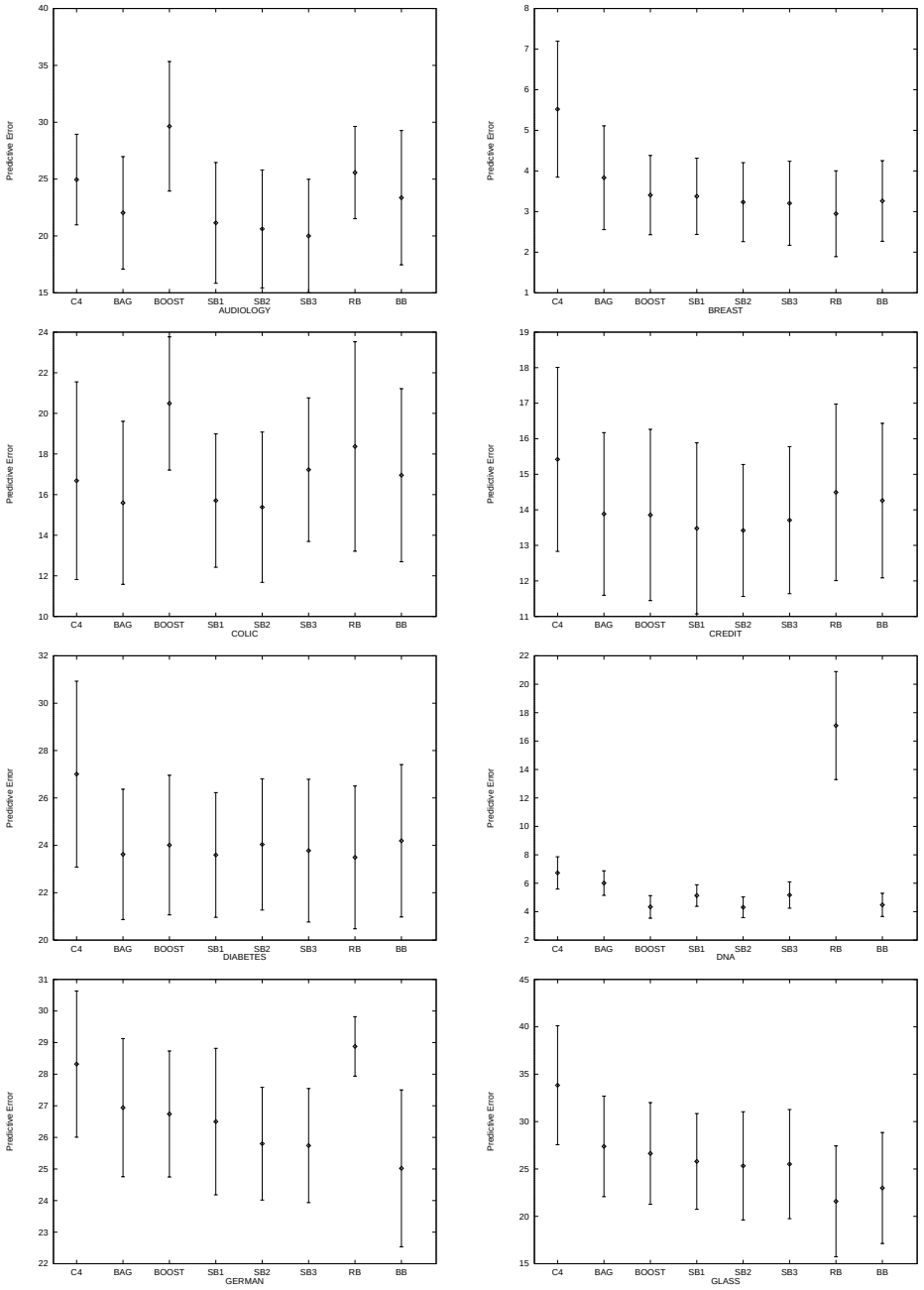

Table 4. Predictive error rates — II

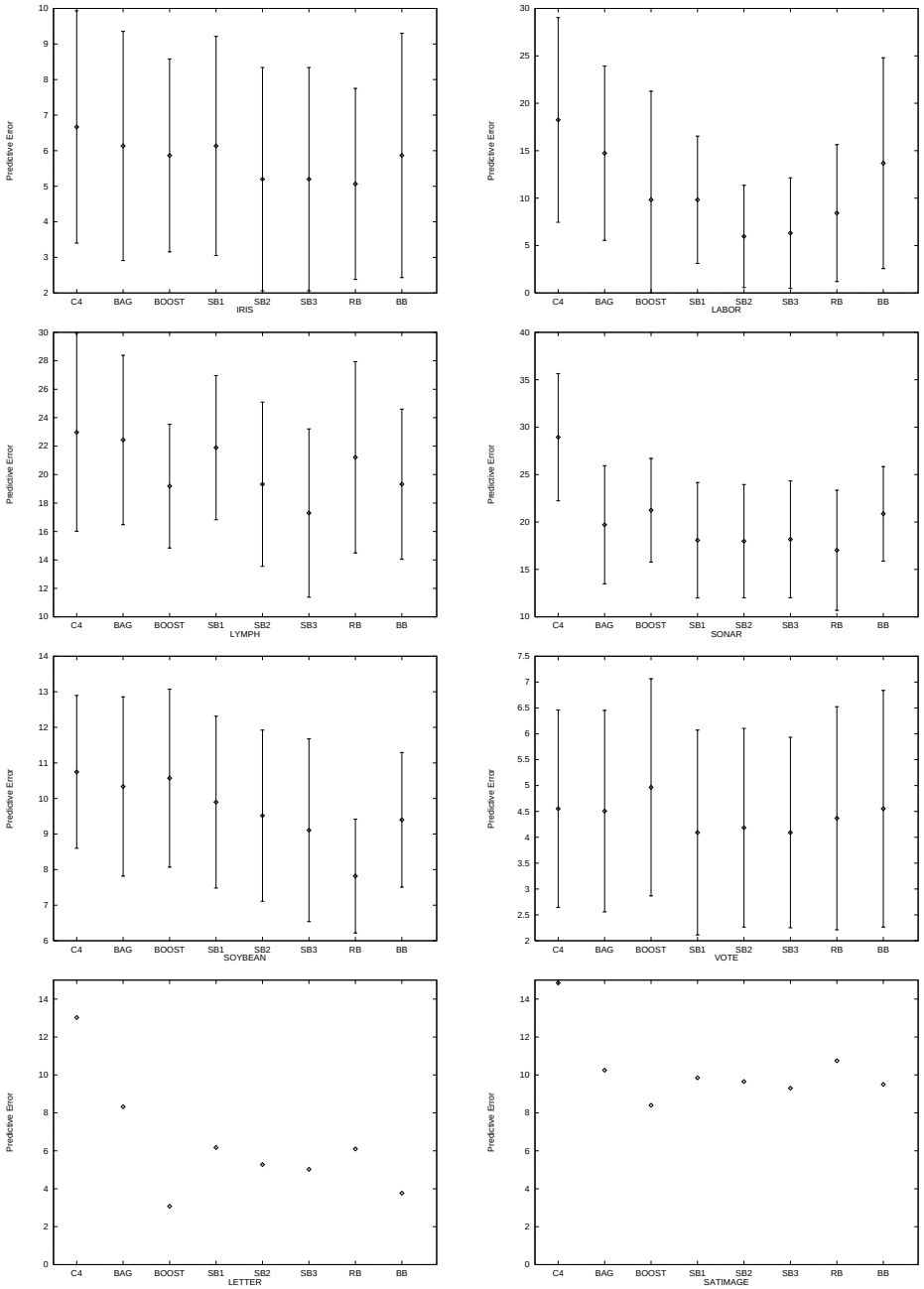

Table 5. Kappa *vs* Predictive error rate, `Breast` domain

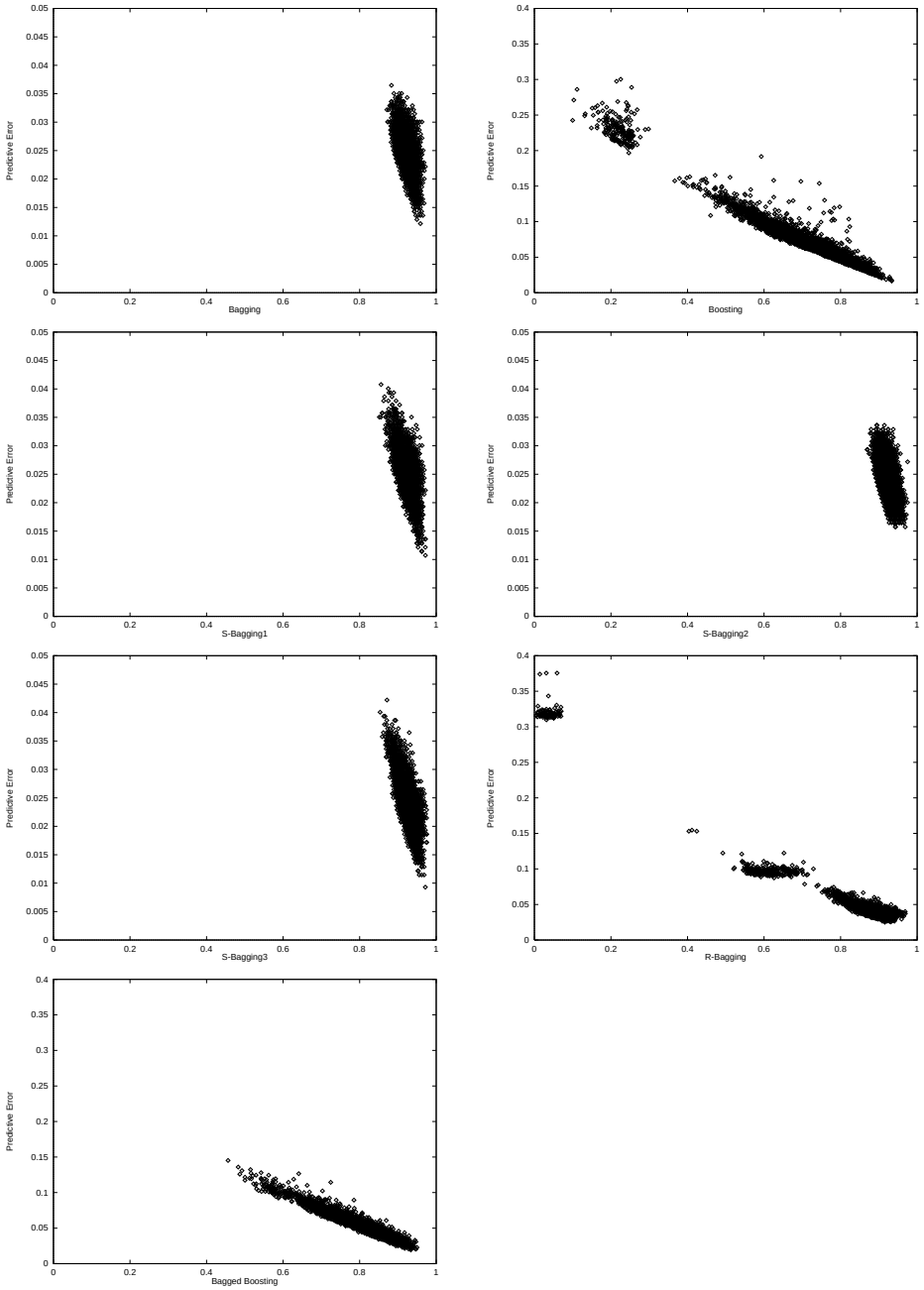

Table 6. Two decision trees t_1 and t_2.

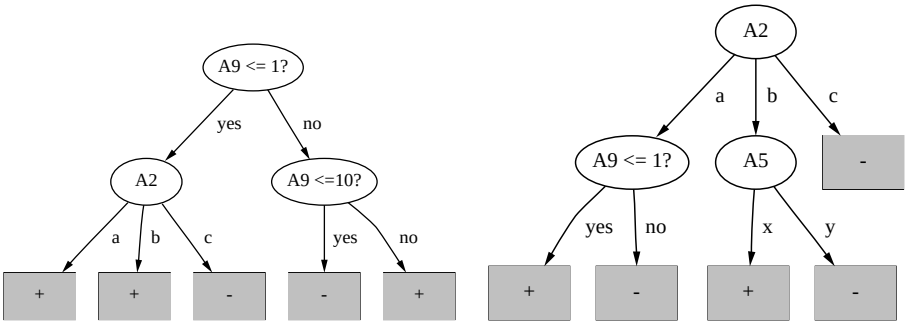

Table 7. DTSim *vs* Predictive error rate, `Breast` domain