AdaBoost.RT: A boosting algorithm for regression problems

Conference Paper · August 2004

DOI: 10.1109/IJCNN.2004.1380102 · Source: IEEE Xplore

Durga Shrestha
The Commonwealth Scientific and Industrial Research Organisation
60 PUBLICATIONS **1,730** CITATIONS

AdaBoost.RT: a Boosting Algorithm for Regression Problems

D. P. Solomatine
UNESCO-IHE Institute for Water Education
P.O. Box 3015, 2601 DA Delft,
The Netherlands
E-mail: d.solomatine@unesco-ihe.org

D. L. Shrestha
UNESCO-IHE Institute for Water Education
P.O. Box 3015, 2601 DA Delft,
The Netherlands
E-mail: shres15@unesco-ihe.org

Abstract– **A boosting algorithm, *AdaBoost.RT* for regression problems is proposed. Its idea is in filtering out the examples with the relative estimation error that is higher than the pre-set threshold value, and then following the *AdaBoost* procedure. Thus it requires to select the sub-optimal value of relative error threshold to demarcate predictions from the predictor as correct or incorrect. Some experimental results using M5 model tree as a weak learning machine for benchmark data sets and for hydrological modeling are reported, and compared to other boosting methods, bagging and Artificial Neural Networks, and to a single M5 model tree. *AdaBoost.RT* is proved to perform better on most of the considered data sets.**

I. INTRODUCTION

Committee machines (mixture of experts) currently receive a lot of attention. In a committee machine an ensemble of predictors is generated by means of a learning process; the overall predictions of the committee machine is the combination of the predictions of the individual committee members [22]. Two popular committee machines method are bagging [3; 4] and boosting [20; 9; 10]. They combine the outputs from different predictors to improve the accuracy of prediction. Several studies of boosting and bagging in classification have demonstrated that these techniques are generally more accurate than the individual classifiers.

Boosting can be used to reduce the error of any "weak" learning machine that consistently generates classifiers which need only be a little bit better than random guessing [9]. Boosting works by repeatedly running a given weak learning machine on different distribution of training data and combining their outputs. At each iteration the distributions of training data depend upon the performance of the machine in the previous iteration. The method to calculate the distribution of the training data and combining the predictions from each machine is different for various boosting methods.

There are different versions of boosting algorithm for classification problems. The original boosting approach is *boosting by filtering* and is described by Schapire [20]. It requires a large number of training data, which is not feasible in many cases. This limitation can be overcome by using another boosting algorithm, *AdaBoost* [9; 10] in several versions. In *boosting by sub-sampling* the fixed training size and training examples are used and they are resampled according to a given probability distribution during training. In *boosting by reweighting* all the training examples are used to train the weak learning machine with weights assigned to each example. So this technique is useful when the weak learning machine can handle the weighted examples. Originally the boosting algorithm was developed for binary classification problems. Freund and Schapire [10] extended *AdaBoost* to a multi-class case, which they called *AdaBoost.M1* and *AdaBoost.M2*. Many researchers have been performing experiments with boosting algorithms for classification problems (for example [17]; [15]; [7]).

In order to solve regression problems, Freund and Schapire [10] extended *AdaBoost.M2* and called it *AdaBoost.R.* It solves regression problems by reducing them to classification ones. Although experimental work shows that the *AdaBoost.R* can be effective by projecting the regression data into classification data set, it suffers from two drawbacks. First, it expands each example in the regression sample into many classification examples and the latter grows linearly in the number of boosted iterations. Second, the loss function (to be defined later) changes from iteration to iteration and even differs between examples on the same iteration. However, in the framework of *AdaBoost.R*, Ridgeway et al. [19] did some experiments by projecting regression problems into classification problems on a data set of infinite size. Breiman [5] proposed *arc-gv* (arcing game value) algorithm for regression problems. Drucker [6] developed *AdaBoost.R2* algorithm, which is ad hoc modification of *AdaBoost.R*. He performed some experiments with *AdaBoost.R2* for regression problems and obtained some promising results. Avnimelech and Intrator [1] also extended the boosting algorithm to regression problems by introducing the notion of weak and strong learning and an appropriate equivalence theorem between them. They introduced the so-called *big errors* with respect to threshold, which has to be chosen prior. They constructed triplet of weak learners and combined them to reduce the big error rate using the median of the outputs of the triplet learners.

0-7803-8359-1/04/$20.00 ©2004 IEEE

Recently many researchers (for example [11]; [12]; [8]; [25]; [18]) have viewed boosting as a gradient machine that optimizes some sort of a loss function. Friedman et al. [11] explained the *AdaBoost* algorithm from statistical perspective. They showed that *AdaBoost* algorithm is a Newton method for optimizing a particular exponential loss function. Although all these methods involve diverse objectives and optimization approaches, they are similar except for the one considered by Zemel and Pitassi [25] in the following sense: the new regression models (hypotheses) are formed by changing both the distribution of the training sample and modifying the target values. Zemel and Pitassi [25] derived a gradient based boosting algorithm, which forms new hypotheses by modifying only the distribution of the training sample. This approach deviates from the standard boosting in that distribution of the sample is not used to control the generation of the hypotheses and each hypothesis is not trained to learn the same function. It should be mentioned however, that no single boosting algorithm has been found that can be declared a clear winner in regression problems.

In an attempt to adopt the idea of boosting for regression problems, the authors took the *AdaBoost.M1* as the basis. The idea is to use some sort of a margin to differentiate correct and incorrect predictions for regression as if in the classification. Typically in the regression, it is not possible to predict the output exactly like in classification. Some discrepancy between the predicted values and the observed ones is typically inevitable. The quantitative measure of this discrepancy allows us to distinguish whether the prediction is correct (acceptable) or not. Once we define some procedures for margin of discrepancy it is straightforward to follow the *AdaBoost.M1* procedure. This was the motivation for the new boosting algorithm *AdaBoost.RT* for regression problems.

In this paper a boosting algorithm *AdaBoost.RT* for regression problems is proposed. The main idea of it is to introduce a constant ϕ as an relative error threshold value used to demarcate predictions as correct or incorrect. Then error rate is calculated by counting number of correct and incorrect predictions. Consequently, weight updating parameter is computed and the distribution of the training set is updated using the weight updating parameter.

The main difference of *AdaBoost.RT* from most of other methods is in computing loss functions using relative error rather than absolute error. This makes it possible to give enough attention to the examples with low values. Moreover, weight updating parameter is computed differently to give more emphasis on harder example when error rate is very low. Another key difference is that the algorithm do not have to stop when error rate becomes greater than 0.5 as it happens in some other algorithms. This makes it possible to run a user-defined number of iterations and in most cases the performance is improved as compared to the early termination of iterations. Lastly, combining the outputs from individual machines is computed by weighted average while most of the methods consider median and former approach appeared to give advantages.

The main advantage of *AdaBoost.RT* is that it is very simple and direct implementation of *AdaBoost.M1* algorithm for regression. As compared to *AdaBoost.R2*, it appeared to be better in the presence of noise. Although it introduces new parameter that has to be selected, there is no significant computational burden even in case it has to be calibrated.

M5 model tree (MT) of Quinlan [16; 24] is used as a weak learning machine. The experiment is performed with six different data sets and comparison is made to bagging, *AdaBoost.R2*, and Artificial Neural Networks (ANNs).

II. THE BOOSTING ALGORITHMS

The boosting algorithms for classification and regression problems are presented simultaneously so that it is easy to compare the algorithms. The following algorithmic layout represents three algorithms: *AdaBoost.M1* [10] used for classification problems, and two distinct regression algorithms *AdaBoost.R2* [6], and the new ***AdaBoost.RT*** (in bold).

```
1. Input:
♦ Sequence of $m$ examples $(x_1, y_1),...,(x_m, y_m)$ where labels
  $y \in Y = \{1,...,k\}$ for classification and $y \in R$ for regression problems
♦ Weak learning algorithm Weak Learner
♦ Integer $T$ specifying number of iterations (machines)
♦ Threshold $\phi$ for demarcating correct and incorrect predictions
  for AdaBoost.RT
2. Initialize:
♦ Machine number or iteration $t = 1$
♦ Distribution $D_t(i) = 1/m$ for all $i$
♦ Error rate $\varepsilon_t$ or average loss function $\overline{L_t} = 0$
3. Iterate:
  while error rate $\varepsilon_t < 0.5$ for classification, or average loss
  function $\overline{L_t} < 0.5$ for AdaBoost.R2, or $t \le T$ for AdaBoost.RT
♦ Call Weak Learner, providing it with distribution $D_t$
♦ Get back a hypothesis $h_t : X \to Y$ for classification
♦ Build the regression model: $f_t(x) \to y$ for regression problems
♦ For classification: Calculate the error rate of $h_t$ :
  $\varepsilon_t = \sum_{i : h_t(x_i) \neq y_i} D_t(i)$
♦ For AdaBoost.R2:
    • Calculate the loss for each training example as:
      $l_t(i) = |f_t(x_i) - y_i|$
    • Calculate the loss function $L_t(i)$ for each training example
      using three different functional forms as:
      Linear: $L_t(i) = l_t(i)/Den_t$ ; Square law: $L_t(i) = (l_t(i)/Den_t)^2$
      Exponential: $L_t(i) = 1 - \exp(-l_t(i)/Den_t)$
      where $Den_t = \max_{i=1...m} (l_t(i))$
    • Calculate an average loss as: $\overline{L_t} = \sum_{i=1}^{m} L_t(i) D_t(i)$
```

- For ***AdaBoost.RT***: calculate the error rate of $f_t(x)$:

$$\varepsilon_t = \sum_{i:\left|\frac{f_t(x_i)-y_i}{y_i}\right|>\phi} D_t(i)$$

- Set $\beta_t = \varepsilon_t/(1-\varepsilon_t)$ for classification, or $\beta_t = \overline{L_t}/(1-\overline{L_t})$ for *AdaBoost.R2*, or $\beta_t = \varepsilon_t{}^2$ for ***AdaBoost.RT***
- Update distribution D_t as

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{Bmatrix} \beta_t \text{ if } h_t(x_i) = y_i \\ 1 \quad \text{otherwise} \end{Bmatrix} \text{ for classification}$$

$$D_{t+1}(i) = \frac{D_t(i)\beta_t^{(1-L_t(i))}}{Z_t} \text{ for } \textit{AdaBoost.R2}$$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{Bmatrix} \beta_t \text{ if } \left|\frac{f_t(x_i)-y_i}{y_i}\right| <= \phi \\ 1 \quad \text{otherwise} \end{Bmatrix} \text{ for } \textbf{\textit{AdaBoost.RT}}$$

where Z_t is a normalization factor chosen such that D_{t+1} will be a distribution

- Set $t = t + 1$

4. Output the final hypothesis:

$$h_{fin}(x) = \arg\max \sum_{t:h_t(x)=y} \log(1/\beta_t) \text{ for classification}$$

$$f_{fin}(x) = \inf\left[y \in Y: \sum_{t:f_t(x)\le y} \log(1/\beta_t) \ge \frac{1}{2}\sum_t \log(1/\beta_t) \right] \text{ for } \textit{AdaBoost.R2}$$

$$f_{fin}(x) = \sum_t \log(1/\beta_t) f_t(x) / \sum_t \log(1/\beta_t) \text{ for } \textbf{\textit{AdaBoost.RT}}$$

In the beginning the weak learning machine is supplied with training set of m examples with the uniform distribution of weights so that each example has an equal chance to be selected in the first training set. The performance of the machine is evaluated by computing the classification or prediction error. For classification problem, error rate ε_t is calculated simply by counting the numbers of correct and incorrect classifications by this machine. In case of *AdaBoost.R2*, the so-called loss function L_t is introduced to compute the performance of the machine. Average loss $\overline{L_t}$ for the machine in the ensemble could be computed by either one of the three-candidate loss functions as presented in the above layout.

On the other hand, in the presented *AdaBoost.RT* algorithms, ε_t is computed using the notion of a pre-set *threshold* ϕ, which is used to demarcate prediction error as correct or incorrect. If the absolute relative error (ARE) for any particular example is greater than ϕ, the predicted value for this example is considered to be incorrect, otherwise it is correct. The numbers of correct and incorrect predictions are counted to calculate ε_t. Indeed, the machine attempts to find the $f_t(x)$ with small ε_t, which is possible only by reducing the ARE of the each example.

Knowing ε_t or $\overline{L_t}$, it is possible to compute weight updating parameter $\beta_t \in [0,1]$ as a function of ε_t or $\overline{L_t}$. Indeed, β_t is a measure of confidence in the predictor. If ε_t or $\overline{L_t}$ is low, then β_t is also low and low β_t means high confidence in the predictor. Intuitively, the value of $\log(1/\beta_t)$ (weight of the machine) becomes higher and consequently higher weight is given to this machine when combining the output from individual machines.

To compute the distribution for the next machine, we multiply the weight of each example by β_t if the previous machine classifies or predicts this example correctly (this reduces the weight of the example), and otherwise the weight remains unchanged. Thus it seems that the regression problem in *AdaBoost.RT* is projected into the binary classification problems while updating the weights of the examples. In *AdaBoost.R2*, however, all the weights are updated according to the exponential loss functions of β_t. The weights are then normalized to make their set a distribution.

The process is repeated until a preset number of machines are constructed or $\varepsilon_t < 0.5$ for classification or $\overline{L_t} < 0.5$ for *AdaBoost.R2*. It should be noted that for *AdaBoost.RT* iterations do not stop when ε_t is higher than 0.5. Finally, the output from different machines will be combined to produce single classification or prediction. For classification problem the final output will be the weighted vote of the weak machines' results. For *AdaBoost.R2* the final output is the weighted median, and for *AdaBoost.RT* it is the weighted average.

It is seen from the above algorithmic layout that in *AdaBoost.RT*, β_t is computed differently. As discussed above, when ε_t (in *AdaBoost.M1*) or $\overline{L_t}$ (in *AdaBoost.R2*) is greater than 0.5, iteration will be terminated. Otherwise, the value of β_t will exceed unity and consequently, the value of $\log(1/\beta_t)$ becomes negative. Breiman [4] describes a method by resetting all the weights to be equal and restart if either ε_t is not less than 0.5 or ε_t becomes 0. Following the revision described by Breiman [4], Opitz and Maclin [15] used a very small positive value (0.001) rather than using a negative or 0 weight factor when ε_t is larger than 0.5. They reported slightly better results using this approach. So for *AdaBoost.RT* we reformulated the expression for β_t to overcome the value of $\log(1/\beta_t)$ becoming negative even ε_t is greater than 0.5. Furthermore, it seems plausible that when ε_t is very low (close to 0.1 or even less), then relatively more weights are given to the harder examples (as β_t deviates further from 1) and the machine concentrates on these hard examples. This expression proposed in the algorithm is quite simple and it works well for the considered data sets, however there would be different functional relationship for β_t. Unfortunately, we do not have the proof of the algorithm convergence.

In contrast to the *AdaBoost.R2* algorithm, *AdaBoost.RT* needs to optimally select ϕ. The experiments with the *AdaBoost.RT* have shown that the performance of the committee machine is sensitive to ϕ. If ϕ is too low, then it is generally very difficult to get a sufficient number of correctly predicted examples. On the other hand if ϕ is very high, it is possible that there are only few "hard" examples, which are often outliers, and intuitively these examples will get boosted. This will affect the performance of the committee machine seriously and may make it unstable. In order to estimate the preliminary values of ϕ, it is required to find out statistics on prediction error (possibly ARE) for the single machine before committing the boosting.

Note that the test used by *AdaBoost.RT* to demarcate incorrectly predicted examples includes the division by y_i This means that in case of small values of y_i the data transformation would be advisable. The idea behind using the relative error in *AdaBoost.RT* is to give enough attention to the examples with low values, which in case of using absolute error may be ignored. Such examples would be hopefully given higher weight and processed by a specialized machine in the ensemble.

III. Experimental Results

A. Data Sets

In order to evaluate the performance of boosting, two groups of data sets were selected. The first one constituted a number of data sets from the UCI repository [2]. These data sets can be also used as benchmarks for comparing results with the previous research. Another group of data sets comes from previous research [21] using ANNs and MTs to predict flows in the Sieve catchments in Italy. Prediction of river flows Q_{t+i} several hours ahead (i=3 or 6) is based on using the previous values of flow ($Q_{t-\tau}$) and previous values of rainfall ($RE_{t-\tau}$), τ being between 0 and 3 hours. The sought regression models were based on 2154 examples and were formulated as follows: *SieveQ3* model where $Q_{t+3} = f(RE_t, RE_{t-1}, RE_{t-2}, RE_{t-3}, Q_t, Q_{t-1})$, and *SieveQ6* model where $Q_{t+6} = f(RE_t, Q_t)$.

Table 1: Data sets summary

Data Set	Training	CV	Test	Attributes	
				Continuous	Discrete
Housing	401	80	25	13	1
SieveQ3	1554	300	300	7	0
SieveQ6	1554	300	300	3	0
Auto-Mpg	262	-	136	5	3
CPU	137	-	72	7	1
Friedman#1	990	-	510	6	0

For *SieveQ3*, *SieveQ6* and Housing, the data set was randomly divided into training, cross validation (CV) (sometimes referred to as simply validation) and test subsets. The division of data for Housing is kept as in the previous research work by Drucker [6]. The division of data for *SieveQ3* and *SieveQ6* is based on the previous research done by Solomatine and Dulal [21]. Due to time constraint, the rest of the data sets was randomly divided into training (66%) and test set (34%) only. Table 1 summarizes the characteristics of the data sets.

B. Methods

The data sets for training, testing and CV (if any) was randomly selected without replacement from the original data. It was repeated for ten times with different random number generating seeds to have ten statistically different subsets of data. These ten subsets were prepared to obtain the consistent results. It is worthwhile to point out that the CV data sets for the first three in Table 1 (hereafter called first group of data sets) are used only to tune ϕ for *AdaBoost.RT*, while other techniques (committee and non-committee) are trained on combination of training and CV data sets.

Both *AdaBoost.R2* and *AdaBoost.RT* algorithms were implemented in Java and incorporated in Weka software [23]. MT was ued as a *Weak Learner*. The reason to select this learning technique is that it is simple, easy and fast to train. Solomatine and Dulal [21] have applied MT in hydrologic modeling. We also performed the experiments with bagging and ANNs to compare the results.

1) *MT*: The first experiment with all data sets was to construct a single machine – a MT. The purpose of these experiments was two-fold. The first was to establish a benchmark against which to compare the boosting and bagging techniques. The second was to optimize the parameter of the single machine if any, such as pruning factor (number of linear models) in case of the MT. Often for better generalization it is necessary to prune the MT. The performance of the MT is presented in the Table 2. Note that the results are averaged for 10 different runs with independent data sets.

Table 2: Comparison of performance (RMSE) of different machines in test data set.

Data Set	MT	Bagging	ANN	AdaBoost.R2	AdaBoost.RT
Housing	3.62	3.24	3.54	3.23	3.23
SieveQ3	14.67	13.93	17.09	15.01	13.81
SieveQ6	26.55	26.47	32.87	28.36	26.37
Auto-Mpg	3.01	2.86	3.79	2.84	2.96
CPU	34.65	32.64	13.91	24.45	26.52
Friedman #1	2.19	2.06	1.51	1.82	1.72

2) *Boosting MT using AdaBoost.RT:* The next step was to implement experiments with the *AdaBoost.RT*. As described in the previous section, sub-optimal value of ϕ is selected using simple line search. If there is CV data set then the sub-optimal ϕ should correspond to the minimal RMSE on the CV set. Otherwise, the ϕ should be taken when the training error is minimal. Calibrating the ϕ adds to the total training time, however there is a tradeoff between the computation time (hence cost) and the prediction accuracy.

The experiment was performed using CV data set for first group of data sets. It was found that using CV set, the problem of overfitting the data is overcome, nevertheless the improvement of the performance is not significant. The experiments were also carried out for second group of data sets (last three in Table 1). Although the performance of *AdaBoost.RT* was not satisfactory on Auto-Mpg data set, it outperformed the single machine for CPU and Friedman#1 data set.

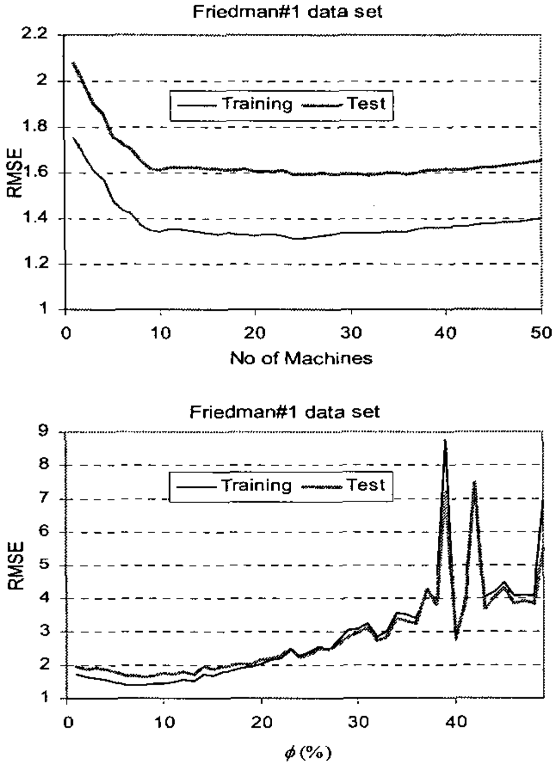

Fig. 1. Performance of *AdaBoost.RT* for different number of machines (top) and ϕ values (bottom) for the Friedman#1 data set.

It can be observed from Table 2 that *AdaBoost.RT* reduces the RMSE for all of the data sets considered. Two-tailed sign test also indicates that *AdaBoost.RT* is superior to single machine (MT) at a significance level better than 1%. The Fig. 1 (top) depicts the performance of the *AdaBoost.RT* on one out of the ten Friedman#1 test data set against the number of machines. As expected, RMSE is reduced significantly and reaches its minimum with 10 machines. With the increase of the number of machines the overall model overfits and RMSE on testing starts to grow. A large number of machines allow the composite committee machine to become very complex [9]. The performance of the *AdaBoost.RT* for different ϕ on one of Friedman#1 test data sets is presented in the Fig. 1 (bottom). It is observed that performance is sensitive to the value of ϕ.

3) Boosting MT using AdaBoost.R2: For comparison, MT was also boosted using *AdaBoost.R2*. The overall performance in testing is shown in Table 2. In general, the overall RMSE is satisfactory. There is a significant improvement in the performance over the CPU data set, however for the other data sets the performance is mixed.

4) Bagging MT: The experiments with bagging were also performed. The results of these experiments are presented in Table 2. Bagging outperformed the *AdaBoost.RT* in only one data set out of six namely Auto-Mpg by only 3.4%. On other data sets *AdaBoost.RT* supercedes bagging by as much as 18.8% (for CPU data set). These results demonstrate that *AdaBoost.RT* outperforms bagging, although there are no considerable differences for the first group of data sets.

5) ANN: Finally, an ANN model (multi-layer perceptron) was also set up. The hyperbolic tangent function was used for the hidden layer and the linear transfer function for the output layer. Due to time constraint, it was not possible to find the optimum values for the learning rate, momentum rate, number of hidden units and the number of epochs. Thus for all the data set, default value of 0.2 and 0.7 was taken as learning rate and momentum rate respectively. Number of hidden nodes was selected according to the number of attributes. The number of epoch was 1000. We used NeuroSolutions [14] and NeuralMachine [13] software for experiments.

The results are presented in the Table 2. It is observed that the performance of the ANN is worse for 3 data sets as compared to MT. This may be due to using non-optimal values of ANN parameters. However, for CPU and Friedman#1 data set ANN outperformed MT considerably.

6) Comparing all Methods: From the comparison of different machines performances as presented in the Table 2, it is found that results are mixed i.e. there is no clear winner for any particular machine over others. However if one is interested to analyze the relative performance of the algorithms more precisely then some quantitative measure is needed rather than just subjective comparison. For this reason we used the so-called scoring matrix (Table 3).

Table 3: Scoring matrix for different machines (values are expressed in %)

Machine	MT	Bagging	ANN	AdaBoost.R2	AdaBoost.RT	Total
MT	0	-5.4	-6.5	-9	-10.6	-31.6
Bagging	5.4	0	-2.2	-4	-5.6	-6.3
ANN	6.5	-2.2	0	0.1	-1.6	7.1
AdaBoost.R2	9	4	-0.1	0	-1.4	11.5
AdaBoost.RT	10.6	5.6	1.6	1.4	0	19.3

It shows the average relative performance (in %) of one technique over another technique for all the data sets considered. Element of scoring matrix $SM_{i,j}$ should be read as average performance of i_{th} machine (header row in Table 3) over j_{th} machine (header column in Table 3) and is calculated for N number of data sets as follows:

$$SM_{i,j} \underset{(for\ i\neq j)}{} = \frac{1}{N}\sum_{k=1}^{N}\frac{RMSE_{k,j}-RMSE_{k,i}}{\max(RMSE_{k,j},RMSE_{k,i})}$$

It can be clearly observed from the Table 3 that *AdaBoost.RT* scores highest.

7) Comparison with the Previous Research: In spite of the attention boosting receives in classification problems, relatively few comparisons between boosting techniques for regression exist. Drucker [6] performed some experiments using *AdaBoost.R2* with Friedman#1 and Housing data sets. He obtained RMSE of 1.69 and 3.27 for Friedman#1and Housing data sets respectively. Our results are consistent with his results, however there are certain procedural differences in experimental settings.

IV. Conclusions

A new boosting algorithm *AdaBoost.RT* for regression problems was proposed. Unlike several recent boosting methods for regression, which can be seen as gradient descent algorithms, *AdaBoost.RT* builds the regression model by simply changing the distribution of the sample. The training examples are classified in two classes by comparing the accuracy of prediction with the pre-set relative error threshold. The modified weight updating parameter not only ensures that the value of $\log(1/\beta_t)$ is non-negative, but also gives relatively more emphasis to the harder examples. Although unlike *AdaBoost.R2,* this boosting algorithm needs a parameter (relative error threshold) to be calibrated, the boosting technique doesn't depend on the size of the prediction error for the particular training example. Once examples are selected based on whether they are correctly predicted or not, all the incorrectly predicted examples are boosted by the same quantity. Owing to this characteristic, one may suggest *AdaBoost.RT* gives the better performance in presence of noises; the preliminary experiments support this, but should be further investigated.

The experiments with the data sets from UCI benchmark data sets clearly demonstrated that there is an improvement using this version of *AdaBoost* algorithm. The two-tail sign test indicates that *AdaBoost.RT* is better than a single machine with the confidence level higher than 99%. Relative performance measures (scoring matrix) also gives an indication of improved accuracy of *AdaBoost.RT* with respect to other machines considered. However, for more accurate statistically significant comparison, more experiments are needed.

As for the further research and improvements, an obvious step would be to automate the choice of an (sub) optimal value for the threshold depending on the characteristics of the data set and to test other functional relationships for weight updating parameter.

References

[1] Avnimelech, R., & Intrator, N. (1999). Boosting regression estimators. *Neural Computation,* 11 (2), 499-520.

[2] Blake, C.L., & Merz, C.J. (1998). UCI Repository of machine learning databases. Irvine, CA: Univ. of California, Dep. of Information and Computer Science. Available http://www.ics.uci.edu/ ~mlearn/MLRepository.html.

[3] Breiman, L. (1996a). Stacked regressor. *Machine Learning,* 24 (1), 49-64.

[4] Breiman, L. (1996b). Bias, variance, and arcing classifiers (Technical Report 460). *Statistics Dep., Univ. of California, Berkeley, CA.*

[5] Breiman, L. (1997). Prediction Games and Arcing Algorithms. *Neural Computation,* 11 (7), 1493-1518.

[6] Drucker, H. (1997). Improving Regressor using Boosting. *Douglas H. Fisher, Jr (Eds.), Proc. of the 14th Int. Conf. on Machine Learning (pp 107-115) Morgan Kaufmann.*

[7] Drucker, H. (1999). Boosting Using Neural Networks. In *A. J. C. Sharkey (ed.), In Combining Artificial Neural Nets (pp.* 51-*77). London: Springer-Verlag.*

[8] Duffy, N., & Helmbold, D.P. (2000). Leveraging for regression. *Proc.of the 13th Annu. Conf. on Comput. Learning Theory (pp. 208-219). San Francisco: Morgan Kaufmann.*

[9] Freund, Y., & Schapire, R. (1996). Experiment with a new boosting algorithm. *Proc. of the 13th Int. Conf. on Machine Learning (pp 148-156). Bari, Italy.*

[10] Freund, Y., & Schapire, R. (1997). A decision-theoretic generalisation of on-line learning and an application of boosting. *J. of Computer and System Sciences,* 55 (1), 119-139.

[11] Friedman, J.,Hastie, T., & Tibshirani, R. (2000). Additive Logistic Regression: A Statictical View of Boosting. *The Annals of Statistics,* 28 (2), 337-374.

[12] Friedman, J. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics,* 29 (5), 1189-1232.

[13] Neural Machine: http://www.data-machine.com.

[14] NeuroSolutions: http://www.nd.com.

[15] Opitz, D., & Maclin, R. (1999). Popular Ensemble Methods: An Empirical Study. *J. of Artificial Intelligenece Research,* 11 (1999), 169-198.

[16] Quinlan, J.R. (1992). Learning with continuous classes. *Proc. of the 5th Autralian Joint Conf. on AI (pp. 343-348).* World Secientific, Singapore.

[17] Quinlan, J.R. (1996). Bagging, boosting and C4.5. *Proc. of the 13th national Conf. on Artificial Intelligence (pp. 725-730). Portland, OR.*

[18] Ridgeway, G., (1999). The state of boosting. *Computing Science and Statistics,* 31, 172-181.

[19] Ridgeway, G., Madigan, D., & Richardson, T. (1999). Boosting methodology for regression problems. *Proc. of the 7th Int. Workshop on Artificial Intelligence and Statistics (pp. 152-161). Fort Lauderdale, FL.*

[20] Schapire, R. (1990). The strength of weak learnability. *Machine Learning,* 5 (2), 197-227.

[21] Solomatine, D.P., & Dulal, K.N. (2003). Model tree as an alternative to neural network in rainfall-runoff modelling. *Hydrological Science J.,* 48 (3), 399–411.

[22] Tresp, V. (2001). Committee Machines. *Yu Hen Hu and Jenq-Neng Hwang (Eds.), Handbook for neural network signal processing. CRC Press.*

[23] Weka Software: http://www.cs.waikato.ac.nz/ml/weka/

[24] Witten, I.H., & Frank, E. (2000). Data mining. *San Francisco: Morgan Kaufmann.*

[25] Zemel, R., & Pitassi, T. (2001). A gradient-based boosting algorithm for regression problems. Leen, T.K., Dietterich, T.G., & Tresp, V. (Eds.), Advances in Neural Information Processing Systems 13. MIT press.