A Comparison of Model Aggregation Methods for Regression

Zafer Barutçuoğlu and Ethem Alpaydın

Department of Computer Engineering, Boğaziçi University, Istanbul, Turkey
zbarutcu@turk.net, alpaydin@boun.edu.tr

Abstract. Combining machine learning models is a means of improving overall accuracy. Various algorithms have been proposed to create aggregate models from other models, and two popular examples for classification are Bagging and AdaBoost. In this paper we examine their adaptation to regression, and benchmark them on synthetic and real-world data. Our experiments reveal that different types of AdaBoost algorithms require different complexities of base models. They outperform Bagging at their best, but Bagging achieves a consistent level of success with all base models, providing a robust alternative.

1 Introduction

Combining multiple instances of the same model type is a means for increasing robustness to variance, reducing the overall sensitivity to different starting parameters and noise. Two well-known algorithms for this purpose are *Bagging* [1] and *AdaBoost* [2,3]. Both have been analyzed for classification in much more detail than regression, possibly due to the wider availability of real-life applications. Adapting classification algorithms to regression raises some issues in this setting. In this paper we compare the Bagging algorithm and several AdaBoost variants for regression.

2 Bagging

The *Bagging* (Bootstrap Aggregating) algorithm [1] uses *bootstrapping* (equiprobable selection with replacement) on the training set to create many varied but overlapping new sets. The base algorithm is used to create a different base model instance for each bootstrap sample, and the ensemble output is the average of all base model outputs for a given input.

The best enhancement by Bagging is when the model instances are very different from each other, since averaging will not have much effect when the outputs are already close. Hence, the most suitable base models for Bagging are *unstable* models, where small changes in the training set can result in large changes in model parameters. Multilayer perceptrons and regression trees are good candidates.

The particular bootstrap sample size being used has an effect on the performance of Bagging, but the optimal ratio of sample to training set size depends on the data. Instead of manually finetuning this ratio per application, we used validation to automate a coarse adjustment that we named *Best-Ratio Bagging*. It removes a fraction of the

O. Kaynak et al. (Eds.): ICANN/ICONIP 2003, LNCS 2714, pp. 76–83, 2003.
© Springer-Verlag Berlin Heidelberg 2003

training set for validation, performs multiple Bagging instances with different ratios on the remaining training examples, and chooses the Bagging model with the lowest error on the validation set as the final model. Although costly, Best-Ratio Bagging is useful for illustrating the best case performance of Bagging with respect to sample size.

Bagging takes a simple average of outputs, but the evaluation part of AdaBoost can be adopted and a weighted median may be used instead. The weights (confidences) can be calculated as in AdaBoost, using average loss with respect to a loss function. We implemented this variant using linear loss. See Section 3.2 for the computation of confidence and weighted median.

To produce similar but perturbed subsets from one training set, *K-fold cross-validation* is an alternative to bootstrapping. The training set $\mathcal{X}$ is randomly partitioned into K sets $\mathcal{X}_i$ of equal size, and each base model is trained on $\mathcal{X} - \mathcal{X}_i$. We called this algorithm *Cross-Validation Aggregating* (CVA). Evaluation is by averaging outputs, as in Bagging. As opposed to bootstrapping, cross-validation is guaranteed to use all training examples exactly once in exactly $K - 1$ subsets. For small K, this leads to more efficient use of data than bootstrapping. However as K increases, we get increasingly similar subsets, which should decrease the positive effect of combining.

3 The AdaBoost Approach

Since individual bootstrap samples are selected independently, the collective success of the models they produce is through mere redundancy. The *boosting* approach uses the base models in sequential collaboration, where each new model concentrates more on the examples where the previous models had high error. Different ways of realizing this dynamic focus lead to different algorithms. *AdaBoost* (*Ada*ptive *Boost*ing) [2,3] is an efficient and popular implementation of the boosting principle, applied and analyzed with much deeper interest for classification than regression. Since the latter is a more general problem, the basic concept of AdaBoost can be generalized in more than one way for regression.

3.1 AdaBoost.R

The originally proposed AdaBoost for regression *AdaBoost.R* is based on decomposing regression into infinitely many classification tasks [2]. This construction does allow an implementation, but it involves keeping track of a different updatable and integrable loss function for each example. Furthermore, the base learner must be able to accommodate such dynamic loss functions per example. This *dynamic-loss* approach was also used by Ridgeway *et al.* [4], but their experiments using naive Bayes base learners yielded no significant justification to afford a per-example redefinable loss, seriously constraining the choice of base learners if not time complexity.

3.2 Distribution-Based Algorithms

Drucker's AdaBoost. Drucker's AdaBoost for regression [5] is an *ad hoc* adaption of the classification AdaBoost. Despite the lack of a rigorous derivation, it uses scalar selection probabilities, unlike AdaBoost.R. It works much like classification AdaBoost, favoring examples with high error. The ensemble output is the weighted median of the base model outputs, weighted by the models' training confidences.

The weighted median can be computed by first sorting the outputs in order of magnitude, and then summing their weights until the sum exceeds half the weight total. If the weights were integers, this would be analogous to duplicating the outputs by their weights and taking the regular median.

At each step i, the algorithm minimizes the error function (in rearranged notation)

$$J_i = \sum_{t=1}^{N} \exp(-c_i) \exp\left(c_i L_i^t\right)$$

by minimizing per-example losses L_i^t. c_i is a measure of confidence over all examples, also used as the *combination coefficient* during evaluation. Drucker's AdaBoost chooses $c_i = \ln\left[(1 - \overline{L}_i)/\overline{L}_i\right]$ using $\overline{L}_i = \sum_{t=1}^{N} L_i^t p^t$ to minimize error, but this appears to be an unjustified adoption of the analytical result for classification. In the experiments we used linear loss (absolute difference) $L = |y - r|/D$ in Drucker.AD and square loss $L_S = |y - r|^2/D^2$ in Drucker.S where $D = \sup_t |y^t - r^t|$.

Zemel & Pitassi's Algorithm. Zemel & Pitassi [6] provide an algorithm similar to Drucker's, but with alternative mathematical particulars. Here the error function is

$$J_i = \sum_{t=1}^{N} c_i^{-1/2} \exp\left(c_i |y_i^t - r^t|^2\right)$$

where the loss function is squared error, and not scaled to $[0, 1]$.

Although the multiplier is now $c_i^{-1/2}$, replacing Drucker's $\exp(-c_i)$, with $0 < c_i \leq 1$ they behave similarly except near zero. Notably Zemel & Pitassi acknowledge that here c_i cannot be analytically determined, and simple line search is used. Finally, this algorithm uses weighted mean instead of weighted median to combine outputs.

We implemented this algorithm as Zemel-Pitassi.S and Zemel-Pitassi.AD, using the original square loss and linear loss respectively. In Zemel-Pitassi.AD we replaced weighted mean by weighted median to match the loss function.

3.3 Relabeling Algorithms

Another group of algorithms [7,8,9], although from different viewpoints, all aim to minimize *residual* error. In these algorithms each new base model learns artificial labels formed using the per-example training errors (*residues*) of the current combined model. After training each model i the residues are updated by subtracting the prediction y_i^t of the new model weighted by its coefficient c_i. Due to the subtractive training, combination is additive, using a weighted sum.

The LS_Boost **Algorithm.** The algorithm LS_Boost is from Friedman's gradient-based boosting strategy [7], using square loss $L = (y - r)^2/2$ where r is the actual training label and y is the current cumulative output $y_i = c_0 + \sum_{j=1}^{i-1} c_j h_j + c_i h_i = y_{i-1} + c_i h_i$. The new training labels $\hat{r}$ should be set to the direction that minimizes the loss, which is the negative gradient with respect to y evaluated at y_{i-1}. Then $\hat{r} = [-\partial L/\partial y]_{y=y_{i-1}} = r - y_{i-1}$ which is the current residual error. Substituting into the loss, we get the training error

$$E = \sum_{t=1}^{N} [c_i h_i^t - \hat{r}^t]$$

where $\hat{r}^t$ are the current residual labels. The combination coefficients c_i are determined by solving $\partial E/\partial c_i = 0$.

Duffy & Helmbold [8] give an algorithm SquareLev.R which is identical in effect. SquareLev.C, a variant of SquareLev.R, is more interesting in that while also based on residual error, it still uses probabilities. The base learner is fed not the residues $\hat{r}$, but their signs $sign(\hat{r}) \in \{-1, +1\}$, while the distribution weight of each example is made proportional to $|\hat{r}|$, so each example is still "emphasized" in proportion to its residual error. At the cost of handling probabilities, SquareLev.C allows using binary classifiers.

The LAD_Boost **Algorithm.** The LAD_Boost algorithm from [7] is derived from the same gradient-based framework as LS_Boost, but using linear loss (absolute deviation). The gradient of linear loss leads to the sign of the residue, so the base models are trained on $\{-1, +1\}$ labels, which also allows using classifiers. Here the derivation of c_i yields another weighted median computation. See [7] for details.

4 Experiment Design

We tested the algorithms using J-leaf regression trees with constant leaf labels. The learner subdivides the leaf having the greatest total squared deviation from the mean, until a specified node count J is reached or all leaves have a single training element. J is used to control base model complexity. Values of $\{2, 5, 10, 15, 20\}$ were used for the number of base trees to combine.

Bagging used a fixed 50% sample size ratio, while Best-Ratio Bagging compared the ratios $10\%, 20\%, \ldots, 90\%$ of the remaining examples for sample size using 50% of the examples for validation. All experiments were repeated ten times, using 5×2-fold cross-validation to partition the datasets. The algorithms were compared by the *5×2-fold cross-validated F test* [10] at 95% confidence. We used the datasets in Table 1 for our experiments. All of them have one-dimensional continuous output for regression.

`syndata` was synthetically generated for observing the algorithms visually. It has unidimensional input, and on an output range of $[-15, +15]$ it has Gaussian noise of zero mean and unit variance. `abalone`, `boston` and `calif1000` are from [11]. `prostate` and `birth` are from [12]. `votes` and `kin8` datasets are from the StatLib archive of Carnegie Mellon University. For each dataset, we repeated the experiments using 5×2-fold cross-validation. The error bars in the figures indicate one standard deviation above and below the mean error of the ten runs.

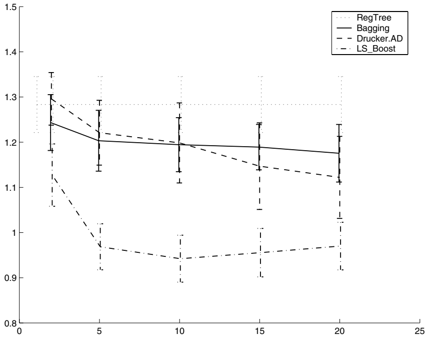

Fig. 1. 5-leaf `syndata` errors

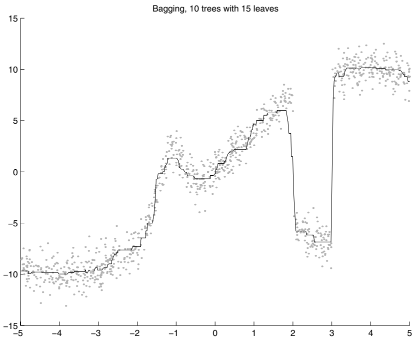

Fig. 4. 15-leaf Bagging

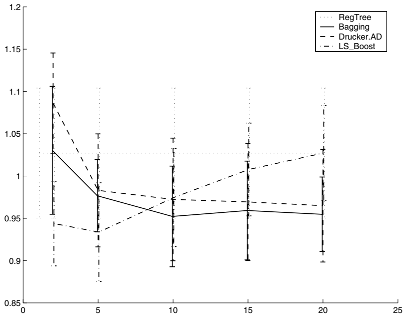

Fig. 2. 10-leaf `syndata` errors

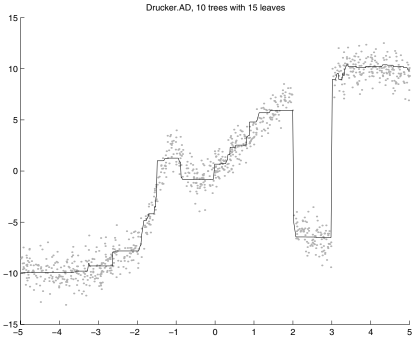

Fig. 5. 15-leaf Drucker.AD

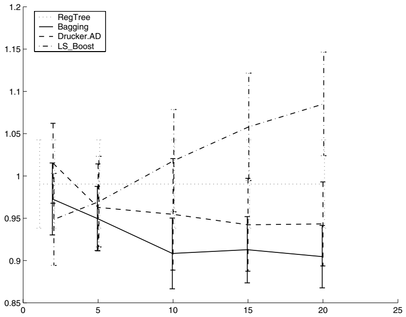

Fig. 3. 15-leaf `syndata` errors

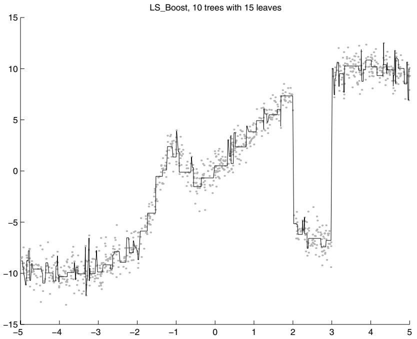

Fig. 6. 15-leaf LS_Boost

5 Simulation Results

Figures 1 to 3 show the test errors of BAGGING, LS_BOOST and DRUCKER.AD on `syndata` as the number of trees changes. The unaggregated base algorithm REGTREE is also included, plotted as constant. These figures illustrate typical behaviors, also observed numerically on other datasets. Figures 4, 5 and 6 show example outputs on `syndata` using 15-leaf regression trees as base models.

The Bagging methods used both small and large trees with consistent success, although they took a large number of large trees to catch up with the relabeling AdaBoost algorithms. CVA was slightly better than Bagging algorithms for very few base models, and fell behind quickly thereafter as the cross-validated training sets became increasingly similar. W-BAGGING never significantly decreased test error beyond Bagging, sometimes even increasing it. Considering that bootstrap samples are selected uniformly, it is not surprising that "confidence" values derived from accidental differences are bound to disrupt Bagging rather than enhance it. Compared to a fixed 50% ratio of sample size with BAGGING, BR-BAGGING did not show significant improvement despite the nine-fold execution time.

ADABOOST.R, despite its unwieldy time complexity, was not able to improve the base model beyond the statistical significance threshold on any of the datasets.

Drucker's and Zemel & Pitassi's algorithms did not perform well using small trees on large datasets. They even increased training error, indicating that this is not due to overfitting, but the base models were too coarse to be useful to them.

LAD_BOOST and LS_BOOST started overfitting at much smaller trees than the base algorithm alone, because their modification of labels reduces the complexity of data. This is especially true of LAD_BOOST which greatly simplifies the problem for the base learners by discretizing pseudo-targets to binary. The rapid overfitting can be observed in Figure 6.

Over the tree sizes used and model counts up to ten, the best instances are reported in Table 2 as average errors and standard deviations over ten runs. The results are compared using the 5$\times$2-fold cross-validated F-test with 95% confidence on each dataset. Some illustrative pairs of algorithms are shown in Table 3, where the column "$>$" denotes on how many datasets the left-hand algorithm was significantly superior.

6 Conclusion

Bagging proved to be very robust with respect to base model complexity. It was able to reduce test error successfully whether the underlying base models were overfit or underfit. Our variants of Bagging failed to offer any significant improvement over the original BAGGING algorithm, though we did thus verify the integrity of Bagging.

ADABOOST.R as it was originally proposed did not show any improvement over the unaggregated base model, let alone BAGGING, despite its special base model requirements for dynamic loss and prohibitive time complexity.

The distribution-based AdaBoost algorithms needed sufficiently complex base models. Otherwise they failed to reduce even the training error. DRUCKER and ZEMEL-PITASSI

Table 1. Properties of the datasets used

	inputs	size		inputs	size		inputs	size
syndata	1	1,000	prostate	7	376	kin8fh	8	8,192
boston	12	506	birth	5	488	kin8nm	8	8,192
calif1000	8	1,000	votes	6	3,107	kin8nh	8	8,192
abalone	10	4,177	kin8fm	8	8,192			

Table 2. Base model and Bagging results

	RegTree	Bagging	BR-Bagging	W-Bagging	CVA	AdaBoost.R
	avg ± std	avg ± std	avg ± std	avg ± std	avg ± std	avg ± std
syndata	.943± .049	.890±.043	.888± .034	.887±.035	.903±.044	.919± .084
boston	.350± .023	.295±.016	.294± .026	.296±.019	.308±.028	.307± .028
calif1000	.506± .028	.445±.016	.444± .017	.436±.019	.452±.018	.464± .022
votes	.493± .013	.444±.006	.445± .005	.446±.004	.458±.007	
prostate	.668± .072	.635±.042	.642± .035	.605±.044	.627±.053	
birth	.812± .055	.780±.033	.777± .029	.781±.031	.785±.030	
abalone	.545± .005	.521±.018	.513± .011	.483±.008	.523±.015	
kin8fm	.439± .003	.316±.006	.314± .008	.330±.007	.358±.009	
kin8fh	.553± .005	.457±.005	.456± .007	.464±.005	.488±.008	
kin8nm	.595± .012	.523±.004	.523± .003	.511±.007	.546±.006	
kin8nh	.657± .011	.601±.007	.600± .008	.597±.007	.615±.009	
	Drucker.AD	Drucker.S	Z&P.AD	Z&P.S	LAD_Boost	LS_Boost
	avg ± std	avg ± std	avg ± std	avg ± std	avg ± std	avg ± std
syndata	.917± .055	.921±.061	.910± .039	.900±.045	.978±.050	.934± .058
boston	.276± .019	.297±.014	.280± .018	.286±.023	.346±.025	.335± .029
calif1000	.429± .015	.457±.016	.425± .020	.447±.019	.455±.021	.468± .016
votes	.443± .006	.455±.005	.447± .004	.449±.006	.472±.012	.481± .014
prostate	.650± .051	.678±.039	.650± .075	.631±.049	.600±.023	.678± .053
birth	.791± .026	.792±.034	.790± .026	.790±.026	.793±.028	.783± .028
abalone	.514± .015	.544±.029	.497± .011	.544±.039	.497±.005	.520± .008
kin8fm	.288± .005	.279±.006	.294± .005	.295±.007	.316±.012	.296± .011
kin8fh	.444± .006	.438±.004	.446± .004	.446±.005	.481±.005	.485± .004
kin8nm	.502± .004	.523±.007	.510± .005	.523±.004	.531±.009	.528± .012
kin8nh	.597± .011	.600±.012	.601± .009	.604±.012	.627±.011	.624± .010

Table 3. Significant superiority over 11 datasets

		>	=	<			>	=	<
Drucker.AD	ZP.AD	1	10	0	Drucker.S	RegTree	5	6	0
Drucker.S	ZP.S	2	8	1	W-Bagging	Bagging	0	9	2
Drucker.AD	LAD_Boost	4	7	0	BR-Bagging	Bagging	0	11	0
Drucker.S	LS_Boost	2	9	0	LAD_Boost	RegTree	6	5	0
Bagging	RegTree	7	4	0	LS_Boost	RegTree	5	6	0
Bagging	CVA	3	8	0	LAD_Boost	Bagging	0	9	2
Drucker.AD	RegTree	7	4	0	LS_Boost	Bagging	0	7	4

were almost always equal in performance. In that case DRUCKER may be slightly more preferable, considering the inconvenient line search in ZEMEL-PITASSI.

The relabeling AdaBoost algorithms, in contrast, called for very simple models that would normally underfit. With complex base models their performance deteriorated rapidly as they started overfitting the data.

In selecting an aggregation algorithm for a regression task, if the base models are inherently simple or their complexity can be adjusted by some validation method, the relabeling algorithms should be considered, since they can provide the best accuracy using the fewest base models. If the models cannot be prevented from overfitting, one of the distribution-based AdaBoost algorithms can be used. The choice of loss function depends on the data at hand. If one algorithm must be selected to handle both simple and complex base models, Bagging is a safe bet.

Acknowledgments. This work has been supported by Boğaziçi University Scientific Research Project 02A104D and the Turkish Academy of Sciences, in the framework of the Young Scientist Award Program (EA-TÜBA-GEBIP/2001-1-1).

References

1. Breiman, L., “Bagging Predictors”, *Machine Learning*, Vol. 24, No. 2, pp. 123–140, 1996.
2. Freund, Y. and R. E. Schapire, “A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting”, *European Conf. on Computational Learning Theory*, pp. 23–37, 1995.
3. Freund, Y. and R. E. Schapire, “Experiments with a New Boosting Algorithm”, *International Conf. on Machine Learning*, pp. 148–156, 1996.
4. Ridgeway, G., D. Madigan and T. Richardson, “Boosting methodology for regression problems”, *Proc. of Artificial Intelligence and Statistics*, pp. 152–161, 1999.
5. Drucker, H., “Improving regressors using boosting techniques”, *Proc. 14th International Conf. on Machine Learning*, pp. 107–115, Morgan Kaufmann, San Francisco, CA, 1997.
6. Zemel, R. S. and T. Pitassi, “A Gradient-Based Boosting Algorithm for Regression Problems”, *Adv. in Neural Information Processing Systems*, Vol. 13, 2001.
7. Friedman, J. H., *Greedy Function Approximation: a Gradient Boosting Machine*, Tech. Rep. 7, Stanford University, Dept. of Statistics, 1999.
8. Duffy, N. and D. Helmbold, “Leveraging for Regression”, *Proc. 13th Annual Conf. on Computational Learning Theory*, pp. 208–219, Morgan Kaufmann, San Francisco, CA, 2000.
9. Rätsch, G., M. Warmuth, S. Mika, T. Onoda, S. Lemm and K.-R. Müller, “Barrier Boosting”, *Proc. 13th Annual Conference on Computational Learning Theory*, 2000.
10. Alpaydın, E., “Combined 5×2cv F Test for Comparing Supervised Classification Learning Algorithms”, *Neural Computation*, Vol. 11, No. 8, pp. 1885–1992, 1999.
11. Blake, C. and P. M. Murphy, “UCI Repository of Machine Learning Databases”, `http://www.ics.uci.edu/~mlearn/MLRepository.html`.
12. Hosmer, D. and S. Lemeshow, *Applied Logistic Regression*, John Wiley & Sons Inc., 2nd edn., 2000.