Bagging with Adaptive Costs

Yi Zhang and W. Nick Street, *Member, IEEE*

***Abstract*— Ensemble methods have proved to be highly effective in improving the performance of base learners under most circumstances. In this paper, we propose a new algorithm that combines the merits of some existing techniques, namely bagging, arcing and stacking. The basic structure of the algorithm resembles bagging. However, the misclassification cost of each training point is repeatedly adjusted according to its observed out-of-bag vote margin. In this way, the method gains the advantage of arcing – building the classifier the ensemble needs – without fixating on potentially noisy points. Computational experiments show that this algorithm performs consistently better than bagging and arcing with linear and nonlinear base classifiers. In view of the characteristics of bacing, a hybrid ensemble learning strategy, which combines bagging and different versions of bacing, is proposed and studied empirically.**

***Index Terms*— Data mining, ensemble methods**

I. Introduction

A TYPICAL ensemble is a multi-classifier system in which each component classifier tries to solve the same task. The final model is obtained by a combination such as a weighted average of the results provided by each of the base classifiers. One wants each individual base model to be accurate, and at the same time, these classifiers should have different inductive biases and thus generalize in distinct ways [1]–[5]. In recent years, many approaches have been taken to systematically create a group of predictors that perform better when combined, such as bagging [6], boosting [7], arcing [8], stacking [9], and random forests [10]. The idea behind bagging and random forests is to introduce randomness into the procedure of building individual classifiers, say, by subsampling the training points. This generates variance among them, and by combining their predictions, the bias of the ensemble converges while the variance gets much smaller than that of the base classifiers. By contrast, boosting and arcing apply a "smart" weighting scheme to the training points, causing the learning algorithm to focus on those examples that were misclassified by the most recently-built classifier (in the case of boosting) or the existing ensemble (arcing). The key idea of stacking [9] is to improve generalization by using a hold-out set to adjust the combination weights of an existing ensemble, so that the error on the hold-out set is minimized. All of these methods have been shown to be effective at improving generalization performance in a wide variety of domains and for base learning algorithms.

While these ensemble methods have proved to be powerful, each has its own drawbacks. Bagging does nothing purposely to reduce the bias so that any bias reduction is solely by chance. Boosting and arcing are very sensitive to outliers and can result in overfitting [3], [11], [12]. Boosting and arcing use training error to tune the weights despite the fact that training error is often highly biased. Stacking requires large amounts of data so that the hold-out set is big enough to approximate the real data distribution, making it unsuitable for small data sets. We propose a new algorithm that attempts to incorporate the merits of these methods while avoiding their downfalls. Based on the structure of the algorithm, we term it as Bagging with Adaptive Costs, or bacing (pronounced as "baking"). Two variations of the bacing algorithm with computational results will be presented. A hybrid ensemble generation strategy which combines bagging and bacing is also studied. Following is an outline of the paper. Section II derives the original bacing algorithm in detail. Section III outlines the computational experiments and Section IV discusses the results. Section V introduces a variation of the original bacing algorithm. The hybrid ensemble generation strategy is presented in Section VI. Section IV concludes the paper.

Zhang is with Microsoft AdCenter, One Microsoft Way, Redmond, WA 98052. Email: yzhan@microsoft.com

Street is with the Management Sciences Department, S232 Pappajohn Business Building, University of Iowa, Iowa City, IA 52242. Email: nick-street@uiowa.edu

II. Bacing Algorithm

Suppose we have a set of classifiers C for a two-class problem and a training set S. S is composed of two subsets, namely A and B, where each data record in A and B belongs to class 1 and 2, respectively. Each classifier C_k will give each record x_i in the training set a classification label c_{ik}. If x_i is classified as class 1 by C_k, then $c_{ik} = 1$ and $c_{ik} = -1$ otherwise. The final classification by the ensemble is decided by looking at the sign of the weighted average of the output of those single classifiers. If this weighted average on a test point is positive, this point is classified as class 1 and vise versa.

If all the data records could be correctly classified, the following inequalities would be satisfied:

$$\sum_{k \in C} c_{ik} w_k \geq 0, \quad \forall i \in A \tag{1a}$$

$$\sum_{k \in C} c_{jk} w_k < 0, \quad \forall j \in B \tag{1b}$$

where w_k is the weight on classifier C_k. The vote margin of each training data point is given by

$$m_i = \sum_{k \in C} c_{ik} w_k, \qquad \forall i \in A \tag{2a}$$

$$m_j = \sum_{k \in C} -c_{jk} w_k, \quad \forall j \in B. \tag{2b}$$

A positive $m_{i(j)}$ implies that the training point is correctly labeled by the current ensemble. The larger the magnitude of a positive $m_{i(j)}$, the more tolerance we can allow for the next built classifier to make a mistake on this certain point.

A negative $m_{i(j)}$ corresponds to a misclassification made by the current ensemble and thus we want the next classifier to contribute a positive vote margin in order to increase the overall vote margin of the ensemble on this point.

Assume that we use a linear support vector machine [13], [14] as the base learner. A linear SVM is a linear separator, or a hyperplane in the feature space that separates the two classes. An L1-linear SVM model can be described as the following linear programming problem,

$$\begin{aligned} \min \quad & \sum_{l \in S} \delta_l e_l + \sum_{n=1}^{N} |\alpha_n| \\ & \sum_{n=1}^{N} \alpha_n x_{in} \geq 1 - e_i, \qquad \forall i \in A \\ & \sum_{n=1}^{N} \alpha_n x_{jn} \leq -1 + e_j, \quad \forall j \in B \\ & e_i, e_j \geq 0. \end{aligned} \tag{3}$$

Here, α represents the coefficient vector of the hyperplane, e_l is proportional to the distance between the corresponding misclassified point and the separating plane, N is the dimension of the feature space, and δ_l is the misclassification cost of point l. We assume that initially, misclassification costs for the points are all equal.

Suppose now we already have an ensemble of linear separators and we want to build a new separator to insert into the existing ensemble. For those points that already possess positive margins, we may allow the new separator to make mistakes on them. But for those points that the existing ensemble does not get right, we should encourage the new separator to classify them correctly. This objective can be achieved by adjusting the misclassification cost of each point according to its margin given by the existing ensemble. Generally, the misclassification costs of those points that have a smaller margin should be tuned up so that misclassifying them will get more costly. A simple linear cost adjustment scheme is as follows:

$$\delta_{l,t+1} = 1 - \frac{m_{l,t}}{1 + |C_t|}. \tag{4}$$

Here, $\delta_{l,t+1}$ is the adjusted misclassification cost of point l in the next round of training, $|C_t|$ is the number of classifiers in the existing ensemble (which is in fact t) and $m_{l,t}$ is the current margin of point l. It should be pointed out that the misclassification cost of a point is essentially the same as the weight of a point in arcing or boosting, in terms of the role it plays in the linear support vector machine used here as the base learner.

The method described so far is close to arcing, though our cost-tuning formula is somewhat different. In arcing, each data point is weighted proportionally to the number of misclassifications by the previously created classifiers

$$\delta_{l,t} = (1 + MC_{l,t})^q, \tag{5}$$

where $MC_{l,t}$ is the number of misclassifications (wrong votes) of the current ensemble on data point l and q is a positive constant. The whole training set is used for building each individual separator and the margin is calculated by resubstitution. As we know, resubstitution error is often biased down compared to true error and hence the resubstitution margin is biased up so that the cost derived will not correctly reflect the relative importance of each point in the next round of training. To fix this over-estimation of margins, the bagging idea can be applied. In bagging, a bootstrap sample (bag) is generated by uniformly sampling points from the training set with replacement. Each separator is built from a different bootstrap sample. For a given bootstrap sample, a point in the training set has probability of approximately 0.632 of being selected at least once. The remaining 36.8% will not be picked, which constitutes a natural hold-out set, often called the out-of-bag set. We may compute the test margin of each point by only aggregating the predictions of separators that are not trained on it and get a more fair estimate. Let T_k represent the subset of the training set that is used to train the kth separator. The modified margin formula is

$$m_i = \sum_{k \in \{k | i \notin T_k\}} c_{ik} w_k, \qquad \forall i \in A \tag{6a}$$

$$m_j = \sum_{k \in \{k | j \notin T_k\}} -c_{jk} w_k, \quad \forall j \in B. \tag{6b}$$

The new cost adjustment scheme becomes

$$\delta_{l,t+1} = 1 - \frac{m'_{l,t}}{1 + |C'_{l,t}|} \tag{7}$$

Here, $C'_{l,t}$ is the current set of separators that are not trained on point l, and $m'_{l,t}$ is the margin for the point obtained by $C'_{l,t}$. Notice that using the out-of-bag performance for weight-tuning is a characteristic of stacking, although here the subject of the adjustment is the misclassification cost instead of the weight on each individual learner. In fact, it is still possible to modify the weight on each separator to minimize the error on the training set. However, our experiments indicate that this leads to (often dramatic) overfitting. The weight on each separator is therefore kept uniformly equal to one throughout the ensemble construction.

As mentioned above, bagging, arcing and stacking each plays a part in the new algorithm. It is hoped that bootstrapped aggregation will reduce the variance components as it does in bagging, cost adjustment will reduce the bias as it does in arcing and the out-of-bag margin estimation will result in better generalization as it does in stacking.

III. Computational Experiments

Bacing was implemented using MATLAB and tested on 14 UCI repository data sets [15]: Autompg, Bupa, Glass, Haberman, Housing, Cleveland-heart-disease, Hepatitis, Ion, Pima, Sonar, Vehicle, WDBC, Wine and WPBC. These data sets were picked mainly because they are relatively small in size so that the SVM problem can be solved in reasonable time. Some of the data sets do not originally depict two-class problems so we did some transformation on the dependent variables to get binary class labels. Specifically in our experiments, Autompg data is labeled by whether the mileage is greater than 25mpg, Housing data by whether the value of the house

```
Input: Training set $T$ of size $n$, base learner $\ell$, integer $k$ (number of training rounds)
let $\delta_j = \frac{1}{n}, \forall j = 1..n$; # initialize cost vector $\delta$
let $m'_j = 0, \forall j = 1..n$; # initialize out-of-bag margin $m$
let $\Gamma_j = 0, \forall j = 1..n$; # initialize out-of-bag counter $\Gamma$
for  $i = 1 : k$
  $T_i = bootstrap(T)$ #bootstrapping $T$ with uniform distribution;
  $C_i = \ell(T_i, \delta)$ # training with the cost sensitive learner $\ell$
  for each  $x_j$
    if  $x_j \notin T_i$
      $m'_j = m'_j + y_j C_i(x_j)$; # update out-of-bag margin
      $\Gamma_j = \Gamma_j + 1$; # update out-of-bag counter
      $\delta_j = 1 - \frac{m'_j}{1+\Gamma_j}$; # update cost
    endif
  end
  Normalize $\delta_j$ so that $\sum_j \delta_j = 1$;
endfor
Output: Ensemble classifier $\{C_1, C_2, \cdots, C_k\}$.
Testing: $C^*(x) = \arg\max_{y \in Y} \sum_{i:\, C_i(x)=y} 1$
```

Fig. 1. Pseudocode for Bacing

exceeds $\$25,000$, Cleveland-heart-disease by the presence or absence of the disease, and Glass by window and non-window varieties. Vehicle and Wine are multi-class problems so we set the problem as separating one class pair each time, resulting in a total of 20 data sets. Discrete variables were changed to binary "dummy" variables for the linear SVM tests, and all data points with missing values were removed. Table I lists the characteristics of the 20 datasets.

In order to observe the characteristics of the algorithm thoroughly, we built 100 classifiers for each run. For an ensemble of such size, we may safely deduce the convergence property of the algorithm and judge whether it will eventually overfit the training data. The results are averaged over five ten-fold cross-validations.

Since bacing can be viewed as a modification of bagging, a comparison was made between bacing and bagging. In each test, the same bootstrap samples were obtained for both algorithms in each round. We also compared bacing with arcing-x4 which was reported to be empirically the best among its variants and comparable with Adaboost [8]. The arcing-x4 weighting scheme sets $q = 4$ in (5). While the weighting schemes of bacing and arcing have somewhat the same flavor, there are still crucial differences that need be stressed between these two algorithms:

- The possible range of weights created by bacing is much smaller than that by arcing-x4. As a result, arcing may finally concentrate on a very limited number of training points (possibly noisy points) and therefore be more prone to overfitting.
- Arcing does not use "out-of-bag" error estimates for weight updating.

The experimental results showed that these modifications do help bacing outperform arcing in most cases. While we focus on comparisons with bagging and arcing, we also include results of the popular Adaboost method for completeness.

IV. Results and Discussion

A. Error Evolution and Cost Adjustment History

The error evolution of bacing and bagging versus the size of the ensemble for some data sets are shown in Figure 2. It can be seen from the plots that both the training and test errors of bacing are consistently below that of bagging. The fluctuation of bacing is stronger during the first several rounds. This may be attributed to the large cost adjustment at the beginning. In the first few rounds, the costs can easily fluctuate between their maximum (2) and minimum (0) values.

Figure 3 illustrates the change of costs over the 100-round run. The y axis represents the norm of the difference of the cost vector between the two neighboring rounds. The plots show that the costs undergo drastic changes at the start, but as a good cost structure is obtained after a few rounds of adjustment, the magnitude of the change slumps quickly and remains stable at a low level thereafter. This low level of adjustment is unavoidable because of the bootstrapping effect. This mode of cost adjustment is quite the opposite of that of boosting, in which the magnitude of weight adjustment increases as the ensemble size gets larger. The extreme change of weights in the end does help boosting drive the training error down by focusing directly on difficult points, however it often causes overfitting if there is classification noise in the data set [7], [16], [17].

It is the simple linear weight adjustment that shapes the behavior of bacing. To examine bacing's behavior, we differentiate between "difficult" points, which are routinely misclassified and may in fact be noisy or otherwise detrimental to good generalization (the sort of points that boosting and arcing tend to focus on), and boundary points, which have a margin near zero. Under our scheme, large costs will be assigned not only to the difficult points in the long run, but also to the boundary points, because if a boundary point is not in the training set, it can easily be misclassified and have its weight tuned up. In this way, the focus on the difficult points is diversified and pulled to the boundary points, which is a desirable effect. Usually, the number of the boundary points is much larger than that of the difficult points so that the trend of the cost adjustment will be influenced more by the boundary points. Since the set of boundary points will become relatively stable after a few rounds, so will the cost structure. As a result of the convergence of the cost structure, the error evolution line of bacing shows no obvious sign of overfitting. Often, the error of bacing stabilizes after a certain number of rounds, which is

TABLE I
CHARACTERISTICS OF DATASETS

Dataset	# of Points	# of Features	Class Distribution (%:%)
autompg	392	9	39.8:60.2
bupa	345	7	42.0:58.0
glass	214	10	23.8:76.2
haberma	306	4	73.5:26.5
heart	297	14	46.1:53.9
hepatit	129	18	81.4:18.6
housing	506	13	22.4:77.6
ion	351	35	35.9:64.1
pima	768	9	34.9:65.1
sonar	208	61	46.6:53.4
vehicle12	429	19	49.4:50.6
vehicle13	430	19	49.3:50.7
vehicle14	411	19	51.6:48.4
vehicle23	435	19	49.9:50.1
vehicle24	416	19	52.2:47.8
vehicle34	417	19	52.3:47.7
wdbc	569	31	62.7:37.3
wine12	130	14	45.4:54.6
wine23	119	14	59.7:40.3
wpbc	188	34	77.1:22.9

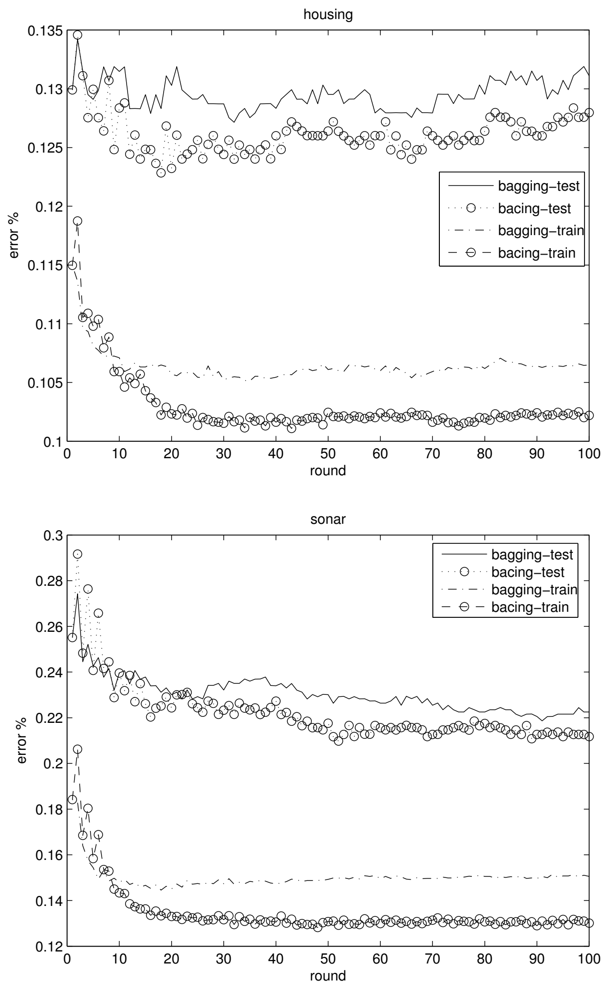


Fig. 2. Error evolution of bacing and bagging

a nice property belonging to bagging. Now we have obtained an algorithm that converges like bagging but can reduce bias by generating classifiers specifically designed to correct the errors of the existing ensemble.

B. Performance Comparison

Table II compares the performance of a single linear support vector machine with 100-round ensembles formed by bacing, bagging, Adaboost, and arcing-x4 on the 20 data sets. The error estimates listed are average over five ten-fold cross-validation runs. Standard deviations are shown in parentheses. Note that Vehicleij stands for the subset of Vehicle with only class i and class j. The same format applies to Wine. Wine13 is omitted from the table because all four algorithms generalized perfectly on it.

Table III and Table IV summarize the comparison results. It is clear that all four ensemble algorithms outperform a single SVM. Though bacing is seldom statistically better than bagging and arcing (2 out of 20), it beats both bagging and arcing 14 times out of the 20 experiments we ran in terms of mean performance. What's more, bacing shows a significant improvement over the base learner six times, compared to two or three for the other methods.

C. Discussion

These results to some extent demonstrate the advantage of bacing over bagging and arcing. Bacing is better than bagging because its iterative weighting scheme enables it to reduce more bias than bagging. Bacing outperforms arcing-x4 because it applies a more conservative cost (weight) adjustment scheme which possibly avoids overfitting and the "out-of-bag" estimation provides more accurate performance information for cost updating.

TABLE II

PERFORMANCE OF LINEAR SVM, BAGGING, ADABOOST, ARCING-X4 AND BACING. VALUES ARE PERCENTAGE ERROR, WITH STANDARD DEVIATION IN PARENTHESES. THE BEST RESULT FOR EACH DATA SET IS BOLDED.

Datasets	SVM	Bagging	Adaboost	Arcing	Bacing
Autompg	9.85 (0.46)	9.60 (0.23)	9.12(1.86)	9.29 (1.18)	**8.83 (0.44)**
Bupa	31.07 (0.44)	30.84 (0.82)	30.61(3.70)	31.17 (3.02)	**30.38 (0.50)**
Glass	28.13 (0.43)	27.74 0.86)	29.31(1.25)	27.12 (1.60)	**27.08 (1.25)**
Haberman	28.10 (0.64)	27.77 (0.60)	25.81(0.95)	25.89 (1.33)	**25.50 (0.40)**
Heart	**16.48 (0.49)**	**16.48 (0.49)**	17.96(1.52)	17.64 (3.18)	17.14 (1.15)
Hepatitis	15.50 (1.58)	**13.47 (1.17)**	14.28(2.49)	14.86 (4.47)	13.94 (1.45)
Housing	12.95 (0.33)	13.11 (0.66)	12.18(1.58)	**11.83 (1.77)**	12.68 (0.49)
Ion	12.42 (0.97)	11.51 (0.92)	**10.60(1.38)**	11.06 (2.12)	11.40 (0.89)
Pima	23.26 (0.36)	23.02 (0.36)	**22.76(1.82)**	23.30 (1.41)	22.94 (0.39)
Sonar	23.69 (1.18)	22.25 (1.27)	21.20(2.84)	**19.36 (2.13)**	20.97 (1.55)
Vehicle12	33.46 (0.73)	33.23 (0.89)	32.75(3.64)	32.78 (2.55)	**32.44 (0.44)**
Vehicle13	**1.16 (0.23)**	1.63 (0.16)	2.19(0.63)	3.35 (0.35)	1.77 (0.27)
Vehicle14	2.97 (0.80)	2.23 (0.21)	3.87(0.62)	3.21 (1.00)	**2.14 (0.20)**
Vehicle23	3.03 (0.60)	2.76 (0.43)	**2.40(0.76)**	4.24 (0.70)	2.62 (0.26)
Vehicle24	2.45 (0.40)	**2.26 (0.13)**	3.53(1.47)	2.88 (0.52)	**2.26 (0.13)**
Vehicle34	2.11 (0.26)	1.87 (0.39)	**1.20(0.24)**	3.07 (0.78)	2.11 (0.35)
WDBC	4.71 (0.31)	4.74 (0.21)	**3.69(1.36)**	3.94 (0.61)	4.50 (0.16)
Wine12	2.77 (0.42)	3.08 (0.77)	2.15(0.34)	**2.00 (1.69)**	2.46 (0.34)
Wine23	4.02 (1.08)	3.18 (0.34)	**3.10(0.56)**	4.73 (2.05)	3.68 (0.44)
WPBC	24.98 (0.35)	23.50 (0.98)	23.13(3.86)	**22.66 (2.44)**	23.05 (0.86)
# of Best	2	3	6	4	7

TABLE III

SUMMARY OF PERFORMANCE COMPARISON RESULTS AMONG SVM, BAGGING, ADABOOST ARCING AND BACING

Algorithm Pair	W-L-T
Bagging vs. SVM	15-4-1
Adaboost vs. SVM	15-5-0
Arcing vs. SVM	11-9-0
Bacing vs. SVM	17-2-1
Bacing vs. Bagging	14-5-1
Bacing vs. Adaboost	12-8-0
Bacing vs. Arcing	14-6-0

TABLE IV

SUMMARY OF SIGNIFICANT t-TEST RESULTS AMONG SVM, BAGGING, ADABOOST ARCING AND BACING

Algorithm Pair	W-L
Bagging vs. SVM	2-1
Adaboost vs. SVM	3-1
Arcing vs. SVM	2-1
Bacing vs. SVM	6-1
Bacing vs. Bagging	2-0
Bacing vs. Adaboost	2-1
Bacing vs. Arcing	2-0

In addition, the performance of bacing is more stable than that of arcing-x4 (or boosting), which leads to smaller standard deviations and makes the improvement over the base classifier more likely to be statistically significant. The distribution of weights may explain why bacing is a more stable algorithm than arcing. Figure 4 shows the change of weights (normalized) of arcing over the 100-round run. Obviously, the normalized weights of arcing converge in the long run like those of bacing. However, the convergence states of the two algorithms are different.

Figure 5 compares the distribution of point weights of bacing and arcing-x4 at the final round by histograms. (The WPBC data set as was used for this example. The plots of other data sets are similar.) On both plots, there is a large portion of points accumulated around zero. These are the "easy points." The major difference is on the other end of the graph. Arcing-x4 heavily weights a very limited number of data points, which may cause large fluctuations in performance among different folds in the cross-validation since a small change in the training set may make a big difference on the separating surface because of the over-concentration. On the other hand, the effectively non-zero weights generated by bacing are more spread over the training data. As mentioned before, bacing is able to concentrate on both the boundary points and the difficult points. This smoother distribution of non-zero weights reduces the risk of overfitting and makes the performance of the algorithm more robust.

D. Out-of-Bag Margin and In-Bag Margin

An important characteristic of bacing is that it uses out-of-bag margin for cost updating. To the best of the author's knowledge, all other existing sequential weight-updating ensemble algorithms such as boosting and arcing all use in-bag error estimation. The advantage of out-of-bag estimation is that it is a more fair estimate compared with resubstitution. However, there is also a downside. The probability of each point being sampled in a uniform bagging is about 63%. Therefore, the out-of-bag margin of each point is calculated

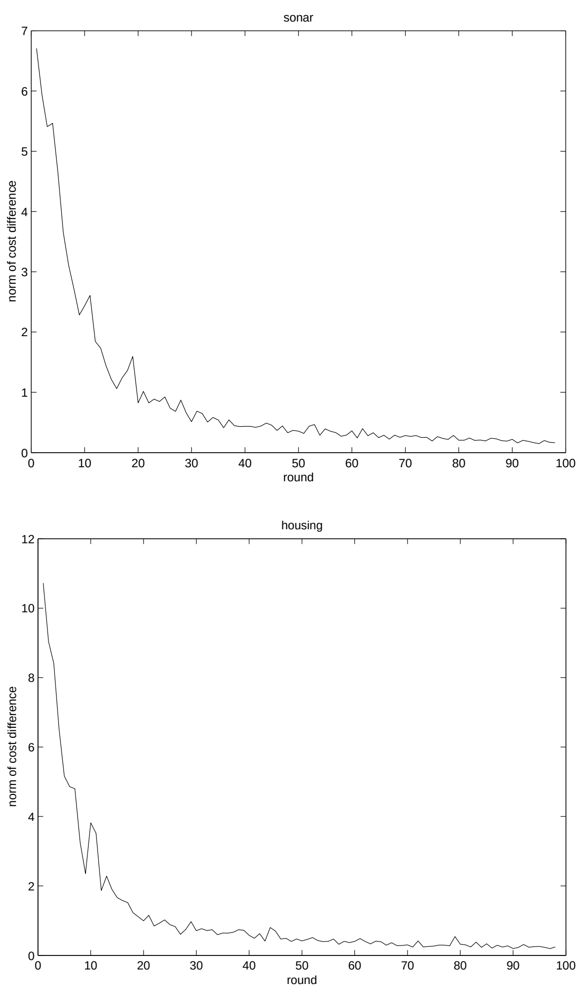


Fig. 3. Change of misclassification costs, bacing

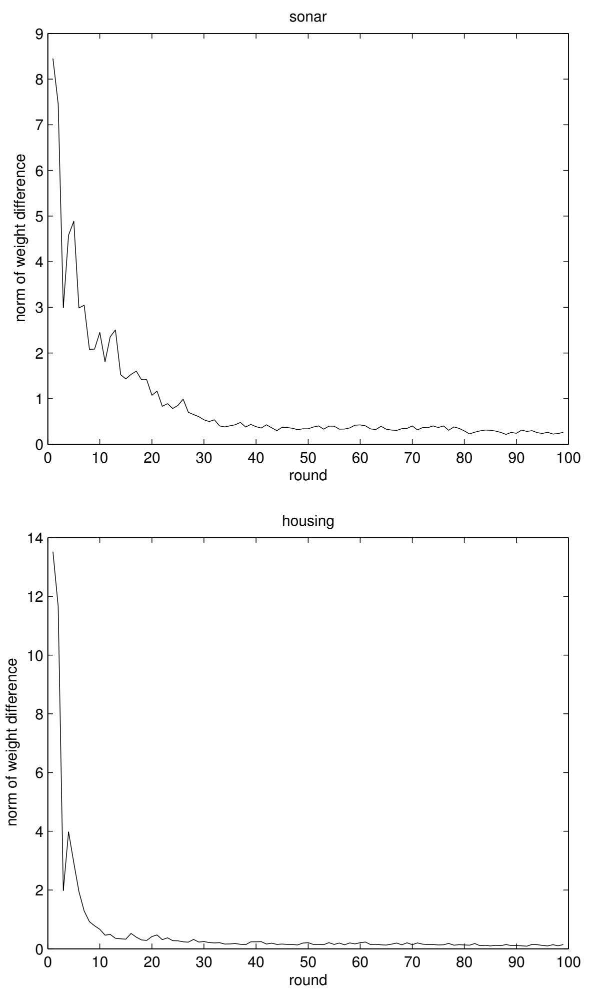


Fig. 4. Change of weights, arcing-x4

only based on roughly one-third of the ensemble classifiers, which may cause insufficient information. Whether out-of-bag margin will bring in better performance is hence a trade-off between fair estimation and sufficient information.

We did experiments to see how these two error estimation perform empirically. Two bacing ensembles of size 100 with linear SVM as base classifier are created on each dataset, one based on in-bag margin calculation and the other on out-of-bag. The results are shown in Table V.

In terms of performance, these two versions of bacing are not significantly different. However, the out-of-bag bacing seems to be a more stable algorithm by looking at the standard deviations of classification errors. In most cases, the standard deviations of out-of-bag bacing are much lower than that of in-bag bacing. Therefore, when compared to other algorithms, out-of-bag bacing is more likely to be significantly different, as was seen in Table IV.

E. Bacing with Decision Trees

Previous experiments use a linear SVM as the base classifier. Linear classifiers generally display high bias and low variance. The variance reduction effect of an ensemble is not a major factor leading to better performance with a linear classifier as the base leaner. Therefore, to observe the ability of bacing to reduce variance, experiments with nonlinear base classifiers were conducted.

We chose unpruned C4.5 decision tree [18] as the base classifier for the experiments.The C4.5 decision tree algorithm is fast, easy to implement and its corresponding classification function is highly nonlinear, resulting in large variance. The only problem is that it is not designed as a cost-sensitive classifier. However, we are able to get around the problem by treating the misclassification cost on each data point as a resampling prior, which has an equivalent effect [3], [8]. The algorithm is shown in Figure 6.

The experiments were conducted on the same UCI datasets as with the linear SVM. Since bagging is known to be one of the best variance reduction ensemble algorithms, a performance comparison between bagging and bacing is made. Table VI shows the training and test error of bacing and bagging, as well as the test error of a single C4.5 tree, arcing, and boosting. Table VII and VIII summarize the comparison results based on mean value and paired t test.

With a low-bias, high-variance classifier such as a C4.5 decision tree, the performance improvement of an ensemble

TABLE V
PERFORMANCE COMPARISON BETWEEN IN-BAG BACING AND OUT-OF-BAG BACING. LINEAR SVM AS BASE CLASSIFIER. VALUES ARE PERCENTAGE TEST ERROR, WITH STANDARD DEVIATION IN PARENTHESES.

dataset	In-Bag Bacing	Out-of-bag Bacing
autompg	9.54(1.40)	**8.83(0.44)**
bupa	31.08(3.49)	**30.38(0.50)**
glass	27.80(0.60)	**27.28(1.25)**
haberma	**24.84(2.32)**	25.50(0.40)
heart	**16.93(1.78)**	17.14(1.15)
hepatit	14.45(3.53)	**13.94(1.45)**
housing	**12.29(0.32)**	12.68(0.49)
ion	**10.95(0.95)**	11.40(0.89)
pima	23.05(0.83)	**22.94(0.39)**
sonar	21.04(3.36)	**20.97(1.55)**
vehicle12	**32.17(1.20)**	32.44(0.44)
vehicle13	**1.49(0.61)**	1.77(0.27)
vehicle14	**1.99(0.47)**	2.14(0.20)
vehicle23	3.13(0.90)	**2.62(0.26)**
vehicle24	**2.11(0.71)**	2.26(0.13)
vehicle34	**1.68(1.12)**	2.11(0.35)
wdbc	4.60(1.18)	**4.50(0.16)**
wine12	**2.31(0.77)**	2.46(0.34)
wine23	**3.21(0.72)**	3.68(0.44)
wpbc	**21.78(3.28)**	23.05(0.86)
Out vs. In	Absolute W-L-T	8-12-0
Out vs. In	Significant W-L-T	0-0-20

TABLE VI
PERFORMANCE OF C4.5, BAGGING, ADABOOST, ARCING AND BACING. VALUES ARE PERCENTAGE ERROR, WITH STANDARD DEVIATION IN PARENTHESES. THE BEST RESULT FOR EACH DATA SET IS BOLDED.

Dataset	C45-test	Bag-train	Bac-train	Bag-test	Ada-test	Arc-test	Bac-test
autompg	11.06(2.46)	0.87(0.10)	0.00(0.00)	**10.56(1.41)**	10.67(1.80)	10.77(1.80)	10.71(1.39)
bupa	35.93(3.08)	1.10(0.26)	0.00(0.00)	30.49(2.56)	**28.90(2.25)**	30.61(1.45)	30.19(3.12)
glass	24.48(4.07)	1.63(0.46)	0.00(0.00)	17.10(1.54)	14.53(3.04)	**13.44(5.55)**	14.35(2.81)
haberma	35.76(4.65)	6.19(0.18)	1.76(0.15)	**30.79(2.35)**	33.99(3.43)	34.25(1.18)	33.93(1.96)
heart	27.26(4.44)	0.90(0.14)	0.00(0.00)	19.87(2.78)	20.18(1.65)	**19.83(3.73)**	21.57(2.44)
hepatitis	20.94(1.91)	2.39(0.25)	0.00(0.00)	17.23(3.11)	16.24(3.78)	15.99(3.82)	**15.36(2.35)**
housing	16.56(1.48)	0.66(0.08)	0.00(0.00)	12.73(0.92)	11.47(1.35)	**10.78(1.05)**	11.36(1.60)
ion	8.94(1.39)	0.44(0.11)	0.00(0.00)	6.15(0.91)	5.75(1.03)	**5.73(0.87)**	5.98(0.51)
pima	29.09(3.27)	1.24(0.02)	0.00(0.00)	24.37(1.38)	24.56(2.52)	**24.25(1.30)**	25.88(1.12)
sonar	30.00(3.39)	0.09(0.07)	0.00(0.00)	24.73(1.70)	22.56(4.75)	22.76(3.68)	**19.13(1.80)**
vehicle12	42.10(2.70)	0.35(0.07)	0.00(0.00)	42.01(1.80)	41.88(2.85)	42.60(2.16)	**41.32(2.44)**
vehicle13	5.02(1.92)	0.20(0.05)	0.00(0.00)	3.81(1.02)	1.83(0.75)	2.19(0.63)	**1.81(0.60)**
vehicle14	7.01(1.42)	0.19(0.09)	0.00(0.00)	5.26(1.25)	4.83(1.21)	5.01(1.04)	**4.72(0.79)**
vehicle23	5.37(1.41)	0.42(0.06)	0.00(0.00)	4.19(0.64)	**2.67(1.31)**	2.83(1.09)	2.72(0.88)
vehicle24	8.41(1.83)	0.14(0.04)	0.00(0.00)	5.73(1.64)	**4.27(1.94)**	4.66(0.63)	5.00(0.55)
vehicle34	1.97(1.06)	0.27(0.02)	0.00(0.00)	1.40(0.10)	1.34(0.62)	**1.10(0.77)**	1.30(0.14)
wdbc	4.67(1.54)	0.54(0.07)	0.00(0.00)	4.01(0.74)	3.91(0.77)	**3.34(0.60)**	3.80(0.56)
wine12	3.08(2.05)	0.38(0.19)	0.00(0.00)	**2.62(0.88)**	2.92(1.48)	3.08(1.72)	2.77(1.29)
wine23	6.56(2.16)	0.04(0.05)	0.00(0.00)	**3.20(1.81)**	3.38(1.59)	3.35(1.44)	4.23(2.85)
wpbc	32.20(6.19)	1.02(0.27)	0.00(0.00)	24.19(3.64)	24.23(2.21)	24.15(1.81)	**23.94(1.78)**

```
Input: Training set $T$ of size $n$, base learner $\ell$, integer $k$ (number of training rounds)
let $\delta_j = \frac{1}{n}, \forall j = 1..n$; # initialize cost vector $\delta$
let $m'_j = 0, \forall j = 1..n$; # initialize out-of-bag margin $m$
let $\Gamma_j = 0, \forall j = 1..n$; # initialize out-of-bag counter $\Gamma$
for  $i = 1 : k$
  $T_i = bootstrap(T, \delta)$ # bootstrapping $T$ with sampling prior $\delta$
  $C_i = \ell(T_i)$ #training with C4.5 tree learner $\ell$ and training set $T_i$
  for each  $x_j$
    if  $x_j \notin T_i$
      $m'_j = m'_j + y_j C_i(x_j)$; # update out-of-bag margin
      $\Gamma_j = \Gamma_j + 1$; # update out-of-bag counter
      $\delta_j = 1 - \frac{m'_j}{1+\Gamma_j}$; # update cost
    endif
  end
  Normalize $\delta$ so that $\sum_j \delta_j = 1$;
endfor
Output: Ensemble classifier $\{C_1, C_2, \cdots, C_k\}$.
Testing: $C^*(x) = \arg\max_{y \in Y} \sum_{i:\, C_i(x)=y} 1$
```

Fig. 6. Pseudocode for Bacing with C4.5

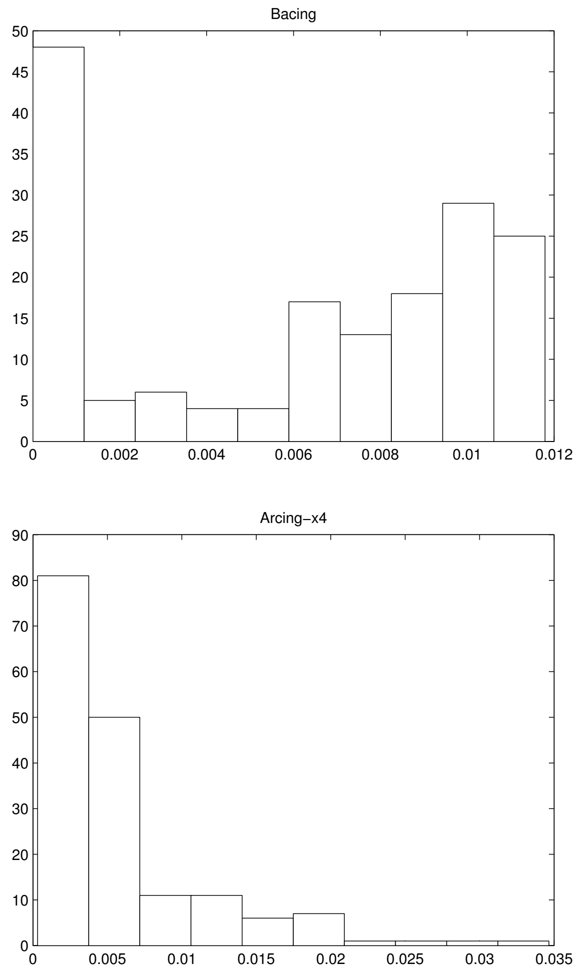


Fig. 5. Distribution of weights, bacing and arcing-x4

TABLE VII
SUMMARY OF PERFORMANCE COMPARISON RESULTS AMONG C4.5, BAGGING, ADABOOST, ARCING AND BACING

Algorithm Pair	W-L-T
Bagging vs. C4.5	20-0-0
Adaboost vs. C4.5	20-0-0
Arcing vs. C4.5	18-1-1
Bacing vs. C4.5	20-0-0
Bacing vs. Bagging	14-6-0
Bacing vs. Adaboost	12-8-0
Bacing vs. Arcing	11-9-0

TABLE VIII
SUMMARY OF SIGNIFICANT t-TEST RESULTS AMONG C4.5, BAGGING, ADABOOST, ARCING AND BACING

Algorithm Pair	W-L
Bagging vs. C4.5	12-0
Adaboost vs. C4.5	13-0
Arcing vs. C4.5	13-0
Bacing vs. C4.5	13-0
Bacing vs. Bagging	5-2
Bacing vs. Adaboost	0-0
Bacing vs. Arcing	0-0

algorithm like bagging or boosting is substantial. None of the ensembles we tested were ever worse than a single decision tree. Bagging significantly improves accuracy 12 times while bacing does it 13 times out of 20 datasets. Since a single C4.5 tree is itself complex enough in most cases, the error reduction of the ensemble algorithm comes mainly from variance reduction. In terms of improving a single C4.5 tree, bagging and bacing perform almost equally well. Therefore, we may conclude that bacing is as competent as bagging for variance reduction. On the other hand, the significant win-loss statistic between bacing and bagging is 5-2. This is not surprising because the cost-updating scheme enables bacing to reduce bias whenever possible. Interestingly, for most of the datasets, bacing is able to drive the training error to be zero. Although the cost updating scheme of bacing seems ad hoc, it is able to empirically achieve the same goal of Adaboost, whose weighting scheme is well-crafted. Bacing and arcing perform similarly in the nonlinear classifier case. However, bacing consistently reduces the variation across different runs;

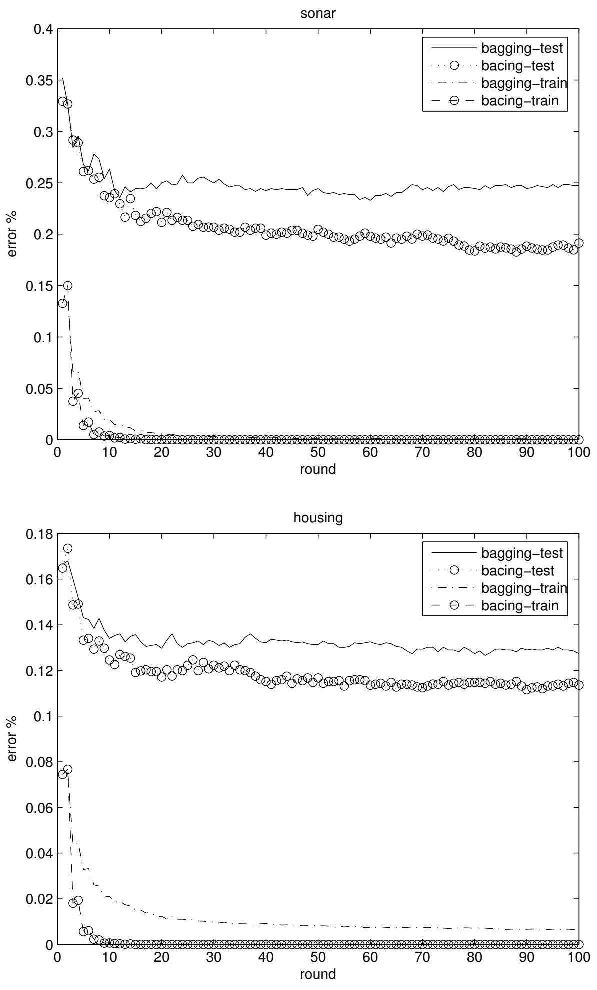


Fig. 7. Error evolution of bacing with C4.5 as base classifier, for Sonar and House data sets.

the standard deviation of the five cross-validation runs is lower than arcing on 15 of the 20 tests.

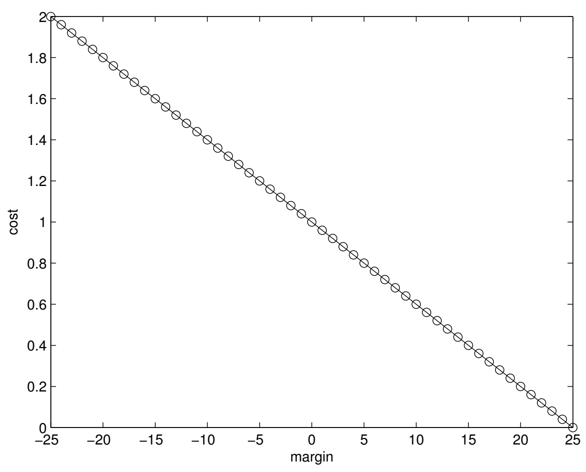
The training and testing error evolution curves over the 100-run of the House and Sonar datasets are displayed in Figure 7. Similar to the linear SVM case, the training and testing errors of bacing are consistently below those of bagging.

V. Absolute Margin: A New Cost Updating Scheme

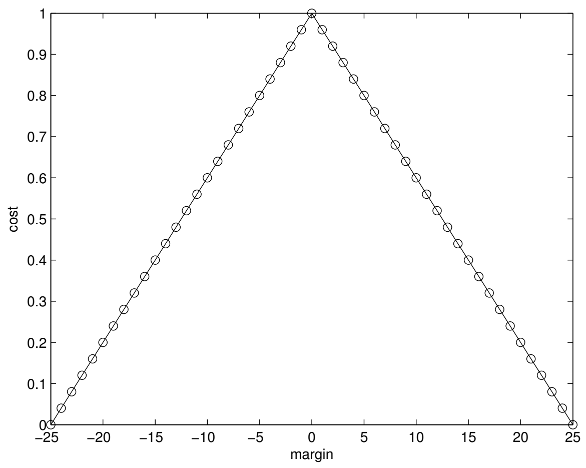
In the current bacing setting, the cost updating scheme is fixed throughout the ensemble generation process. The out-of-margin based cost adjustment scheme (7) puts emphasis on those points that are misclassified by the current ensemble. Although the bootstrapping step alleviates the problem of overfitting compared with boosting when there is a certain amount of noise in the training set, the possibility of overfitting never goes away. An alternative cost adjustment scheme, as shown in 8, seems to be able to handle the overfitting problem.

$$\delta_{l,t+1} = 1 - \frac{|m'_{l,t}|}{1 + |C'_{l,t}|} \tag{8}$$

The only difference between scheme (7) and (8) is that (8) puts an absolute value on the out-of-bag margin $m'_{l,t}$. Figure 8



(a) $\delta_{l,t+1} = 1 - \frac{m'_{l,t}}{1+|C'_{l,t}|}$



(b) $\delta_{l,t+1} = 1 - \frac{|m'_{l,t}|}{1+|C'_{l,t}|}$

Fig. 8. Cost curves of two updating schemes, with $|C'_{l,t}| = 25$

illustrates the two schemes.

As a result, under scheme (8), the emphasis is no longer put on the points having large negative margins, but on those points with margins with small magnitude, either on the positive side or the negative side. Noisy points that will often be misclassified by most classifiers and accumulate negative margins will ultimately take a very small weight and effectively be ignored in the later rounds of training. Therefore, this scheme will theoretically not result in overfitting. However, the downside of this scheme is also obvious. In the first several rounds of training, it is not able to differentiate boundary points from difficult / noisy points. Although like the noisy points, boundary points can often be misclassified, they are important training points for the learning process to form the classification boundary. Under the alternative cost updating scheme, some boundary points may be de-emphasized if they are not correctly classified by the first several classifiers.

From now on, we call bacing with scheme (7) as “bacing-0” and bacing with scheme (8) as “bacing-1”. Table IX compares the performance of these two versions of bacing on the 20 UCI datasets. C4.5 trees are used as the base classifier. One hundred classifiers are built for each ensemble. Bagging results are also listed as the benchmark. A simple comparison statistic of how many times each algorithm performs the best among three algorithms (in terms of mean error) is listed at the end of the table.

Both of the two versions of bacing perform substantially better than bagging. There is no big difference between them in terms of test error. Generally speaking, bacing-0 is good at exploring. It attempts to modify the ensemble classification function to maximize the number of points it can correctly classify. On the other hand, bacing-1 is good at focusing on and fixing the classification boundary. Its major effort is put on increasing the margin of the boundary points. It turns out that both the strategies are effective.

VI. Combining Bagging and Bacing

Currently, the bacing algorithm starts weight-tuning right after the first classifier is built. It is however questionable whether this is the best way to exploit the boosting effect of the cost adjustment scheme. In fact, the later classifiers built are dependent on the first classifier since their target is to correct its errors, directly or indirectly. Therefore, if the first classifier takes a wrong direction by chance, the whole ensemble may be led astray or it may take many iterations to come back on the right track.

Under the current setup, the first classifier is built on a uniformly bootstrapped set which involves a bit of randomness. There is a possibility that the first classifier, with the next several classifiers built heavily dependent on it, is not representative enough in terms of identifying easy points, boundary points and difficult points, which is essential for the success of bacing. Hence, we consider starting bacing with an initial set of classifiers with less such bias introduced by randomness.

A simple strategy is to build a bagging ensemble as a start. This initial bagging ensemble may create a good picture of the learnability of the training points. The accumulated margin on each data point of this bagging ensemble should be more trustworthy than that of a single classifier. Therefore, we expect that this hybrid strategy, which essentially fuses bagging and bacing, may produce better results.

Table X lists the computation experiment results of combining bagging and bacing. Since there are two versions of bacing, namely bacing-0 and bacing-1, bagging is combined with each of them and each combined ensemble is compared with the original bacing ensemble. Each hybrid ensemble starts with 50 rounds of bagging followed by 50 rounds of bacing. The size of the pure bacing ensembles are 100 as well.

The empirical results shows that combining bagging with bacing-0 generally improves the performance. As mentioned before, the initial bagging helps create better margin information and thus the cost adjustment scheme of bacing-0 is able to act more accurately to modify the separating surface. On the other hand, the initial bagging seems not very helpful for bacing-1. Our explanation is bacing-1’s lack of initiative of exploring when the separating surface becomes clear. After bagging, the accumulated margin more clearly defines the boundary and bacing-1 can only slightly tune this boundary. It simply ignores those data points with a large negative margin after bagging, which might still be correctable.

For an ideal sequential ensemble algorithm, at the beginning, it should be able to explore the potential of the classification function to make it fit as many training points as possible. Then it should focus on the gradually-shaped boundary to increase margins of the boundary points. In terms of the bias-variance tradeoff, the former part is for bias reduction and the latter part for variance reduction. Interestingly, bacing-0 and bacing-1 are somehow following these two kinds of strategies respectively, regarding their cost updating schemes. Therefore, if we combine these two versions of bacing, we might get closer to an ideal sequential ensemble.

Depending on whether we start the ensemble-building process with some initial rounds of bagging, we have two new hybrid algorithms: bacing-0 plus bacing-1, and bagging plus bacing-0 plus bacing-1. We compare these two algorithms with the best algorithm so far: bagging plus bacing-0.

In each of the different ensemble setups, 50 rounds were run for every component, namely bagging, bacing-0 or bacing-1. Again, C4.5 was picked as the base classifier.

The computational experiments show that bagging plus bacing-0 dominates the other two hybrid algorithms (see Table XI). It seems that the bacing-1 stage is unnecessary, if not harmful, because the first 100 rounds of “Bag+Bac0” and “Bag+Bac0+Bac1” are exactly the same. The extra 50 rounds of bacing-1 failed to help boost the performance in most cases. As Figure 9 shows, the test error stops going down during the bacing-1 stage. We conjecture that bacing-1’s over-attention to the well-established boundary found by bagging and bacing-0 may lead to the slight deterioration in performance in some cases.

Although the preliminary computational results are not satisfactory, it is still early to relinquish the idea of combining the three seemingly complementary algorithms. Changing the combination strategy or slightly revising the cost-updating schemes of bacing algorithm might have a positive impact on the hybrid strategy.

VII. Conclusions and Future Work

The bacing algorithm combines some of the nice features of bagging, boosting and stacking. An effective boosting-like bias correction mechanism is added into the standard bagging method. The bootstrapping procedure of bagging naturally provides a hold-out set that helps the correction mechanism get a fair estimate on the performance of the current ensemble. Therefore, it is expected that bacing will be a generally better method than bagging and arcing. The computational experiments supported the conjecture for two kinds of base learners, linear support vector machines and decisions trees.

The algorithm itself is simple to implement, adds very little computational complexity to bagging and can be applied

TABLE IX
PERFORMANCE COMPARISON AMONG BAGGING, BACING-0 AND BACING-1. C4.5 AS BASE CLASSIFIER. VALUES ARE PERCENTAGE TEST ERROR, WITH STANDARD DEVIATION IN PARENTHESES.

Dataset	**Bagging**	**Bacing-0**	**Bacing-1**
autompg	**10.56(1.41)**	10.71(1.39)	10.86(2.34)
bupa	30.49(2.56)	30.19(3.12)	**29.21(2.10)**
glass	17.10(1.54)	**14.35(2.81)**	16.45(1.93)
haberma	30.79(2.35)	33.93(1.96)	**30.14(1.47)**
heart	**19.87(2.78)**	21.57(2.44)	20.48(2.54)
hepatit	17.23(3.11)	15.36(2.35)	**15.22(3.13)**
housing	12.73(0.92)	**11.36(1.60)**	12.68(0.89)
ion	6.15(0.91)	5.98(0.51)	**5.93(0.89)**
pima	24.37(1.38)	25.88(1.12)	**24.33(2.68)**
sonar	24.73(1.70)	**19.13(1.80)**	22.67(3.02)
vehicle12	42.01(1.80)	**41.32(2.44)**	42.14(0.97)
vehicle13	3.81(1.02)	**1.81(0.60)**	1.95(0.63)
vehicle14	5.26(1.25)	**4.72(0.79)**	4.96(0.70)
vehicle23	4.19(0.64)	**2.72(0.88)**	3.13(1.03)
vehicle24	5.73(1.64)	**5.00(0.55)**	6.63(0.98)
vehicle34	1.40(0.10)	**1.30(0.14)**	1.63(0.65)
wdbc	4.01(0.74)	**3.80(0.56)**	**3.80(0.70)**
wine12	2.62(0.88)	**2.77(1.29)**	2.92(2.13)
wine23	**3.20(1.81)**	4.23(2.85)	**3.20(1.59)**
wpbc	24.19(3.64)	23.94(1.78)	**23.26(5.05)**
# of Wins	3	11	8

TABLE X
PERFORMANCE COMPARISON BETWEEN PURE BACING AND BACING COMBINED WITH BAGGING. C4.5 AS BASE CLASSIFIER. VALUES ARE PERCENTAGE TEST ERROR, WITH STANDARD DEVIATION IN PARENTHESES.

Dataset	**Bac0**	**Bag+Bac0**	**Bac1**	**Bag+Bac1**
autompg	10.71(1.39)	**10.51(1.98)**	10.86(2.34)	**10.77(1.10)**
bupa	30.19(3.12)	**28.56(3.76)**	**29.21(2.10)**	29.73(2.50)
glass	**14.35(2.81)**	15.00(2.48)	**16.45(1.93)**	17.11(3.77)
haberman	33.93(1.96)	**32.18(3.98)**	**30.14(1.47)**	30.90(3.51)
heart	21.57(2.44)	**20.60(1.97)**	20.48(2.54)	**20.09(2.27)**
hepatitis	**15.36(2.35)**	15.53(2.98)	**15.22(3.13)**	16.14(0.79)
housing	**11.36(1.60)**	12.05(1.41)	**12.68(0.89)**	12.96(1.11)
ion	5.98(0.51)	**5.65(1.62)**	5.93(0.89)	**5.81(1.31)**
pima	25.88(1.12)	**24.76(1.93)**	**24.33(2.68)**	24.35(1.49)
sonar	**19.13(1.80)**	22.60(2.00)	**22.67(3.02)**	22.86(1.65)
vehicle12	**41.32(2.44)**	42.06(3.06)	42.14(0.97)	**40.79(2.29)**
vehicle13	1.81(0.60)	**1.12(0.34)**	1.95(0.63)	**1.58(1.07)**
vehicle14	**4.72(0.79)**	4.86(1.88)	4.96(0.70)	**4.72(1.37)**
vehicle23	2.72(0.88)	**2.34(1.37)**	**3.13(1.03)**	3.17(0.88)
vehicle24	5.00(0.55)	**4.99(0.77)**	6.63(0.98)	**5.43(1.20)**
vehicle34	1.30(0.14)	**1.01(0.43)**	1.63(0.65)	**1.10(0.62)**
wdbc	3.80(0.56)	**3.20(0.44)**	**3.80(0.70)**	3.90(1.52)
wine12	2.77(1.29)	**2.15(1.67)**	**2.92(2.13)**	3.23(1.48)
wine23	4.23(2.85)	**3.88(0.93)**	**3.20(1.59)**	3.56(2.84)
wpbc	23.94(1.78)	**23.11(4.57)**	**23.26(5.05)**	23.86(2.95)
# of Wins	6	14	12	8

TABLE XI

PERFORMANCE COMPARISON AMONG BAGGING PLUS BACING, BACING-0 PLUS BACING-1 AND BAGGING PLUS BACING-0 PLUS BACING-1. C4.5 IS THE BASE CLASSIFIER. VALUES ARE PERCENTAGE TEST ERROR, WITH STANDARD DEVIATION IN PARENTHESES.

Dataset	**Bag+Bac0**	**Bac0+Bac1**	**Bag+Bac0+Bac1**
autompg	**10.51(1.98)**	10.67(1.95)	10.87(2.06)
bupa	28.56(3.76)	28.70(3.12)	**27.84(2.04)**
glass	15.00(2.48)	16.55(1.74)	**14.47(2.96)**
haberman	32.18(3.98)	**29.41(2.71)**	33.38(2.91)
heart	**20.60(1.97)**	20.94(3.01)	20.67(3.09)
hepatitis	**15.53(2.98)**	16.19(4.28)	16.94(1.80)
housing	12.05(1.41)	12.81(1.56)	**11.20(1.56)**
ion	**5.65(1.62)**	6.21(0.60)	5.76(1.49)
pima	**24.76(1.93)**	25.21(1.64)	24.87(1.39)
sonar	22.60(2.00)	21.51(2.82)	**20.98(2.53)**
vehicle12	42.06(3.06)	**40.69(2.11)**	41.44(2.06)
vehicle13	**1.12(0.34)**	2.47(1.07)	1.44(0.50)
vehicle14	4.86(1.88)	4.91(0.64)	**4.63(1.42)**
vehicle23	**2.34(1.37)**	3.23(0.81)	2.40(0.64)
vehicle24	4.99(0.77)	4.99(1.29)	**4.81(0.18)**
vehicle34	**1.01(0.43)**	1.35(0.54)	1.20(0.51)
wdbc	**3.20(0.44)**	4.21(1.00)	3.55(1.14)
wine12	**2.15(1.67)**	2.62(1.40)	2.62(1.40)
wine23	3.88(0.93)	**2.88(1.65)**	3.03(1.00)
wpbc	**23.11(4.57)**	23.63(2.17)	23.61(0.83)
# of Wins	11	3	6

using any cost-sensitive base learner. For non-cost-sensitive base learners like decision trees, resampling techniques can be applied to produce similar improvement.

The cost-updating rule of bacing is somewhat ad hoc. A second cost-updating rule (8), which puts an absolute value on the margin in the original rule (7), is studied. Computational tests reveal that the bacing-1 algorithm based on this rule compares favorably with bagging, as the original algorithm bacing-0 does. However, the mechanisms of these two versions of bacing that drive the improvement are different. Bacing-0 tends to fix any misclassified point by the out-of-bag margin estimate while bacing-1 put more focus on those points close to the separating surface, either correctly or wrongly classified.

The different characteristics of bacing-0 and bacing-1 prompt the idea of hybrid strategies, which sequentially combine different ensemble algorithms. The combination schemes involving bagging, bacing-0 and bacing-1 are extensively studied by computational experiments. The addition of the bagging stage is based on the idea of establishing a good initial margin estimate. Empirical evidence shows that the strategy of sequentially combining bagging and bacing-0 performs the best among all different combination schemes studied. Although combining bacing-0 and bacing-1 is intuitively promising, computational results show otherwise. Adding bacing-1 to the ensemble generation process seems to be unnecessary, if not harmful. Whether revising the cost updating schemes of bacing-1 will change the current conclusion awaits future research work.

We did not provide any termination rules for the bacing algorithm. In a real-world application such a rule would be useful for efficiency purposes. In fact, we have tried to set a threshold on the declining rate of training error as a termination criterion, but it did not work well empirically. Exploring other characteristics of the algorithm or a combination of them to find a good termination rule will be a direction for future work. Another area for exploration is to look for a better way to aggregate the ensemble output other than the majority-vote rule applied here.

REFERENCES

[1] A. Krogh and J. Vedelsby, “Neural network ensembles, cross validation, and active learning,” in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 7. The MIT Press, 1995, pp. 231–238. [Online]. Available: citeseer.nj.nec.com/krogh95neural.html

[2] T. G. Dietterich, “Ensemble methods in machine learning,” *Lecture Notes in Computer Science*, vol. 1857, pp. 1–15, 2000. [Online]. Available: citeseer.nj.nec.com/dietterich00ensemble.html

[3] E. Bauer and R. Kohavi, “An empirical comparison of voting classification algorithms: Bagging, boosting, and variants,” *Machine Learning*, vol. 36, no. 1-2, pp. 105–139, 1999. [Online]. Available: citeseer.ist.psu.edu/bauer99empirical.html

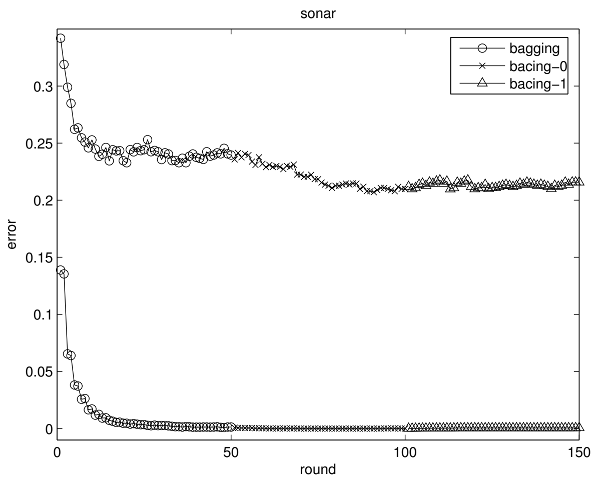
[4] D. Opitz and R. Maclin, “Popular ensemble methods: An empirical study,” *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999. [Online]. Available: citeseer.ist.psu.edu/opitz99popular.html

[5] T. G. Dietterich, “An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization,” *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.

[6] L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996. [Online]. Available: citeseer.nj.nec.com/breiman96bagging.html

[7] Y. Freund and R. E. Schapire, “Experiments with a new boosting algorithm,” in *13th International Conference on Machine Learning*, 1996, pp. 148–156. [Online]. Available: citeseer.nj.nec.com/freund96experiments.html

[8] L. Breiman, “Arcing classifiers,” *Annals of Statistics*, vol. 26, pp. 801–849, 1998.



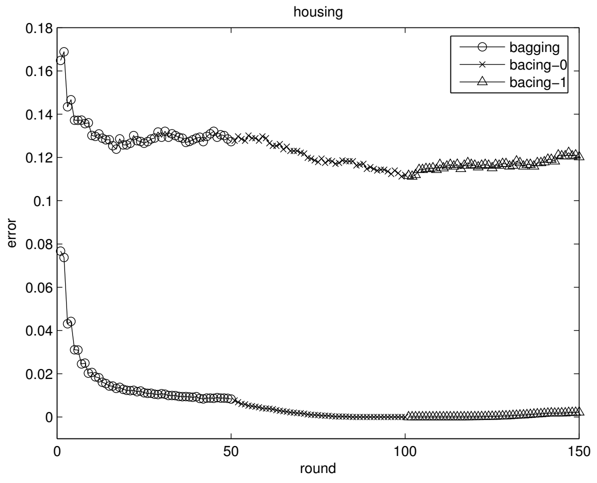


Fig. 9. Error evolution of “Bag+Bac0+Bac1” for sonar and housing data sets. The top line is testing error and the bottom line is training error.

[9] D. H. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, pp. 241–259, 1992. [Online]. Available: citeseer.nj.nec.com/wolpert92stacked.html

[10] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: citeseer.nj.nec.com/breiman01random.html

[11] A. J. Grove and D. Schuurmans, “Boosting in the limit: Maximizing the margin of learned ensembles,” in *AAAI/IAAI*, 1998, pp. 692–699. [Online]. Available: citeseer.ist.psu.edu/grove98boosting.html

[12] G. Ridgeway, “Discussion of additive logistic regression: A statistical view of boosting,” *Annals of Statistics*, vol. 28, pp. 393–400, 2000.

[13] P. S. Bradley, U. M. Fayyad, and O. L. Mangasarian, “Mathematical programming for data mining: Formulations and challenges,” *INFORMS Journal on Computing*, vol. 11(3), pp. 217–238, 1999. [Online]. Available: citeseer.nj.nec.com/bradley98mathematical.html

[14] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[15] C. Blake and C. Merz, “UCI repository of machine learning databases,” 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.html. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[16] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” in *European Conference on Computational Learning Theory*, 1995, pp. 23–37. [Online]. Available: citeseer.nj.nec.com/article/freund95decisiontheoretic.html

[17] S. Abney, R. Schapire, and Y. Singer, “Boosting applied to tagging and PP attachment,” in *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999. [Online]. Available: citeseer.nj.nec.com/abney99boosting.html

[18] R. J. Quinlan, *C4.5: Programs for Machine Learning*. San Manteo, CA: Morgan Kaufman, 1993.

Yi Zhang Yi Zhang received a Ph.D. in Management Sciences from the University of Iowa in 2006, an M. Eng. in High Performance Computation from Singapore-MIT Alliance in 2001, and a B.S. in Mechanical Engineering from Tongji University (China) in 1997. He is currently an applied researcher at Microsoft AdCenter Labs. His research work focuses on mathematical programming, machine learning, data mining and their applications on real-world problems.

W. Nick Street received a Ph.D. in Computer Sciences from the University of Wisconsin-Madison in 1994, an M.S. in Computer Science from DePaul University in 1990, and a B.A. in Math and Computer Science from Drake University in 1985. He is currently an associate professor and Henry B. Tippie Research Fellow in the Management Sciences Department at the University of Iowa, and has joint appointments in the Computer Science Department and the College of Nursing. His research interests are in machine learning and data mining, particularly the use of mathematical optimization in inductive learning techniques. His recent work has focused on ensemble construction methods for large and distributed data sets, rank learning, subset clustering for market segmentation, active learning, and medical decision making. He has received an NSF CAREER award and an NIH INRSA postdoctoral fellowship. He is a member of IEEE, ACM, INFORMS and AAAI.