Bagging with Adaptive Costs

Yi Zhang
yi-zhang-2@uiowa.edu

W. Nick Street
nick-street@uiowa.edu

Department of Management Sciences, The University of Iowa
Iowa City, IA 52242-1000

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, using a linear support vector machine (SVM). 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.

1 Introduction

A typical ensemble is a multi-learner system in which each component learner 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 individual learner. One wants each individual learner to be accurate, and at the same time, these learners should have different inductive biases and thus generalize in distinct ways [8, 6]. In recent years, many approaches have been taken to systematically create a group of learners that perform better when combined, such as bagging [3], boosting [7], arcing [4] and random forests [5]. The idea behind bagging and random forests is to introduce randomness into the procedure of building individual learners, hoping to generate variance among them and thus by average, the bias of the ensemble converges while the variance gets much smaller than that of the base learner. By contrast, boosting and arcing force the learner to focus on difficult training examples and pay less attention to those that the most recently-built learner (boosting) or the existing ensemble (arcing) got right, by applying a "smart" weighting scheme on the training data. These methods are effective at improving generalization performance in a wide variety of domains and for diverse base learners. Wolpert proposed the stacking idea which improves generalization by using a hold-out set to adjust the weights of an existing ensemble so that the error on the hold-out set is minimized [10].

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 [1]. 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.

2 Algorithm

This section describes the derivation of our proposed algorithm, termed Bagging with Adaptive Costs, or bacing (pronounced "baking"). Suppose we have a set of classifiers C for a two-class problem and a training set $\{x_i, y_i\}$. $y_i = 1$ or -1 according to the true class label of each data record. 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 a weighted average on those single classifiers. If all the data records could be correctly classified, the following inequalities would be satisfied:

$$y_i \sum_{k \in C} c_{ik} w_k \geq 0, \tag{1}$$

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

$$m_i = y_i \sum_{k \in C} c_{ik} w_k. \tag{2}$$

A positive m_i implies that the training point is correctly labeled by the current ensemble. The larger the magnitude of a positive m_i, the more tolerance we can allow for the next classifier to make a mistake on this point. A negative m_i corresponds to a misclassification made by the current ensemble, so we want the next classifier to contribute a positive vote margin in order to increase the ensemble's margin on this point.

We use a linear support vector machine [9] as the base learner. An L1-linear SVM model can be described by the linear programming problem

$$\begin{aligned} \min \quad & \sum_{i \in S} \delta_i e_i + \sum_{n=1}^{N} |\alpha_n| \\ \text{s.t.} \quad & y_i \sum_{n=1}^{N} \alpha_n x_{in} \geq 1 - e_i, \\ & e_i \geq 0. \end{aligned} \tag{3}$$

Here, α represents the coefficient vector of the separating hyperplane, e_i is proportional to the distance between the corresponding misclassified point and the plane, N is the dimension of the feature space, and δ_i is the misclassification cost of point i.

Suppose we have an ensemble of linear separators and we wish to build a new separator to insert into the existing ensemble. We may allow the new separator to make mistakes on points that already possess positive margins. But for those points misclassified by the existing ensemble, 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 is more costly. A simple linear cost adjustment scheme is as follows:

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

Here, $\delta_{i,t+1}$ is the adjusted misclassification cost of point i in the next round of training, $|C_t|$ is the number of classifiers in the existing ensemble (equal to t) and $m_{i,t}$ is the current vote margin of point i. The misclassification cost of a point is essentially the same as the weight of a point in arcing or boosting in terms of its role in the linear SVM 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: $\delta_{i,t} = (1 + MC_{i,t})^q$, where $MC_{i,t}$ is the number of misclassifications (wrong votes) of the current ensemble on data point i and q is a positive constant (set to 4 here). The whole training set is used to build each individual separator and the vote 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 derived cost 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% constitute a natural hold-out set, often called the out-of-bag set. We 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

$$m_i = y_i \sum_{k \in \{k | i \notin T_k\}} c_{ik} w_k, \tag{5}$$

The new cost adjustment scheme becomes

$$\delta_{i,t+1} = 1 - \frac{m'_{i,t}}{1 + |C'_{i,t}|} \tag{6}$$

Here, $C'_{i,t}$ is the current set of separators not trained on point i, and $m'_{i,t}$ is the margin for the point obtained by $C'_{i,t}$. 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 individual learners. While it is possible to modify the weight on each separator to minimize the training set error, our experiments indicate that this leads to (often dramatic) overfitting. The weight on each separator is therefore kept equal to one throughout the ensemble construction.

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

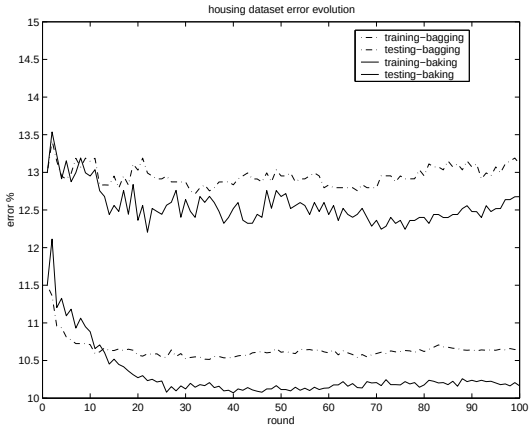

Figure 1. Error evolution of bacing, bagging

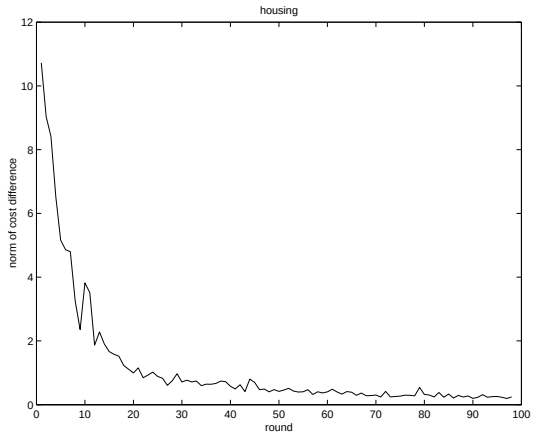

Figure 2. Change of bacing costs

3 Experiments and Results

Bacing was implemented in MATLAB and tested on 14 UCI repository data sets [2]. Some data sets were modified to create binary classification problems. Autompg is labeled by whether the mileage is greater than 25mpg, Housing by whether the house value exceeds $\$25,000$, and Cleveland-heart-disease by the presence or absence of the disease. For multi-class data sets Vehicle and Wine, we separated one class pair each time, resulting in a total of 20 data sets. 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 results are averaged over five ten-fold cross-validations. In comparing bacing with bagging, identical bootstrap samples were obtained for both algorithms in each round. We also compared bacing with arcing-x4, reported to be empirically the best among its variants and comparable with Adaboost [4].

The error evolution of bacing and bagging versus the size of the ensemble for the housing data set is shown in Figure 1; plots for other data sets are similar. Both the training and test errors of bacing are consistently below that of bagging. In the first few rounds, bacing costs can easily fluctuate between their max (2) and min (0) values, so the error fluctuation of bacing is stronger.

Figure 2 illustrates the change of costs over the 100-round run, showing the norm of the difference of the cost vector between neighboring rounds. The costs undergo drastic changes at the start, but as a good cost structure is obtained, the magnitude of the change drops quickly and then remains stable at a low level. Small adjustments are unavoidable because of the bootstrapping effect. This mode of cost adjustment is quite the opposite 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].

Under bacing's simple linear weight adjustment scheme, not only the difficult points will have a large cost in the long run, but also the boundary points, since a boundary point that is not in the training set can easily be misclassified and have its weight tuned up. Thus, the focus on the difficult points is diversified and beneficially pulled to the boundary points. Often, the number of boundary points is much larger than that of the difficult points so that the new classifiers will be influenced more by the boundary points. Since the set of boundary points becomes relatively stable after a few rounds, so will the cost structure. As a result of the convergence of the costs, the error evolution line of bacing shows no obvious sign of overfitting. Often, the error of bacing stabilizes after a certain number of rounds, as in 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.

Table 1 compares the performance of a single linear SVM with 100-round ensembles produced by bacing, bagging and arcing-x4. Vehicleij is the subset with only classes i and j. The same format applies to Wine. Wine13 is omitted because it is linearly separable.

Table 2 summarizes the comparison results. All three ensemble algorithms outperform a single SVM. Though bacing is seldom statistically better than bag-

Table 1. Percentage error of SVM, Bagging, Arcing-x4, Bacing. Best results are bolded.

Data	SVM	Bagging	Arcing	Bacing
Autompg	9.85	9.60	9.29	**8.83**
Bupa	31.07	30.84	31.17	**30.38**
Glass	28.13	27.74	27.12	**27.08**
Haberman	28.10	27.77	25.89	**25.50**
Heart	**16.48**	**16.48**	17.64	17.14
Hepatitis	15.50	**13.47**	14.86	13.94
Housing	12.95	13.11	**11.83**	12.68
Ion	12.42	11.51	**11.06**	11.40
Pima	23.26	23.02	23.30	**22.94**
Sonar	23.69	22.25	**19.36**	20.97
Vehicle12	33.46	33.23	32.78	**32.44**
Vehicle13	**1.16**	1.63	3.35	1.77
Vehicle14	2.97	2.23	3.21	**2.14**
Vehicle23	3.03	2.76	4.24	**2.62**
Vehicle24	2.45	**2.26**	2.88	**2.26**
Vehicle34	2.11	**1.87**	3.07	2.11
WDBC	4.71	4.74	**3.94**	4.50
Wine12	2.77	3.08	**2.00**	2.46
Wine23	4.02	**3.18**	4.73	3.68
WPBC	24.98	23.50	**22.66**	23.05

ging and arcing (2 out of 20), it beats both of them 14 times out of the 20 experiments in terms of mean performance. Further, bagging and arcing significantly improve the base learner only twice while bacing achieves it six times. 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. In addition, the performance of bacing is more stable than that of arcing-x4, which leads to smaller standard deviations (not shown). The distribution of weights may explain why bacing is a more stable algorithm than arcing. Arcing-x4 heavily weights a small number of data points, which may incur big fluctuations in performance among different folds in the cross-validation since a small change in the training set may dramatically change the separating surface. Conversely, the effectively non-zero weights generated by bacing are more spread over the training data, balancing the focus between the boundary points and the "hard" points. This smoother distribution of non-zero weights reduces the risk of overfitting and makes the performance of the algorithm more robust.

Table 2. Summary of accuracy results

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

4 Conclusion

The bacing algorithm combines good features of bagging, arcing and stacking. An effective arcing-like bias correction mechanism is added to the standard bagging method. Bootstrap sampling provides a natural hold-out set that helps the correction mechanism get a fair estimate on the current ensemble's performance. Therefore, it is expected that bacing will be a generally better method than bagging and arcing. The computational experiments supported the conjecture. The algorithm itself is simple to implement, adds very little computational complexity to bagging and can be applied using any cost-sensitive base learner.

References

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

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

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[4] L. Breiman. Arcing classifiers. *Annals of Statistics*, 26:801–849, 1998.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

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

[7] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[8] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.

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

[10] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.