Machine Learning, , 1–22 (1999)

An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization

THOMAS G. DIETTERICH tgd@cs.orst.edu
Department of Computer Science, Oregon State University, Corvallis, OR 97331

Editor: Doug Fisher

Abstract. Bagging and boosting are methods that generate a diverse ensemble of classifiers by manipulating the training data given to a "base" learning algorithm. Breiman has pointed out that they rely for their effectiveness on the instability of the base learning algorithm. An alternative approach to generating an ensemble is to randomize the internal decisions made by the base algorithm. This general approach has been studied previously by Ali and Pazzani and by Dietterich and Kong. This paper compares the effectiveness of randomization, bagging, and boosting for improving the performance of the decision-tree algorithm C4.5. The experiments show that in situations with little or no classification noise, randomization is competitive with (and perhaps slightly superior to) bagging but not as accurate as boosting. In situations with substantial classification noise, bagging is much better than boosting, and sometimes better than randomization.

Keywords: Decision trees, ensemble learning, bagging, boosting, C4.5, Monte Carlo methods

1. Introduction

The goal of ensemble learning methods is to construct a collection (an ensemble) of individual classifiers that are diverse and yet accurate. If this can be achieved, then highly accurate classification decisions can be obtained by voting the decisions of the individual classifiers in the ensemble. Many authors have demonstrated significant performance improvements through ensemble methods (Breiman, 1996b; Kohavi & Kunz, 1997; Bauer & Kohavi, 1999; Maclin & Opitz, 1997).

Two of the most popular techniques for constructing ensembles are bootstrap aggregation ("bagging"; Breiman, 1996a) and the Adaboost family of algorithms ("boosting"; Freund & Schapire, 1996). Both of these methods operate by taking a base learning algorithm and invoking it many times with different training sets. In bagging, each training set is constructed by forming a bootstrap replicate of the original training set. In other words, given a training set S of m examples, a new training set S' is constructed by drawing m examples uniformly (with replacement) from S.

The Adaboost algorithm maintains a set of weights over the original training set S and adjusts these weights after each classifier is learned by the base learning algorithm. The adjustments increase the weight of examples that are misclassified by the base learning algorithm and decrease the weight of examples that are correctly classified. There are two ways that Adaboost can use these weights to construct a new training set S' to give to the base learning algorithm. In *boosting by sampling*, examples are drawn with replacement from S with probability proportional to their weights. The second method, *boosting by weighting*, can be used with base learning algorithms that can accept a weighted training set directly. With such algorithms, the entire training set S (with associated weights) is given to the base learning algorithm. Both methods have been shown to be very effective (Quinlan, 1996).

Bagging generates diverse classifiers only if the base learning algorithm is *unstable*—that is, if small changes to the training set cause large changes in the learned classifier. Breiman (1994) explores the causes of instability in learning algorithms and discusses ways of reducing or eliminating it. Bagging (and to a lesser extent, boosting) can be viewed as ways of exploiting this instability to improve classification accuracy. Adaboost requires less instability than bagging, because Adaboost can make much larger changes in the training set (e.g., by placing large weights on only a few of the examples).

This paper explores an alternative method for constructing good ensembles that does not rely on instability. The idea is very simple: randomize the internal decisions of the learning algorithm. Specifically, we implemented a modified version of the C4.5 (Release 1) learning algorithm in which the decision about which split to introduce at each internal node of the tree is randomized. Our implementation computes the 20 best splits (among those with non-negative information gain ratio) and then chooses uniformly randomly among them. For continuous attributes, each possible threshold is considered to be a distinct split, so the 20 best splits may all involve splitting on the same attribute.

This is a very crude randomization technique. One can imagine more sophisticated methods that preferred to select splits with higher information gain. But our goal in this paper is to explore how well this simple method works. In a previous paper (Dietterich & Kong, 1995), we reported promising results for this technique on five tasks. In this paper, we have performed a much more thorough experiment using 33 learning tasks. We compare randomized C4.5 to C4.5 alone, C4.5 with bagging, and C4.5 with Adaboost.M1 (boosting by weighting). We also explore the effect of random classification noise on the performance of these four techniques.

2. Methods

We started with C4.5 Release 1 and modified it to support randomization, bagging, and boosting by weighting. To implement boosting by weighting, we imported the bug fixes from C4.5 Release 8 that concern the proper handling of continuous splits with weighted training examples.

We employed 33 domains drawn from the UCI Repository (Merz & Murphy, 1996). For all but three of the domains (shuttle, satimage, and phoneme), we performed a stratified 10-fold cross-validation to evaluate each of the three ensemble methods (as well as running C4.5 by itself). The remaining three domains have large designated test sets, so we employed standard train/test methodology. The domains were selected without regard to the results on the current study, and no other domains have been tested as part of the study.[1]

For randomization and bagging, we constructed ensembles containing 200 classifiers. For boosting, we constructed ensembles of at most 100 classifiers. However, if the Adaboost.M1 algorithm terminated early (because a classifier had weighted error greater than 0.5 or unweighted error equal to zero), then a smaller ensemble was necessarily used. In all cases, we evaluated ensembles based on both the pruned and the unpruned decision trees. For pruning, we used a confidence level of 0.10. To check whether our ensembles were sufficiently large, we evaluated the performance of different ensemble sizes to determine which ensemble size first matched (or exceeded) the performance of the final ensemble. For randomized C4.5 and bagged C4.5, the required ensemble sizes are similar: nearly all runs had converged (i.e., reached the same accuracy as an ensemble of size 200) within 50 iterations. The king-rook-vs-king (krk) domain required the largest number of iterations, and some folds had not converged after 200 iterations. The letter-recognition task also required a large number of iterations (ranging between 25 and 137 for randomized C4 and from 32 to 200 for bagging). For Adaboost, 40 iterations was sufficient for most domains, but there were a few cases where more than 100 iterations would probably yield further improvements. These include some folds of king-rook-vs-king (krk), letter-recognition, splice, phoneme, segmentation, and waveform. Pruned trees generally required somewhat smaller ensembles than unpruned trees, but the effect is minor.

For each domain and each algorithm configuration (C4.5 alone, randomized C4.5, bagged C4.5, and boosted C4.5), we used the test data to determine whether pruning was needed. Previous research has shown that pruning can make a substantial difference in algorithm performance (Quinlan, 1993), and we did not want to have the pruning decision confound our algorithm comparison. In real applications, the choice of whether to prune could be made based on internal cross-validation within the training set. By using the test data to make this decision, we are making the optimistic assumption that all such cross-validated decisions would be made correctly.

Table 1 summarizes the 33 domains, the results of the experiments, and whether pruning was employed. We did not find any particular pattern to whether pruning was employed except to note that for Adaboost, pruning made no significant difference in any of the 33 domains. For C4.5 and randomized C4.5, pruning made a difference in 10 domains, while for bagged C4.5, pruning made a significant difference in only 4 domains. The general lack of significant differences is probably a result of the relatively low pruning confidence level (0.10) that we employed.

We performed statistical tests to compare the four algorithm configurations. For the 30 domains where cross-validation was performed, we applied the 10-fold cross-validated t test to construct a 95% confidence interval for the difference in the error rates of the algorithms. If this confidence interval does not include zero, then the test concludes that there is a significant difference in performance between the algorithms. However, when applied to the results of cross-validation, this test is known (Dietterich, 1998) to have elevated type I error (i.e., it will incorrectly find a significant difference more often than the 5% of the time indicated by the confidence level). Hence, if the test is unable to conclude that there is a difference between the

Table 1. The 33 domains employed in this study. In the column labeled "P", an asterisk indicates that pruned trees were employed. The error rate column gives the error rate ± a 95% confidence limit. Error rates estimated by 10-fold cross-validation except for phoneme, satimage, and shuttle

		C4.5		Randomized C4.5		Bagged C4.5		Adaboosted C4.5	
index	name	P	error rate	P	error rate	P	error rate	P	error rate
1	sonar		0.3257±0.0637		0.2018±0.0545	*	0.2752±0.0607	*	0.1651±0.0505
2	letter		0.1225±0.0045		0.0285±0.0023		0.0552±0.0032	*	0.0271±0.0023
3	splice	*	0.0575±0.0081	*	0.0397±0.0068	*	0.0506±0.0076		0.0503±0.0076
4	segment		0.0328±0.0073		0.0203±0.0058		0.0263±0.0065		0.0151±0.0050
5	glass	*	0.3437±0.0636		0.2277±0.0562		0.2723±0.0596	*	0.2277±0.0562
6	soybean		0.1262±0.0371	*	0.0852±0.0312	*	0.1009±0.0337	*	0.0757±0.0296
7	autos		0.2326±0.0578	*	0.1581±0.0499		0.1814±0.0528		0.1814±0.0528
8	satimage	*	0.1515±0.0157		0.0890±0.0125		0.1020±0.0133		0.0850±0.0122
9	annealing	*	0.0132±0.0075		0.0088±0.0061		0.0099±0.0065		0.0055±0.0048
10	krk		0.1887±0.0046		0.1309±0.0039		0.1463±0.0041	*	0.1026±0.0036
11	heart-v	*	0.2762±0.0620	*	0.2429±0.0594		0.2619±0.0609	*	0.2810±0.0623
12	heart-c	*	0.2396±0.0481	*	0.1853±0.0437	*	0.1981±0.0449		0.2045±0.0454
13	breast-y	*	0.2601±0.0508	*	0.2500±0.0502	*	0.2635±0.0511	*	0.3142±0.0538
14	phoneme	*	0.1661±0.0086		0.1437±0.0081		0.1509±0.0082	*	0.1464±0.0081
15	voting	*	0.1146±0.0299	*	0.0921±0.0272	*	0.0966±0.0278	*	0.1034±0.0286
16	vehicle		0.2944±0.0307		0.2477±0.0291		0.2570±0.0294		0.2196±0.0279
17	lymph		0.1962±0.0640		0.1772±0.0615		0.1835±0.0624	*	0.1266±0.0536
18	breast-w	*	0.0494±0.0161	*	0.0353±0.0137		0.0367±0.0139		0.0310±0.0128
19	credit-g	*	0.2921±0.0282		0.2416±0.0265	*	0.2495±0.0268		0.2347±0.0263
20	primary	*	0.5845±0.0525	*	0.5501±0.0530		0.5645±0.0528	*	0.5960±0.0522
21	shuttle		0.0003±0.0003		0.0002±0.0002		0.0002±0.0002		0.0001±0.0002
22	heart-s	*	0.0677±0.0444	*	0.0677±0.0444	*	0.0677±0.0444	*	0.0902±0.0506
23	iris		0.0563±0.0369	*	0.0500±0.0349	*	0.0500±0.0349	*	0.0688±0.0405
24	sick	*	0.0132±0.0036		0.0137±0.0037		0.0137±0.0037	*	0.0095±0.0031
25	hepatitis		0.1758±0.0599		0.1636±0.0582		0.1636±0.0582	*	0.1636±0.0582
26	credit-a	*	0.1614±0.0275	*	0.1400±0.0259		0.1371±0.0257	*	0.1300±0.0251
27	waveform	*	0.2341±0.0117		0.1784±0.0106		0.1675±0.0104		0.1521±0.0100
28	horse-colic	*	0.1561±0.0371		0.1561±0.0371		0.1481±0.0363	*	0.1825±0.0395
29	heart-h	*	0.1645±0.0424	*	0.1809±0.0440	*	0.1579±0.0417		0.2039±0.0461
30	labor		0.1493±0.0925	*	0.1493±0.0925		0.1194±0.0842	*	0.1194±0.0842
31	krkp		0.0075±0.0030		0.0075±0.0030		0.0056±0.0026	*	0.0037±0.0021
32	audiology		0.2203±0.0540	*	0.2458±0.0561		0.1822±0.0503	*	0.1525±0.0469
33	hypo		0.0058±0.0024	*	0.0079±0.0028		0.0042±0.0021	*	0.0040±0.0020

Table 2. All pairwise combinations of the four ensemble methods. Each cell contains the number of wins, losses, and ties between the algorithm in that row and the algorithm in that column.

	C4.5	Adaboost C4.5	Bagged C4.5
Random C4.5	14 – 0 – 19	1 – 7 – 25	6 – 3 – 24
Bagged C4.5	11 – 0 – 22	1 – 8 – 24	
Adaboost C4.5	17 – 0 – 16		

two algorithms (i.e., the interval includes zero), this conclusion can be trusted, but when it finds a difference, this conclusion should be regarded with some suspicion.

For the three domains where a single test set was employed, we constructed a confidence interval based on the normal approximation to the binomial distribution (with a correction for the pairing between the two algorithms). This test is safe, but somewhat conservative.

3. Results

Table 2 summarizes the results of these statistical tests. All three ensemble methods do well against C4.5 alone—Randomized C4.5 is better in 14 domains, Bagged C4.5 is better in 11, and Adaboosted C4.5 is better in 17. C4.5 is never able to do better than any of the ensemble methods.

Figure 1 summarizes the observed differences between randomized C4.5 and bagged C4.5. Figure 2 does the same for randomized C4.5 versus boosted C4.5. These plots are sometimes called "Kohavi plots", because they were introduced by Ronny Kohavi in the MLC++ system (Kohavi, Sommerfield, & Dougherty, 1997). Each point plots the difference in the performance of the two algorithms scaled according to the performance of C4.5 alone. For example, in the sonar task, C4.5 (unpruned) gives an error rate of 0.3257; Randomized C4.5 has an error rate of 0.2018, and Bagged C4.5 has an error rate of 0.2752. This means that Randomized C4.5 would give a 38% reduction in error rate over C4.5, while Bagged C4.5 would give only a 15% reduction. The difference in percentage reduction in error rate is 23%, which is what is plotted in the figure (as −0.23). The upper and lower bounds on the confidence interval have been similarly scaled. Hence, the vertical axis indicates the importance of the observed difference (in terms of the improvement over C4.5) while the error bars indicate the statistical significance of the observed difference. In each plot, the 33 domains are sorted in ascending order of their differences. Numerical indexes were assigned to the domains based on the ordering in Figure 1.

Let us consider Figure 1 first. The left end of the figure shows five domains (with indexes 2, 3, 4, 8, and 10 corresponding to letter-recognition, splice, segmentation, satimage, and king-rook-vs-king (krk)) where Randomized C4.5 is clearly superior to Bagged C4.5. Conversely, there are three domains (with indexes 27, 32, and 33 corresponding to waveform, audiology and hypo) where Bagged C4.5 is superior to Randomized C4.5. Ali and Pazzani (1995, 1996) noticed that the domains where bagging does poorly tend to be domains with a large number of training examples.

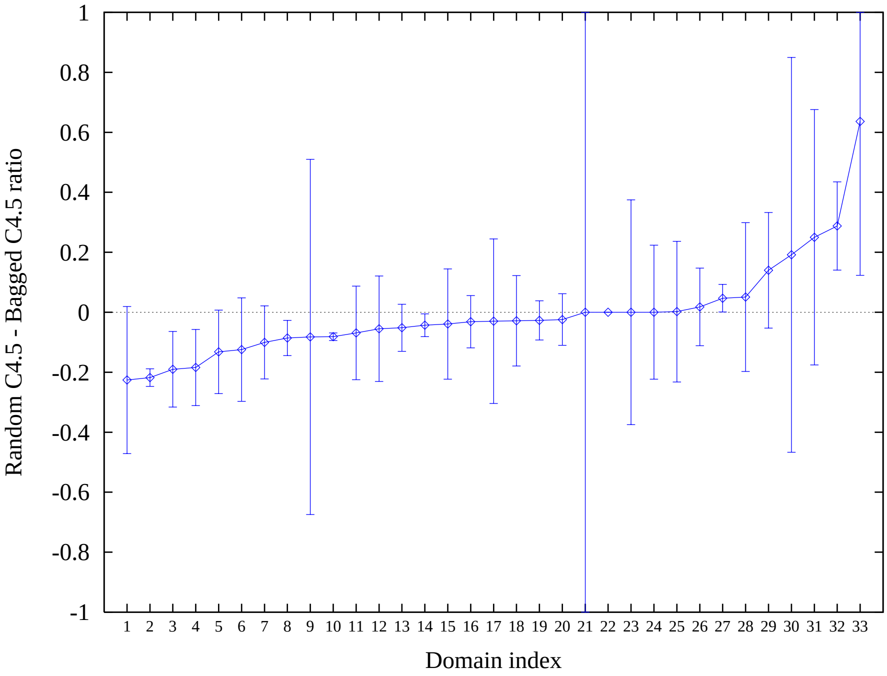

Figure 1. Difference in performance of Randomized C4.5 and Bagged C4.5. The difference is scaled by the error rate of C4.5 alone. Error bars give a 95% confidence interval according to the cross-validated t test (which tends to give intervals that are too narrow). The domains are numbered to correspond with the entries in Table 1

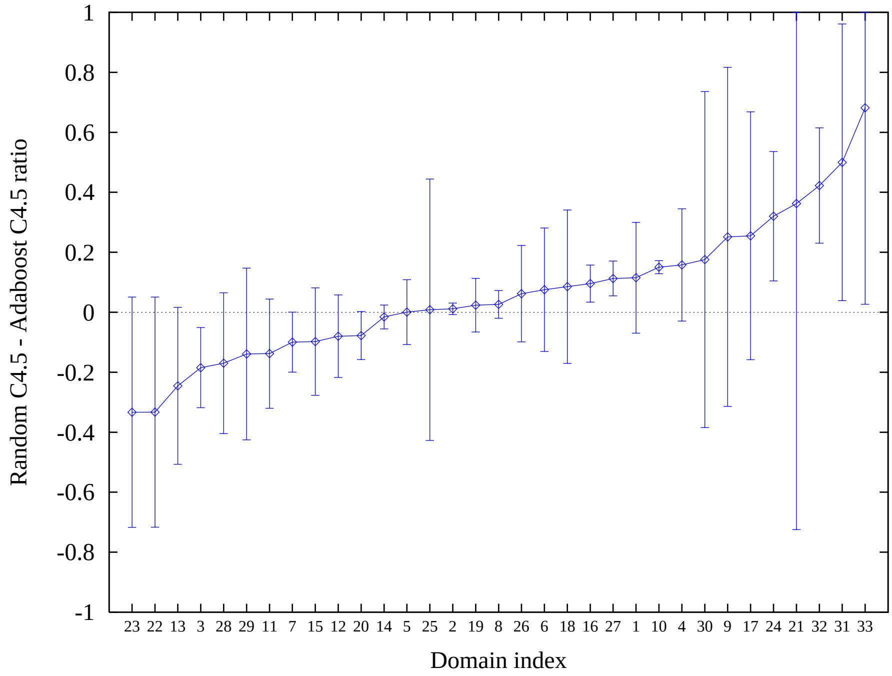

Figure 2. Difference in performance of Randomized C4.5 and Adaboosted C4.5. The difference is scaled by the error rate of C4.5 alone. Error bars give a 95% confidence interval according to the cross-validated t test (which tends to give intervals that are too narrow). The domains are numbered to correspond with the entries in Table 1

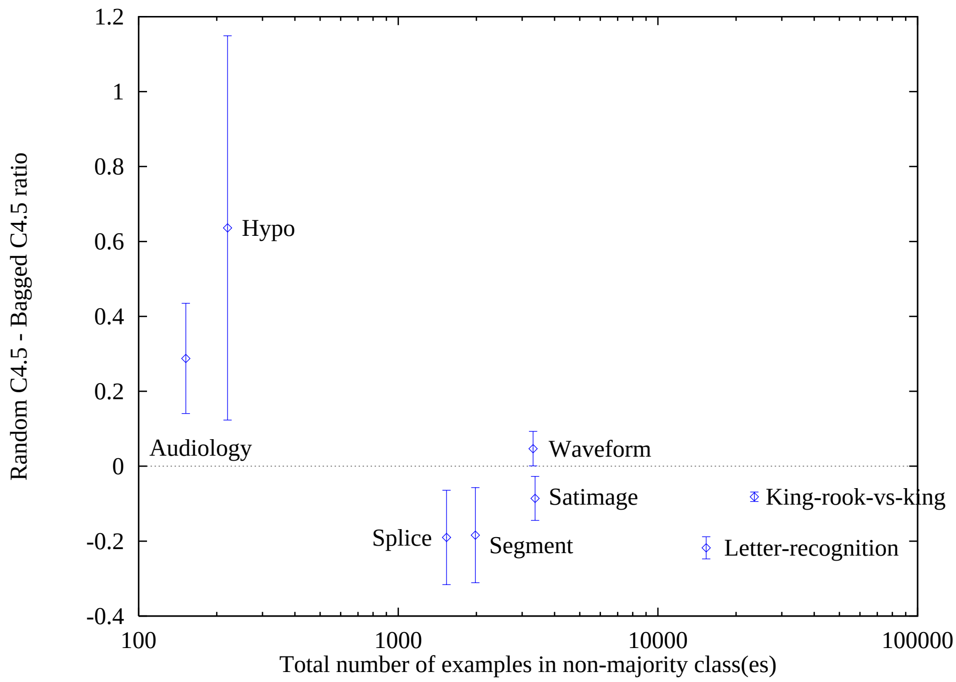

Figure 3. Difference in performance of Randomized C4.5 and Bagged C4.5 as a function of the number of training examples in the non-majority class. The difference is scaled by the error rate of C4.5 alone. Error bars give a 95% confidence interval according to the cross-validated t test (which tends to give intervals that are too narrow).

We can understand this by imagining that C4.5 (without Bagging or Randomization) will produce a particular decision tree T_1 with n_1 leaves. If the bootstrap sample contains enough examples corresponding to each of these n_1 leaves, then Bagged C4.5 will tend to grow the same decision tree T_1. In the limit of infinite sample size, C4.5 will always grow the same tree, and bagging will have no effect on the error rate of C4.5. We can conclude that the effectiveness of bagging will be reduced as the training set becomes very large (unless the corresponding decision trees also become very large).

The effectiveness of randomization, on the other hand, does not depend as much on the size of the training set. Even with an infinitely large training set, Randomized C4.5 would still produce a diverse set of decision trees. (Of course, such an ensemble would probably not be as accurate as a single tree grown without randomness!)

To explore this point, Figure 3 plots the difference in accuracy of Randomized C4.5 and Bagged C4.5 (as in Figure 1) as a function of the total number of training examples in the non-majority classes in the problem. We can see that the five domains where randomization outperforms bagging are five domains with many non-majority-class examples. The domains where Bagging outperforms Randomization are cases where either the confidence interval just barely avoids zero (waveform) or where the training sets are very small. In both of these cases, we must be very suspicious of the cross-validated t test—these are precisely the situations where this test tends to give incorrect results.

From this analysis, we conclude that Randomized C4.5 is certainly competitive with, and probably superior to, Bagged C4.5 in applications where there is relatively little noise in the data.

One disappointing aspect of the results shown in Figure 1 and Table 1 is that randomization did not reach zero error in the letter-recognition domain. In a previous study, Dietterich and Kong (1995) reported an experiment in which 200-fold randomized C4.5 attained perfect performance on the letter-recognition task (training on the first 16,000 examples and testing on the remaining 4,000). We have attempted to replicate that result without success, and we have not been able to determine the source of the discrepancy.

Now let us compare Randomized C4.5 to Adaboosted C4.5. Figure 2 shows that Adaboost is superior to Randomized C4.5 in 7 domains (with indexes 10, 16, 24, 31, 32, and 33 corresponding to king-rook-vs-king (krk), vehicle, sick, king-rook-vs-king-pawn (krkp), audiology, and hypo), while Randomized C4.5 is superior in only 1. The one domain where Randomized C4.5 does better is splice (index 3). We have not been able to identify any particular characteristic of these domains that explains why Adaboosted C4.5 does so well. But the main conclusion is that Adaboosted C4.5 is generally doing as well as or better than Randomized C4.5.

An important issue that has been explored by previous researchers is the question of how well these ensemble methods perform in situations where there is a large amount of classification noise (i.e., training and test examples with incorrect class labels). In his AAAI-96 talk, Quinlan reported some experiments showing that Adaboosted C4.5 did not perform well in these situations. Ali and Pazzani (1996) observed that randomization did not work as well in noisy domains as bagging. However, in their experiments, they only considered ensembles of size 11. We conjectured that larger ensembles might be able to overcome the effects of noise. To explore the effect of classification noise, we added random class noise to nine domains (audiology, hypo, king-rook-vs-king-pawn (krkp), satimage, sick, splice, segment, vehicle, and waveform). These data sets were chosen because at least one pair of the ensemble methods gave statistically significantly different performance on these domains. We did not perform noise experiments with letter-recognition or king-rook-vs-king (krk), because of the huge size of those data sets. To add classification noise at a given rate r, we chose a fraction r of the data points (randomly, without replacement) and changed their class labels to be incorrect (the label for each example was chosen uniformly randomly from the incorrect labels).[2] Then the data were split into 10 subsets for the stratified 10-fold cross-validation (n.b., the stratification was performed using the new labels).

Table 3 shows the win-lose-tie counts for all pairs of learning methods at the four noise levels (0%, 5%, 10%, and 20%). This table reveals some patterns that confirm the observations of Ali and Pazzani and the observations of Quinlan. As we add

Table 3. All pairwise combinations of the four methods for four levels of noise and 9 domains. Each cell contains the number of wins, losses, and ties between the algorithm in that row and the algorithm in that column.

Noise = 0%	C4.5	Adaboost C4.5	Bagged C4.5
Random C4.5	5 – 0 – 4	1 – 6 – 2	3 – 3 – 3
Bagged C4.5	4 – 0 – 5	0 – 5 – 4	
Adaboost C4.5	6 – 0 – 3		

Noise = 5%	C4.5	Adaboost C4.5	Bagged C4.5
Random C4.5	5 – 2 – 2	3 – 2 – 4	1 – 5 – 3
Bagged C4.5	6 – 0 – 3	5 – 1 – 3	
Adaboost C4.5	3 – 3 – 3		

Noise = 10%	C4.5	Adaboost C4.5	Bagged C4.5
Random C4.5	4 – 1 – 4	5 – 1 – 3	1 – 6 – 2
Bagged C4.5	5 – 0 – 4	6 – 1 – 2	
Adaboost C4.5	2 – 3 – 4		

Noise = 20%	C4.5	Adaboost C4.5	Bagged C4.5
Random C4.5	5 – 2 – 2	5 – 0 – 4	0 – 2 – 7
Bagged C4.5	7 – 0 – 2	6 – 0 – 3	
Adaboost C4.5	3 – 6 – 0		

noise to these problems, Randomized C4.5 and Adaboosted C4.5 lose some of their advantage over C4.5 while Bagged C4.5 *gains* advantage over C4.5. For example, with no noise, Adaboosted C4.5 beats C4.5 in 6 domains and ties in 3, whereas at 20% noise, Adaboosted C4.5 wins in only 3 domains and loses in 6! In contrast, Bagged C4.5 with no noise beats C4.5 in 4 domains and ties in 5, but at 20% noise, Bagged C4.5 wins in 7 domains and ties in only 2.

When we compare Bagging and Randomizing to Adaboosted C4.5, we see that with no noise, Adaboost is superior to Bagged C4.5 (5-0-4) and Randomized C4.5 (6-1-2). But with 20% noise, Adaboost is inferior to Bagged C4.5 (0-6-3) and to Randomized C4.5 (0-5-4). Classification noise destroys the effectiveness of Adaboost compared to the other two ensemble methods (and even compared to C4.5 alone in 6 domains).

Finally, when we compare Bagged C4.5 and Randomized C4.5 to each other, we see that with no noise, they are evenly matched (3-3-3). With 20% noise, Bagging has a slight advantage (2 wins, 0 losses, and 7 ties). The high number of ties indicates that Bagging and Randomizing are behaving very similarly as the amount of noise increases.

From this analysis, we can conclude that the best method in applications with large amounts of classification noise is Bagged C4.5, with Randomized C4.5 behaving almost as well. In contrast, Adaboost is not a good choice in such applications.

One further way to gain insight into the behavior of these ensemble methods is to construct κ-error diagrams (as introduced by Margineantu & Dietterich, 1997). These diagrams help visualize the accuracy and diversity of the individual classifiers constructed by the ensemble methods. For each pair of classifiers, we measure their accuracy as the average of their error rates on the test data; we measure their diversity by computing a degree-of-agreement statistic known as κ. We then construct a scatter plot in which each point corresponds to a pair of classifiers. Its x coordinate is the diversity value (κ) and its y coordinate is the mean accuracy of the classifiers.

The κ statistic is defined as follows. Suppose there are L classes, and let C be an $L \times L$ square array such that C_{ij} contains the number of test examples assigned to class i by the first classifier and into class j by the second classifier. Define

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

where m is the total number of test examples. This is an estimate of the probability that the two classifiers agree.

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. The κ statistic corrects for this by computing

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

which estimates the probability that the two classifiers agree by chance, given the observed counts in the table. Specifically, $\sum_{j=1}^{L} \frac{C_{ij}}{m}$ is the fraction of examples that the first classifier assigns to class i, and $\sum_{j=1}^{L} \frac{C_{ji}}{m}$ is the fraction of examples that the second classifier assigns to class i. If each classifier chooses which examples to assign to class i completely randomly, then the probability that they will simultaneously assign a particular test example to class i is the product of these two fractions. In such cases, the two classifiers should have a lower measure of agreement than if the two classifiers agree on which examples they both assign to class i.

With these definitions, the κ statistic is computed as

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

$\kappa = 0$ 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 less than expected by chance—that is, there is systematic disagreement between the classifiers.

Figure 4 shows κ-error diagrams for Randomized C4.5, Bagged C4.5, and Adaboosted C4.5 on the sick dataset. It is illustrative of the diagrams in most of the other domains. We can see that Bagged C4.5 gives a very compact cloud of points. Each point has low error rate and a high value for κ, which indicates that the classifiers are accurate but not very diverse. Randomized C4.5 has a slightly worse error rate but also a more diverse collection of hypotheses. And Adaboosted C4.5 has hypotheses with a wide range of accuracies and degrees of agreement. This clearly

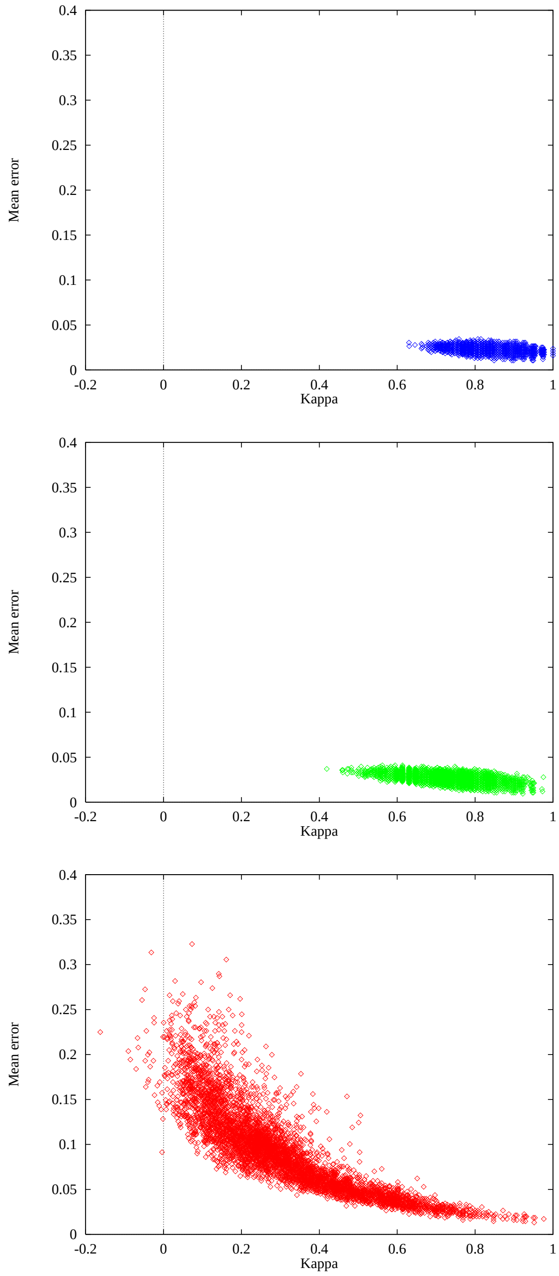

Figure 4. κ-error diagrams for the sick data set using Bagged C4.5 (top), Randomized C4.5 (middle), and Adaboosted C4.5 (bottom). Accuracy and diversity increase as the points come near the origin.

shows the tradeoff between accuracy and diversity. As the classifiers become more accurate, they must become less diverse. Conversely as they become more diverse, they must become less accurate. This shows very dramatically how the Adaboost strategy for constructing ensembles produces much more diverse ensembles than either bagging or randomizing.

While this pattern of accuracy and diversity is observed across many of the data sets, the effect of the pattern on the relative performance is not always the same. As we have seen, Adaboosted C4.5 typically is better than Bagged C4.5 and Randomized C4.5, and this is explained by the much greater diversity of Adaboosted C4.5. But the relative performance of bagging and randomizing is less apparent in the κ-error diagrams. On the sick dataset, for example, Adaboost does better than either Bagging or Randomizing. Bagging and Randomizing are statistically indistinguishable, even though Randomizing has higher diversity.

Figure 5 shows κ-error diagrams for the splice task, which shows the same pattern of relative diversity. Adaboosted C4.5 is more diverse than Randomized C4.5, which is more diverse than Bagged C4.5. In this domain, however, Randomized C4.5 outperforms both boosting and bagging. This can be explained in part because of the many high-error hypotheses that Adaboost also creates (near the top of the κ-error diagram).

The κ-error diagrams also help us understand the effect of noise on the three ensemble methods. Figure 6 shows κ-error diagrams for sick with 20% added classification noise. If we compare these to Figure 4, we can see how the noise affects the three methods. The cloud of points for Randomized C4.5 is basically shifted upward by 0.20, which is what we would expect when 20% classification noise is added. Note, however, that Randomized C4.5 does not become more diverse. In contrast, Bagged C4.5 is shifted upward and to the left—so it becomes substantially more diverse as a result of the noise. And the cloud of points for Adaboosted C4.5 moves close to an error rate of 0.45 (and very small values of κ). This is what we would observe if classifiers are making nearly random guesses. The net result is that Adaboost shifts from being the best method to being the worst, while Randomizing and Bagging remain statistically indistinguishable.

This general pattern is observed in most of the data sets. Noise improves the diversity of Bagging, damages the accuracy of Adaboost severely, and leaves Randomized C4.5 unaffected (aside from the expected shift in error rates).

Figures 7 and 8 show how the segment data set behaves when noise is added. With no noise, Randomized C4.5 is slightly more diverse than Bagged C4.5, and the result is that Randomized C4.5 and Adaboosted C4.5 are tied, and both of them are more accurate than Bagged C4.5. However, when noise is added, the diversity of Randomized C4.5 is hardly changed at all, while the diversity of Bagged C4.5 is substantially increased. Meanwhile, the accuracy of Adaboosted C4.5 is severely degraded, so that many classifiers have error rates greater than 0.5. The net result is that bagging and randomization have equal performance with 20% noise, and both of them are better than Adaboost.

How can these effects be explained? A plausible explanation for the poor response of Adaboost to noise is that mislabeled training examples will tend to receive very

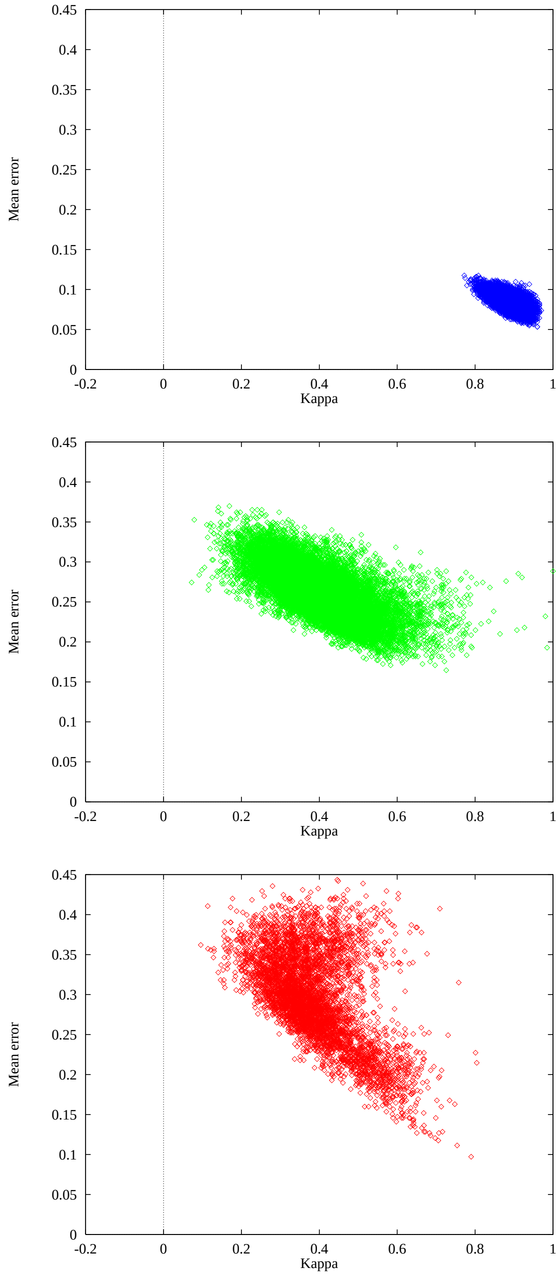

Figure 5. κ-error diagrams for the splice data set using Bagged C4.5 (top), Randomized C4.5 (middle), and Adaboosted C4.5 (bottom).

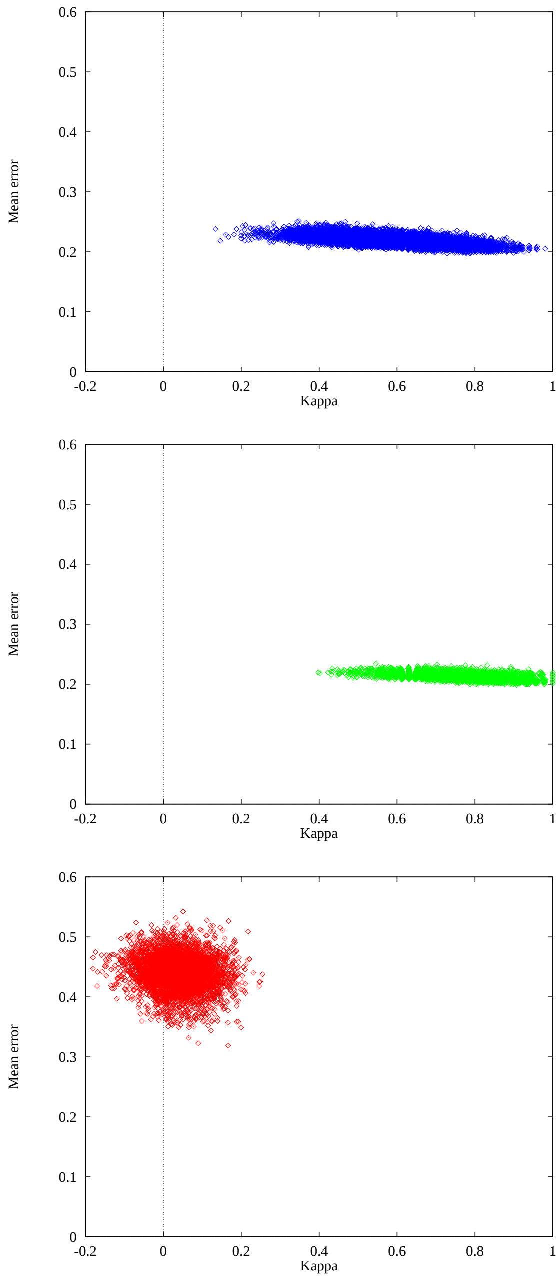

Figure 6. κ-error diagrams for the sick data set with 20% random classification noise. Bagged C4.5 (top), Randomized C4.5 (middle), and Adaboosted C4.5 (bottom).

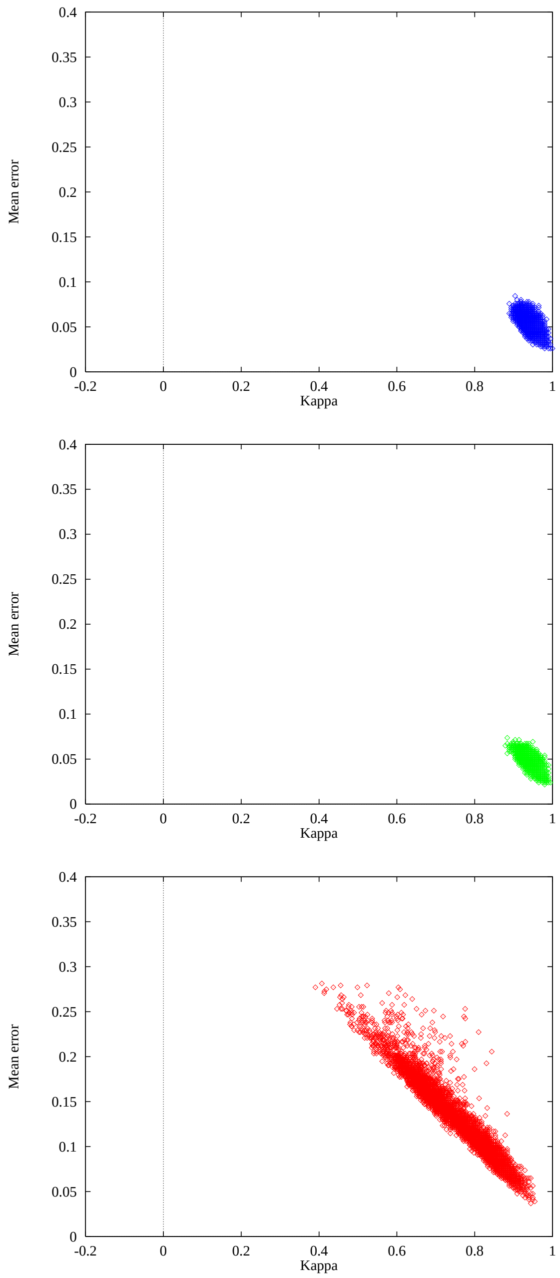

Figure 7. κ-error diagrams for the **segment** data set. Bagged C4.5 (top), Randomized C4.5 (middle), and Adaboosted C4.5 (bottom).

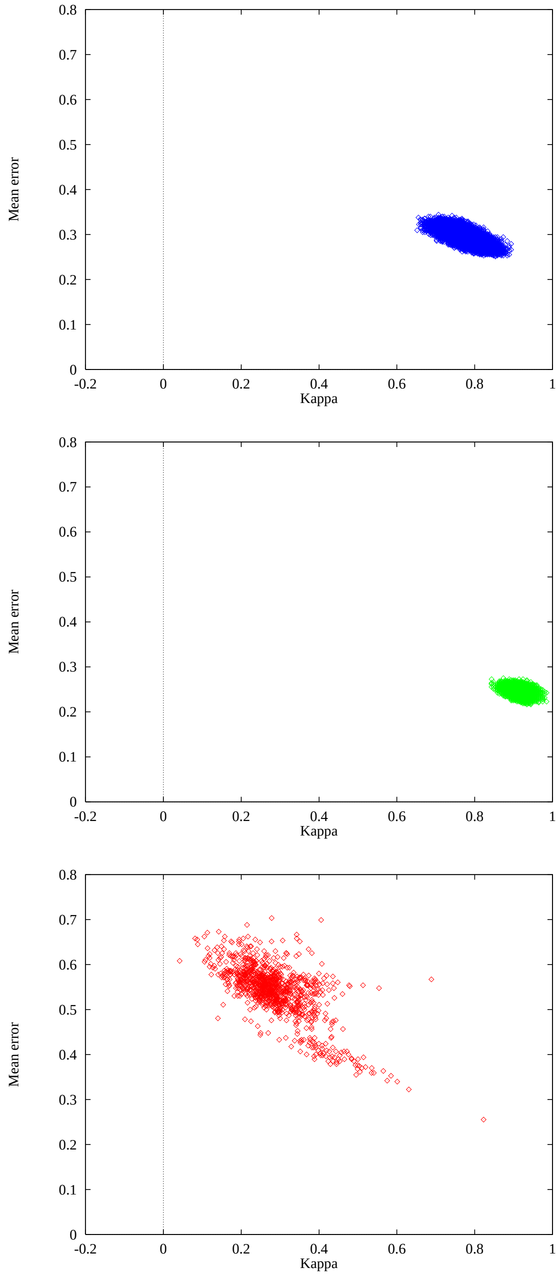

Figure 8. κ-error diagrams for the segment data set with 20% random classification noise. Bagged C4.5 (top), Randomized C4.5 (middle), and Adaboosted C4.5 (bottom).

high weights in the Adaboost algorithm. Hence, after a few iterations, most of the training examples with big weights will be mislabeled examples. A classifier learned from these mislabeled examples will indeed have very low κ values when compared with a classifier learned from the equally-weighted training examples. In fact, one would expect to see negative κ values, and these are observed.

The improved diversity of Bagging could be explained by the following observation. Let us suppose that each mislabeled example can have a substantial effect on the learning algorithm and the classifier that it produces, much the way outliers can have a big effect on linear regression. For example, a mislabeled example can cause C4.5 to split off examples on either side of it, with the result that the training data can become fragmented and the decision tree can become inaccurate. However, in each bootstrap replicate training set, some fraction of the training examples will not appear (in general). Indeed, on average, 36.8% of the training examples will be omitted. Among these omitted examples will be some of the mislabeled training examples, and their omission will lead to large changes in the learned decision tree.

In contrast, Randomized C4.5 never omits any training examples. Hence, even when the splitting decision is randomized, C4.5 still continues making splits until it produces pure (or nearly pure) leaf nodes. So Randomized C4.5 can never ignore any of the mislabeled training examples. This is why its diversity is not affected by the addition of noise. In settings where there is very low noise, Randomized C4.5 produces more diverse classifiers than Bagged C4.5, and this often permits it to do better. Furthermore, the ability of Randomized C4.5 to grow each tree using all of the training data will tend to make each individual tree more accurate. (However, the fact that Randomized C4.5 deliberately makes suboptimal splitting decisions may limit this advantage by reducing the accuracy of the trees.) In settings with moderate amounts of noise, this advantage (of using all of the data) becomes a disadvantage; Bagging becomes more diverse, and occasionally gives better results.

To test the hypothesis that Adaboost is placing more weight on the noisy examples, consider Figure 9. Here we see that much more weight is placed (on the average) on the noisy data points than on the uncorrupted data points. If we consider the fraction of the total weight placed on the corrupted data points, then it rapidly converges to 0.50: Adaboost is placing half of its weight on the corrupted data points even though they make up only 20% of the training set.

It is more difficult to verify the hypothesis concerning the effect of noisy examples on bagging. Further research is needed to explore and test this hypothesis.

4. Conclusions

This paper has compared three methods for constructing ensemble classifiers using C4.5: Randomizing, Bagging, and Boosting. The experiments show that over a set of 33 tasks, Boosting gives the best results in most cases (as long as there is little or no noise in the data). Randomizing and Bagging give quite similar results—there is some evidence that Randomizing is slightly better than Bagging in low noise settings.

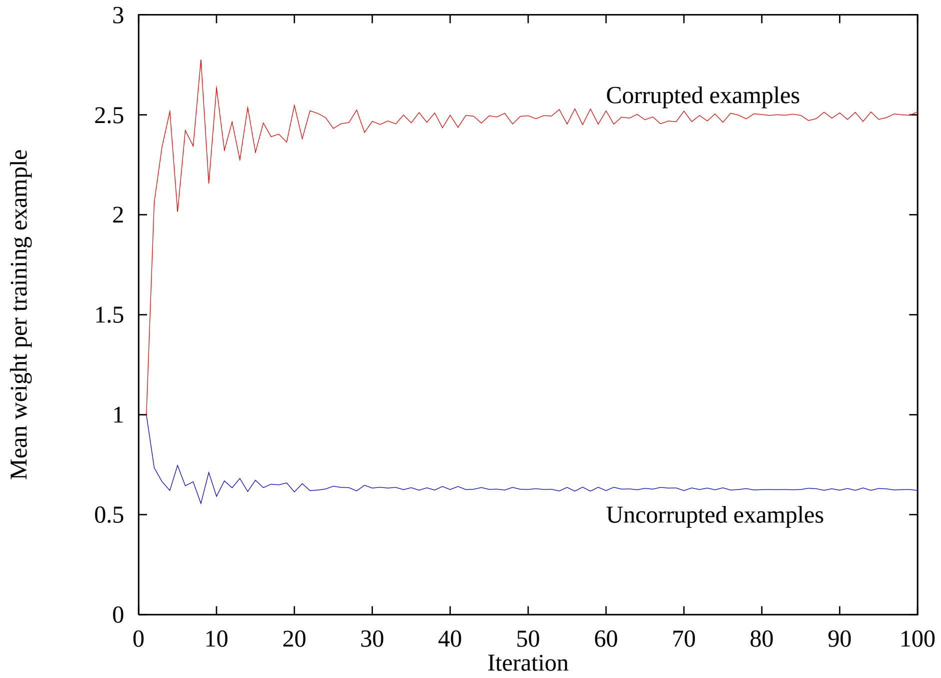

Figure 9. Mean weight per training example for the 560 corrupted training examples and the remaining 2,240 uncorrupted training examples in the sick data set.

With added classification noise, however, Bagging is clearly the best method. It appears to be able to exploit the classification noise to produce more diverse classifiers. The performance of Adaboost can be destroyed by classification noise—the error rates of the individual classifiers become very high. Surprisingly, the performance of Randomized C4.5 with classification noise is not as good as Bagging. Experiments showed that Randomization was not able to increase the diversity of its classifiers as the noise rate increased.

The randomization method that we studied in this paper is very simple: the 20 best candidate splits are computed, and then one of these is chosen uniformly at random. An obvious next step would be to make the probability of chosing a split be proportional to the information gain of that split. Another refinement would be to perform "limited discrepancy" randomization—at most K splits would be randomized within a tree (for some specified value K). The value of K could be set by cross-validation. The algorithm could explicitly consider making 0, 1, 2, ..., K random splits. This would ensure that the "best" tree (i.e., the tree produced by C4.5 itself) would be included in the ensemble. Finally, because randomization can produce trees of different accuracies, it would be worthwhile to consider taking a weighted vote (as in Adaboost), with the weight determined by the accuracy of the tree on the training data. These improvements might make Randomized C4.5 even more competitive with Adaboost in low-noise settings. But without some form of outlier identification and removal, Randomized C4.5 is not likely to do as well as Bagging in high-noise settings.

Acknowledgments

The author gratefully acknowledges the support of the National Science Foundation under NSF Grants IRI-9626584 and CDA-9216172.

Notes

1. In annealing, we treated the unmeasured values as separate attribute values rather than as missing values. In auto, the class variable was the make of the automobile. In the breast cancer domains, all features were treated as continuous. The heart disease data sets were recoded to use discrete values where appropriate. All attributes were treated as continuous in the king-rook-vs-king (krk) data set. In lymphography, the `lymph-nodes-dimin`, `lymph-nodes-enlar`, and `no-of-nodes-in` attributes were treated as continuous. In segment, all features were rounded to four significant digits to avoid roundoff errors in C4.5. In shuttle, all attributes were treated as continuous. In voting-records, the `physician-fee-freeze` attribute was removed to make the task more challenging.
2. Note that other authors have used a different procedure in which with probability r, each training example's label is set to a random class—which may include the original class label or an incorrect class label. Such a procedure only produces a mislabeling rate of $r(k-1)/k$ on the average, where k is the number of classes. Furthermore, in small data sets, it may create levels of mislabeling much higher or much lower than r, whereas the technique we employed guarantees a mislabeling rate of exactly r.

References

Ali, K. M. (1995). A comparison of methods for learning and combining evidence from multiple models. Tech. rep. 95-47, Department of Information and Computer Science, University of California, Irvine.

Ali, K. M., & Pazzani, M. J. (1996). Error reduction through learning multiple descriptions. *Machine Learning*, *24*(3), 173–202.

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, *36*(1/2), 105–139.

Breiman, L. (1994). Heuristics of instability and stabilization in model selection. Tech. rep. 416, Department of Statistics, University of California, Berkeley, CA.

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

Breiman, L. (1996b). Bias, variance, and arcing classifiers. Tech. rep. 460, Department of Statistics, University of California, Berkeley, CA.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*(7), 1895–1924.

Dietterich, T. G., & Kong, E. B. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Tech. rep., Department of Computer Science, Oregon State University, Corvallis, Oregon. Available from `ftp://ftp.cs.orst.edu/pub/tgd/papers/tr-bias.ps.gz`.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pp. 148–146. Morgan Kaufmann.

Kohavi, R., & Kunz, C. (1997). Option decision trees with majority votes. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 161–169 San Francisco, CA. Morgan Kaufmann.

Kohavi, R., Sommerfield, D., & Dougherty, J. (1997). Data mining using MLC++, a machine learning library in C++. *International Journal on Artificial Intelligence Tools*, *6*(4), 537–566.

Maclin, R., & Opitz, D. (1997). An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 546–551 Cambridge, MA. AAAI Press/MIT Press.

Margineantu, D. D., & Dietterich, T. G. (1997). Pruning adaptive boosting. In *Proc. 14th International Conference on Machine Learning*, pp. 211–218. Morgan Kaufmann.

Merz, C. J., & Murphy, P. M. (1996). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

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

Quinlan, J. R. (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.