Final submission, November 9, 2005 (Neural Computaion) NC ms 3101

Experiments with AdaBoost.RT, an Improved Boosting Scheme for Regression

D. L. Shrestha D. P. Solomatine
UNESCO-IHE Institute for Water Education
P.O. Box 3015, 2601 DA Delft, The Netherlands
Email: {d.shrestha, d.solomatine} @unesco-ihe.org

Abstract

The application of boosting technique to the regression problems has received relatively little attention in contrast to the research aimed at classification problems. This paper describes a new boosting algorithm *AdaBoost.RT* for regression problems. Its idea is in filtering out the examples with the relative estimation error that is higher than the preset threshold value, and then following the *AdaBoost* procedure. Thus it requires selecting the sub-optimal value of the error threshold to demarcate examples as poorly or well predicted. Some experimental results using M5 model tree as a weak learning machine for several benchmark data sets are reported. The results are compared to other boosting methods, bagging, artificial neural networks, and to a single M5 model tree. The preliminary empirical comparisons show higher performance of *AdaBoost.RT* for most of the considered data sets.

Keywords: boosting, bagging, model tree, weak learning machine, committee machine, relative error threshold

1 Introduction

Recently many researchers have investigated various techniques combining the predictions from multiple predictors to produce a single predictor. The resulting predictor is generally more accurate than an individual one. The ensemble of predictors is often called a committee machine (or mixture of experts). In a committee machine an ensemble of predictors (often referred as a weak learning machine or simply a machine) is generated by means of a learning process; the overall predictions of the committee machine is the combination of the individual committee member's predictions (Tresp, 2001).

[*Insert* Figure 1 *approximately here*]

Figure 1 presents basic scheme of a committee machine. Each machine (1 through *T*) is trained using training examples which are sampled from the given training set. Filter is employed when different machines are to be fed with different subsets (denoted as type A) of the training set; in this case the machines could be run in parallel. Flows of type B appear when machines pass unclassified data subsets to the subsequent machines thus making a hierarchical committee machine. The individual outputs y_i for each example from each machine are combined to produce the overall output y of the ensemble.

Bagging (Breiman, 1996a; 1996b) and boosting (Schapire, 1990; Freund and Schapire, 1996; 1997) are the two popular committee machines that combine the outputs from different predictors to improve overall accuracy. 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 (Freund and Schapire, 1996). Boosting works by repeatedly running a given weak learning machine on different distributions of training examples and combining their outputs. At each iteration the distributions of training examples depend upon the performance of the machine in the previous iteration. The method to calculate the distribution of the training examples is different for various boosting methods; the outputs of the different machines are combined using voting multiple classifiers in case of classification problems or weighted average or median in case of regression ones. There are different versions of boosting algorithm for classification and regression problems and they are covered in detail in the following section.

This paper introduces a new boosting scheme *AdaBoost.RT* for regression problems initially outline by the authors at the IJCNN conference in 2004 (Solomatine and Shrestha, 2004); it is based on the following idea. Typically in the regression it is not possible to predict the output exactly like in classification as real-valued variables may exhibit chaotic behavior, local non-stationarity and high levels of noise (Avnimelech and Intrator, 1999). Some discrepancy between the predicted value and the observed one is typically inevitable. A possibility here is to use insensitive margins to differentiate the correct prediction from incorrect

ones, like it is done, e.g., in support vector machines (Vapnik, 1995). The discrepancy (prediction error) for each example allows us to distinguish whether an example is well (acceptable) or poorly predicted. Once we define the measure of such discrepancy it is straightforward to follow the *AdaBoost* procedure (Freund and Schapire, 1997) by modifying the loss function that fits regression problems better.

AdaBoost.RT addresses some of the issues associated with the existing boosting schemes covered in section 2 by introducing a number of novel features. Firstly, *AdaBoost.RT* uses the so-called absolute relative error threshold ϕ to project training examples into two classes (poorly and well predicted examples) by comparing the prediction error (absolute relative error) with the threshold ϕ. (The reasons to use the absolute relative error instead of absolute error, as it is done in many boosting algorithms, are discussed in section 3). Secondly, the weight updating parameter is computed differently than e.g., in *AdaBoost.R2* algorithm (Drucker, 1997) to give more emphasis to the harder examples when error rate is very low. Thirdly, algorithm does not have to stop when the error rate is 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 many cases the performance is improved. Lastly, the outputs from individual machines are combined by weighted average while most of the methods consider median; and this appears to give better performance.

The paper covers a number of experiments with *AdaBoost.RT* using model trees (MTs) as weak learning machine, comparing it to other boosting algorithms, bagging and artificial neural networks (ANNs). Finally, conclusions are drawn.

2 The Boosting Algorithms

The original boosting approach is *boosting by filtering* and is described by Schapire (1990). It was motivated by the PAC (Probably Approximately Correct) learning theory (Valiant, 1984; Kearns and Vazirani, 1994). Boosting by filtering requires a large number of training examples which is not feasible in many real-life cases. This limitation can be overcome by using another boosting algorithm, *AdaBoost* (Freund and Schapire 1996; 1997) 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. This technique is applicable only when the weak learning machine can handle the weighted examples.

Originally boosting schemes were developed for binary classification problems. Freund and Schapire (1997) extended *AdaBoost* to a multi-class case, versions of which they called *AdaBoost.M1* and *AdaBoost.M2*. Recently several applications of boosting algorithms for classification problems have been reported (for example, Quinlan, 1996; Drucker, 1999; Opitz and Maclin, 1999).

Application of boosting to regression problems has received attention as well. Freund and Schapire (1997) extended *AdaBoost.M2* to boosting regression problems 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 the two drawbacks. Firstly, it expands each example in the regression sample into many classification examples which number grows linearly in the number of boosted iterations. Secondly, the loss function changes from iteration to iteration and even differs between examples in the same iteration. In the framework of *AdaBoost.R*, Ridgeway et al. (1999) performed experiments by projecting regression problems into classification ones on a data set of infinite size. Breiman (1997) proposed *arc-gv* (arcing game value) algorithm for regression problems. Drucker (1997) developed *AdaBoost.R2* algorithm, which is an ad hoc modification of *AdaBoost.R*. He conducted some experiments with *AdaBoost.R2* for regression problems and obtained good results.

Avnimelech and Intrator (1999) extended the boosting algorithm to regression problems by introducing the notion of weak and strong learning and appropriate equivalence between them. They introduced so-called *big errors* with respect to threshold γ which has to be chosen prior. They constructed triplets of weak learning machines and combined them to reduce the error rate using the median of the outputs of the three machines. Using the framework of Avnimelech and Intrator (1999), Feely (2000) introduced the *Big Error Margin* (*BEM*) boosting technique. Namee et al. (2000) compared the performance of *AdaBoost.R2* with the *BEM* technique.

Recently many researchers (for example, Friedman et al., 2000; Friedman, 2001; Duffy and Helmbold, 2000; Zemel and Pitassi, 2001; Ridgeway, 1999) have viewed boosting as a "gradient machine" that optimizes particular loss function. Friedman et al. (2000) 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 all similar except for the one considered by Zemel and Pitassi (2001). In this latter approach a gradient based boosting algorithm was derived, which forms new hypotheses by modifying only the distribution of the training examples. This is in contrast to the former approaches where the new regression models (hypotheses) are formed by changing both the distribution of the training examples and modifying the target values.

The following sub-sections describe briefly a boosting algorithm for classification problem – *AdaBoost.M1* (Freund and Schapire, 1997). The reason is that our new boosting algorithm is its direct extension for regression problems. The *threshold*- or *margin*-based boosting algorithms for regression problems including *AdaBoost.R2*, which are similar to our algorithm, are described as well.

2.1 AdaBoost.M1

The *AdaBoost.M1* algorithm (see appendix A) works as follows. The first 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 error rate ε_t as the ratio of incorrect classifications.

Knowing ε_t, weight updating parameter denoted by β_t is computed as follows:

$$\beta_t = \varepsilon_t / (1 - \varepsilon_t) \quad (1)$$

As ε_t is constrained on [0, 0.5], β_t is constrained on [0, 1]. β_t is a measure of confidence in the predictor. If ε_t is low, then β_t is also low and low β_t means high confidence in the predictor.

To compute the distribution for the next machine, the weight of each example is multiplied by β_t if the previous machine classifies this example correctly (this reduces the weight of the example), or the weight remains unchanged otherwise. The weights are then normalized to make their set a distribution. The process is repeated until the preset number of machines are constructed or $\varepsilon_t < 0.5$. Finally, the weight denoted by W is computed using β_t as:

$$W = \log(1/\beta_t) \quad (2)$$

W is used to weight the outputs of the machines when the overall output is calculated. It is to be noticed that W becomes larger when ε_t is low, and, consequently, more weight is given to the machine when combining the outputs from individual machines.

The essence of the boosting algorithm is that "easy" examples that are correctly classified by most of the previous weak machines will be given a lower weight, and "hard" example that often tend to be misclassified will get a higher weight. Thus, the idea of boosting is that each subsequent machine focuses on such hard examples. Freund and Schapire (1997) derived the exponential upper bound for the error rate of resulting ensemble and it is smaller than that of the single machine (weak learning machine). It does not guarantee that the performance on an independent test set will be improved, however, if the weak hypotheses are 'simple' and the total number of iterations is 'not too large', then the difference between the training and test errors can also be theoretically bounded.

As reported by Freund and Schapire (1997), *AdaBoost.M1* has a disadvantage that it is unable to handle weak learning machines with the error rate greater than 0.5. In this case the value of β_t will exceed unity, and consequently, the correctly classified examples will get a higher weight and W becomes negative. As a result, the boosting iterations have to be terminated. Breiman (1996c) describes a method by resetting all the weights of the examples 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 (1996c), Opitz and Maclin (1999) used a very small positive value of W (e.g. 0.001) rather than using a negative or 0 when ε_t is larger than 0.5. They reported results slightly better than those achieved by the standard *AdaBoost.M1*.

2.2 AdaBoost.R2

AdaBoost.R2 (Drucker, 1997) boosting algorithm is an ad hoc modification of *AdaBoost.R* (Freund and Schapire, 1997), which in its turn is an extension of *AdaBoost.M2* for regression problems. Drucker's method followed the spirit of *AdaBoost.R* algorithm by repeatedly using regression tree as a weak learning machine followed by increasing the weights of the poorly predicted examples and

decreasing the weights of the well predicted ones. Similar to the error rate in classification he introduced the average loss function to measure the performance of the machine; it is given by:

$$\overline{L_t} = \sum_{i=1}^{m} L_t(i) D_t(i) \tag{3}$$

where L_t is one of three candidate loss functions (see Appendix B) all of which are constrained on [0, 1]. The definition of β_t remains unchanged. However, unlike projecting the regression data into classification in *AdaBoost.R*, the reweighting procedure is formulated in such a way that the poorly predicted examples get higher weights and well predicted examples get lower weights:

$$D_{t+1}(i) = \frac{D_t(i)\beta_t^{(1-L_t(i))}}{Z_t} \tag{4}$$

In this way all the weights are updated according to the exponential loss functions of β_t so that the weight of the example with lower loss (smaller error) will be highly reduced, thus reducing the probability that this example will be picked up for the next machine. Finally the outputs from each machine are combined using the weighted median. Figure 2 presents the relation between β_t or W and $\overline{L_t}$.

[*Insert* Figure 2 *approximately here*]

Similar to *AdaBoost.M1*, *AdaBoost.R2* has a drawback that it is unable to handle weak learning machine with the error rate greater than 0.5. Furthermore, the algorithm is sensitive to noises and outliers as reweighting formulation is proportional to the prediction error. This algorithm, however, has an advantage that it does not have any parameter that has to be calibrated which is not the case in other boosting algorithms described in the subsequent subsections.

2.3 Boosting Method of Avnimelech and Intrator (1999)

Avnimelech and Intrator (1999) were the first to introduce the threshold-based boosting algorithm for regression problems which used an analogy between classification errors and big errors in regression. The idea of using threshold for big error is similar to *ε-insensitive loss function* used in support vector machines (Vapnik, 1995) where loss function of the examples having error less than ε is zero. The examples whose prediction errors are greater than the big error margin γ are filtered out and in subsequent iterations weak learning machine concentrates on them. The fundamental concept of this algorithm is that the mean squared error, which is often significantly greater than the square median of the error, can be reduced by reducing the number of large errors. Unlike other boosting algorithms, the method of Avnimelech and Intrator is an ensemble of only three weak learning machines. Depending upon the distribution of training examples, their method has three versions (see Appendix C).

In the beginning the training examples have to be spilt to three sets *Set 1*, *Set 2* and *Set 3* in such a way that the *Set 1* should be smaller than the other two sets. The first machine is trained on the *Set 1* (constituting a portion of the original training set, e.g. 15%). The training set for second machine is composed of all examples on which first machine has a *big error* on *Set 2* and the same number of

examples sampled from *Set 2* on which the first machine has a small error. Note that *big error* should be defined with respect to threshold γ which has to be chosen prior. In *Boost1,* the training data set for the last machine consists only of examples on which exactly one of the previous machines had a big error. But in *Boost2,* the training data set for the last machine is composed of the data set constructed for *Boost1* plus examples on which previous machines had big errors of different signs. The authors further modified this version and called it *Modified Boost2* where the training data set for the last machine is composed of the data set constructed for *Boost2* plus examples on which both previous machines had big errors, but there is a big difference between the magnitudes of the errors. The final output is the median of the outputs of the three machines. They proved that the error rate could be reduced from ε to $3\varepsilon^2 - 2\varepsilon^3$ or even less.

One of problems of this method is the selection the optimal value of threshold γ for big errors. Theoretically, the optimal value may be the lowest value for which there exists a γ-weak learner. A γ-weak learner is such a learner for which exists some $\varepsilon < 0.5$, and for any given distribution D and $\delta > 0$ it is capable to find function f_D with probability 1-δ such that $\left[\sum_{i:\ f(x_i)-y_i>\gamma} D(i) \right] < \varepsilon$. There are however, practical considerations such as limitation of data sets and the limited number of machines in the ensemble, which makes it difficult to choose the desired value of γ in advance. Furthermore, an issue concerning split of the training data into three subsets may arise. Since all the training examples are not used to construct the weak learning machine, waste of data is crucial issue for small data sets. Furthermore, similarly to *AdaBoost.R2*, the algorithm does not work when the error rate of the first weak learning machine on *Set 2* is greater than 0.5. It should be also noted that the use of an absolute error to identify big errors is not always the right measure in many practical applications when the variability of the target values is very high (explanation follows later in section 3).

2.4 *BEM* boosting method

Big Error Margin (*BEM*) boosting method (Feely, 2000) is quite similar to the *AdaBoost.R2* method. It is based on an approach of Avnimelech and Intrator (1999). Similar to their method, the prediction error is compared with the preset threshold value called *BEM* and the corresponding example is classified into either well or poorly predicted.

BEM method (see Appendix D), however, counts the number of correct and incorrect predictions by comparing prediction error with the preset *BEM* which has to be chosen prior. The prediction is considered to be incorrect if the absolute value of prediction error is greater than *BEM.* Otherwise, the prediction is considered to be correct. Counting the numbers of correct or incorrect predictions allows for computing the distribution of the training examples using the so-called *UpFactor* and *DownFactor* as below:

$$\begin{aligned} Upfactor_t &= m / errCount_t \\ Downfactor_t &= 1 / Upfactor_t \end{aligned} \qquad (5)$$

Using these values, the numbers of copies of each training example to be included for the subsequent machine in the ensemble is calculated. So if the example is correctly predicted by the preceding machine in the ensemble, then the number of copies of this example for the next machine is given by the number of its copies in the current training set multiplied by the *DownFactor.* Similarly, for an incorrectly predicted example the number of its copies to be presented to the subsequent machine is the one multiplied by the *UpFactor*. Finally the outputs from individual machine are combined to give the final output.

Similar to the method of Avnimelech and Intrator (1999), this method requires tuning of the value of *BEM* in order to achieve the optimum results. The way the training examples' distribution is represented is different from that in other boosting algorithms. In *BEM* boosting method, the distribution represents how many times an example will be included in the training set rather than the probability of it appearing. Furthermore, it has another drawback that the size of the training data set is changing for each machine and may increase to an impractical value after few iterations. In addition, this method has a problem encountered in the boosting method of Avnimelech and Intrator (1999) due to the use of absolute error measure when the variability of the target values is very high.

In the new algorithm presented in the next section an attempt has been made to resolve some of the problems mentioned above.

3 Methodology

This section describes a new boosting algorithm for regression problems called *AdaBoost.RT* (here *R* stands for regression and *T* – for threshold).

***AdaBoost.RT* algorithm**

1. Input:
- Sequence of m examples $(x_1, y_1),\ldots,(x_m, y_m)$, where output $y \in R$
- Weak learning algorithm *Weak Learner*
- Integer T specifying number of iterations (machines)
- Threshold ϕ $(0< \phi <1)$ for demarcating correct and incorrect predictions

2. Initialize:
- Machine number or iteration $t = 1$
- Distribution $D_t(i) = 1/m$ for all i
- Error rate $\varepsilon_t = 0$

3. Iterate while $t \leq T$
- Call *Weak Learner*, providing it with distribution D_t
- Build the regression model: $f_t(x) \rightarrow y$
- Calculate absolute relative error for each training example as $ARE_t(i) = \left| \frac{f_t(x_i) - y_i}{y_i} \right|$
- Calculate the error rate of $f_t(x)$: $\varepsilon_t = \sum_{i : ARE_t(i) > \phi} D_t(i)$
- Set $\beta_t = \varepsilon_t^n$, where n = power coefficient (e.g. linear, square or cubic)

- Update distribution D_t as

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t \text{ if } ARE_t(i) <= \phi \\ 1 \quad \text{otherwise} \end{cases}$$

  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:

$$f_{fin}(x) = \sum_t \left(\log \frac{1}{\beta_t} \right) f_t(x) \Big/ \sum_t \left(\log \frac{1}{\beta_t} \right)$$

Similar to the boosting methods described in section 2, the performance of the machine is evaluated by computing the error rate ε_t. Furthermore, similar to threshold based boosting methods of Avnimelech and Intrator (1999) and *BEM*, the regression problem in *AdaBoost.RT* is projected into the binary classification problem while demarcating well predicted and poorly predicted examples. However, unlike absolute big error margin in methods of Avnimelech and Intrator (1999) and *BEM* we use the absolute relative error (referred to simply as relative error afterwards) to demarcate examples as either well or poorly predicted. If the relative error for any particular example is greater than the preset relative error, the so-called *threshold* ϕ, the predicted value for this example is considered to be poorly predicted, otherwise it is well predicted. Weight updating parameter β_t is computed differently than in *Adaboost.M1* or *AdaBoost.R2*. We reformulated the expression for β_t to overcome the situation when the machine weight *W* becomes negative if ε_t is greater than 0.5. β_t will be linear to ε_t when the value of power coefficient *n* is one. Other possible values of *n* are two (square law) and three (cubic law). As the value of *n* increases, then relatively more weight is given to the harder examples in cases when ε_t is very low (close to 0.1 or even less) (as β_t deviates further from 1) and the machine concentrates on these hard examples. However, very high value of *n* may cause instability of the algorithm because the machine will be trained only on a limited number of harder examples and may fail to generalize. So in our experiment we used square law for β_t. It is worthwhile to mention that the normalized *W* used for combining the outputs from each machine is independent of *n*.

Unlike *AdaBoost.M1* and *AdaBoost.R2*, our *AdaBoost.RT* does not have a "natural" stopping rule (when ε_t exceeds 0.5) but can generate any number of machines. This number, in principle, should be determined on the basis of analysis of the cross validation error (training stops when the cross validation error starts to increase). We found that even if ε_t for some of the machines in the ensemble is greater than 0.5, the final output of the combined machine is better than that of a single machine. Finally, the outputs from different machines are combined using weighted average.

Similar to the threshold based boosting methods of Avnimelech and Intrator (1999) and *BEM*, *AdaBoost.RT* has a drawback that the threshold ϕ should be chosen prior, and hence introduces additional complexity that has to be handled. The experiments with *AdaBoost.RT* have shown that the performance of the committee machine is sensitive to ϕ. Figure 3 shows the relation between ϕ and ε_t.

If ϕ is too low (for example, < 1%), 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 a few "hard" examples, which are often outliers, and 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 value of ϕ, it is required to find out statistics on the prediction errors (relative errors) for the single machine before committing the boosting. Theoretically, the lower and upper boundary of ϕ should reside between the minimum and maximum values of the relative error. Simple line search within these boundaries may be required to find the sub-optimal value of ϕ that minimizes root mean squared error (RMSE) and this can be done using the cross validation data set. Note that as ϕ is relative and is constrained to [0, 1].
[*Insert* Figure 3 *approximately here*]

Loss function and evaluation of the machine

An obvious question may arise while dealing with the regression problems: what is the loss function? For *Adaboost.R2,* it is explicitly mentioned in the algorithm and it has three different functional forms. However, for *AdaBoost.RT* we tried to replace misclassification error rate as used by Freund and Schapire (1997) with a loss function that better fits regression problems. Note that because of using the threshold in *AdaBoost.RT* the loss function is represented by a logical expression (computer code) rather than by a simple formula as in *AdaBoost.R2*.

Many boosting schemes use absolute error as the loss function, and this choice can be justified in many applications. In *AdaBoost.RT*, however the error is measured by the relative error. This choice is based on our belief that any machine learning algorithms, if possible, should reflect the essence of the problem or process being learned. One of the motivations to build an improved boosting scheme was to increase the accuracy of predictors used in engineering and hydrology. In these domain areas experts often judge the predictor's accuracy by the relative error, rather than the absolute one. For example, consider a hydrological model that predicts water level in a river. A hydrologist would ignore absolute error of 10 cm when the target water level is 10 m (relative error of 1%). However, the same 10 cm error for target level of 50 cm is not small (20%). It can be said that relative error, in a way, adapts to the magnitude of the target value and, in our view, using relative error is justified in many real-life applications.

However, if the target values are zero or close to zero, the following phenomenon may appear. For small target values, relative errors could be high even if a machine makes a good prediction (in the sense of absolute error); in this case such examples will be assigned (maybe "unfairly") a higher probability to be selected for the next machine. This may lead to its specialization on the examples with small target values only just because the model accuracy was measured by relative error and not by absolute error. We have not specially investigated this issue in our experiments. A possible remedy for this potential problem is special handling of examples with very low target values (e.g., by appropriately modifying relative error), or data transformation (adding a constant).

It is to be noted that the relative error is employed to evaluate the

performance of the weak learning machines. However, to make the comparisons with other machines and with previous research on benchmark data sets, the classical RMSE is used as evaluation of the performance of the ensemble machine (see, e.g., Cherkassky and Ma, 2004). Minimizing relative absolute errors greater than ϕ will reduce the number of large errors and consequently will lead to the lower value of RMSE.

4 Bagging

Bagging (Breiman, 1996a) is another popular ensemble technique used to improve the accuracy of a single learning machine. In bagging multiple versions of a predictor are generated and used to obtain an aggregated predictor (Breiman, 1996b). Bagging is "bootstrap" (Efron and Tibshirani, 1993) aggregating that works by training each classifier on a bootstrap sample, i.e., a sample of size m chosen uniformly at random with replacement from the original training set of size m. Boosting and bagging are to some extent similar: in both of them each machine is trained on different sets of the training examples and the output from the each machine is combined to get a single output. There is however a considerable difference in sampling the training examples and combining the outputs from each machine.

In bagging, for each machine in a trial $t = 1, 2\ldots, T$, training data sets of m examples are randomly sampled with replacement from the whole training set of m examples. Thus it is possible that many of the original examples may be repeated in the resulting training set while others may be left out. In boosting, however, the training examples are sampled according to the performance of the machine in the previous iterations. Based on these training examples, classifier h_t or predictor $f_t(x)$ is constructed. In bagging, the prediction from each machine in an ensemble is combined by simply taking an average of predictions from each machine. In boosting predictions are given different weights and then averaged.

Breiman (1996a; 1996c) showed that bagging is effective on "unstable" learning algorithm where small perturbation in the training set can cause significant changes in the predictions. He also claimed that neural networks and regression trees are examples of unstable learning algorithms.

5 Experimental Setup

5.1 Data sets

In order to evaluate the performance of boosting, several data sets were selected. The first group was composed of some data sets from the UCI repository (Blake and Merz, 1998) and the Friedman #1 set (Friedman, 1991). These data sets can be also used as benchmarks for comparing results with the previous research. Second group of data sets is based on hydrological data which was used in the previous research (Solomatine and Dulal, 2003) where ANNs and MTs were used to predict river flows in the Sieve catchment in Italy. In addition, for comparison with the boosting method of Avnimelech and Intrator, we also used Laser data from the Sante Fe time series competition (data set A) (Weigend and Gershenfeld, 1993). A brief description of the data sets follows.

Hydrological data: Two hydrological data sets *SieveQ3* and *SeiveQ6* are constructed from hourly rainfall (RE_t) and river flow (Q_t) data in Sieve river basin in Italy. From the history of previous rainfall and runoff, a lagged vector of rainfall and runoff time series is constructed. 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 1854 examples. SieveQ3 data set is the 3 hour ahead prediction for the values of river flow and consists of 6 continuous input attributes. The regression model is formulated as $Q_{t+3} = f\,(RE_t, RE_{t-1}, RE_{t-2}, RE_{t-3}, Q_t, Q_{t-1})$. SieveQ6 set is used for the 6 hour ahead prediction for the values of river flow and consists of 2 continuous input attributes. The regression model is formulated as $Q_{t+6} = f\,(RE_t, Q_t)$.

Benchmark data: Four data sets from this group are *Boston housing*, *Auto-Mpg*, *CPU* from the UCI repository and the *Friedman#1*. Boston housing has 506 cases and 12 continuous and 1 binary-valued input attributes. The output attribute is the median price of a house in the Boston area in the USA. Auto-Mpg data concerns city-cycle fuel consumption of automobiles in miles per gallon, to be predicted in terms of 3 multi-value discrete and 4 continuous attributes. CPU data characterizes the relative performance of computers on the basis of a number of relevant attributes. The data consists of 6 continuous and one discrete input attributes. Friedman#1 is the synthetic data that is used in the paper on multivariate adaptive regression splines (MARS) (Friedman, 1991). It has 10 independent random variables $x_1, \ldots x_{10}$, each of which is uniformly distributed over [0, 1]. The response (output attribute) is given by:

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon \qquad (6)$$

where, ε is N [0, 1] and represents variables that have no prediction ability ($x_6, \ldots x_{10}$).

Laser data: This time series is the intensity of a NH3-FIR laser which exhibits Lorenz-like chaos. It has sampling noise due to the A/D conversions to 256 discrete values. We constructed input vectors using 16 previous values to predict the next value as Avnimelech and Intrator (1999) did.

For SieveQ3, SieveQ6 and Boston housing, the data set was randomly divided into training, cross validation (sometimes referred to as simply validation) and test subsets. The division of data for Boston housing is kept as in the previous research by Drucker (1997). The division of data for SieveQ3 and SieveQ6 is based on the previous research by Solomatine and Dulal (2003). The remaining three data sets from UCI repository were randomly divided into training (66% of the data set) and test set (34%) only. Laser data set was randomly split into training (75% of the data set) and test set (25%) which is similar to the split used by Avnimelech and Intrator. Table 1 summarizes the characteristics of the data sets.

[*Insert* Table 1 *approximately here*]

5.2 Methods

The data sets for training, testing and cross validation (if any) were randomly constructed without replacement from the original data. It was repeated ten times

with different random number generating seeds to have ten statistically different data sets. It is worthwhile to point out that the cross validation data sets for the first three data sets in Table 1 (hereafter referred to as *Group 1* of data sets) are used only to tune sub-optimal ϕ value for *AdaBoost.RT*, big error γ for the method of Avnimelech and Intrator and the error margin for *BEM* boosting method, while other techniques (committee and non-committee) are trained on combination of training and cross validation data sets.

All boosting algorithms for regression problems mentioned in the section 2 including our boosting method were implemented in Java by the authors and in form of classes added to the *Weka* software (Witten and Frank, 2000). Our implementation of boosting and bagging algorithms makes it possible to boost a number of weak learners that are already implemented in the original version of *Weka* – M5 model tree (MT), ANN, etc.

In all the experiments with the boosting algorithms we used MT (Quinlan, 1992; see also Witten and Frank, 2000) as a weak learning machine. The reason to select this learning technique is that it is simple, easy and fast to train. Solomatine and Dulal (2003) have shown that MT can be used as an alternative to ANN in rainfall-runoff modeling. MT is similar to a regression tree, but it has 1^{st} order (linear) regression models in leaves, as opposed to the 0^{th} order model in regression trees.

In addition, to show the applicability of the boosting algorithms for different learning machines, experiments were conducted for the Laser data set using ANNs as a weak learning machine. Besides these, we further performed the experiments with bagging using MT as a weak learning machine and carried out experiments with ANNs to compare the results.

Each experiment with boosting, bagging or ANNs was repeated ten times on subsets of all seven data sets constructed as explained above. Moreover, we constructed ten machines for each subset of data sets to form a committee machine for bagging and all boosting methods (except that of Avnimelech and Intrator). All possible efforts were undertaken to ensure the fair comparison of various methods.

6 Experimental Results and Discussion

This section reports the results of applying our boosting algorithm to several data sets as mentioned in the previous section. We investigated the use of cross validation data sets to overcome overfitting problem while selecting sub-optimal value for the threshold. The performance of *AdaBoost.RT* was compared against different number of machines and demonstrated that our boosting algorithm achieved an overall better result even in case of adding more weak learners in boosting schemes when error rate becomes greater than 0.5. Finally, we investigated the problem of bias towards the values more commonly occurring in the training examples. Performance of AdaBoost.RT was compared with that of the other boosting methods, bagging and ANNs.

6.1 Weak Learner: M5 Model Tree (MT)

The first experiment with all data sets was aimed at building a single machine – a

MT. The purpose of these experiments was two-fold. First, to establish a benchmark against which to compare the boosting and bagging techniques. Second, to optimize the parameter of the single machine, if any, such as pruning factor (number of linear models) in case of the MT (often prunining of MT is necessary for better generalization).

A number of MTs with the varying complexity (number of linear models) were generated; their performance for one of the Boston housing data sets is presented in the Figure 4.

[*Insert* Figure 4 *approximately here*]

6.2 Boosting of M5 Model Tree

AdaBoost.RT

The next step entails implementation of experiments with the *AdaBoost.RT*. As described in section 3, the appropriate value for ϕ is to be selected. This was achieved via a calibration process in which a possible range of values was tested to find the sub-optimal value of ϕ. In the developed computer program the user can select the lower and the upper boundary and incremental value of ϕ, and run the algorithm within the range of the values. If there is a cross validation data set then ϕ should correspond to the minimal RMSE on the cross-validation set. Otherwise, ϕ should be taken when the training error is minimal; in this case, however, there is a possibility of overfitting. This calibration process is, in fact, a simple line search that minimizes the cost function (in this case RMSE). Calibrating ϕ adds to the total training time and indeed, there is a tradeoff between the computation time (hence cost) and prediction accuracy.

The experiment was performed using the cross validation data set for the *Group 1* of data sets. Table 2 shows that using cross validation set, the problem of overfitting the data is overcome, however the improvement of the performance is marginal.

[*Insert* Table 2 *approximately here*]

The experiments were also carried out for the *Group 2* of data sets that do not have corresponding cross validation data sets. Although the performance of *AdaBoost.RT* was not satisfactory on Auto-Mpg data set, it outperformed the single machine for CPU, Friedman#1 and Laser data set (23.5%, 21.5% and 31.7% RMSE reduction respectively).

[*Insert* Table 3 *approximately here*]

It can be observed from Table 3 that *AdaBoost.RT* reduces the RMSE for all of the data sets considered. In 70 experiments (10 experiments for each data set) *AdaBoost.RT* wins over the single machine in 54 experiments (i.e. 77% times on average, see Table 5). The two-tail sign test also indicates that *AdaBoost.RT* is significantly better than the single machine at the 99% confidence level. Note that as pointed out by Quinlan (1996) and Opitz and Maclin (1999) there are cases where the performance of committee machine is even worse than the single machine. This is the case in 16 experiments out of 70. Freund and Schapire (1996) also observed that boosting is sensitive to noise and this may be partly responsible for the increase in error in these cases.

[*Insert* Figure 5 *approximately here*]

We investigated the performance of *AdaBoost.RT* with up to 50 machines (Figure 5) for one of the Friedman#1 data sets. It is observed that RMSE is reduced significantly already when only 5-6 machines were trained and after about 10 machines the error was not decreasing much. So we used 10 machines in all experiments; additional reason was that Breiman (1996b) also used 10 machines in bagging.

[*Insert* Figure 6 *approximately here*]

The performance of *AdaBoost.RT* for different values of ϕ on one of the Friedman#1 test data sets is presented in the Figure 6. It can be observed that performance is sensitive to the value of ϕ. The minimum error corresponds to ϕ of around 8% and starts increasing continuously with the increase of ϕ. Finally, at the value of around 40%, the *AdaBoost.RT* becomes unstable due to overfitting and the boosting of noise.

[*Insert* Figure 7 *approximately here*]

Figure 7 shows RMSE and the error rate of *AdaBoost.RT* against the number of machines for one of the Friedman#1 data sets. It demonstrates that *Adaboost.R2*-like algorithm would have stopped at 6 machines when error rate becomes greater than 0.5, but *AdaBosst.RT* continued and achieved higher performance. There is of course a danger of boosting noise, but it is present in any boosting algorithm. Again, ideally, stopping should be associated with the cross validation error and not with the error rate on training set. However, if this is not done, then the number of machines should be determined on the basis of experiments like shown on Figure 5. Our final results show, indeed, that the generalization properties of *Adaboost.RT* are good when 10 machines are used beyond the level of 0.5 error rate.

[*Insert* Figure 8 *approximately here*]

Figure 8 depicts the histogram of training and test data set for one of the Friedman#1 data sets. It is observed that the large numbers of training and test examples are in the middle range (13-19). Many experiments have shown that most of the learning machines are biased towards values more commonly occurring in the training examples and thus there would be high probability that the extreme values will be poorly predicted. As the main idea behind the boosting algorithm is to improve the performance on the hard examples (often extreme events), it is possible that the performance on tails (extreme events) may be improved at the expense of the commonly occurring cases. This effect was also reported by Namee et al. (2000). We investigated whether *AdaBoost.RT* also suffers from this problem. One of the Friedman#1 data sets was selected for this purpose and the resulting average error (absolute error in this case) for different ranges of test data is presented on Figure 9. It can be seen that *AdaBoost.RT* outperformed MT on all ranges of data set (low, medium and high).

[Insert Figure 9 approximately here]

Figure 10 shows that *AdaBoost.RT* works very well for the extreme cases and is quite successful to reduce the errors on these parts of the data range without deteriorating the performance in the middle range.

[*Insert* Figure 10 *approximately here*]

AdaBoost.R2

For comparison, experiments with *AdaBoost.R2* were performed as well; MT was used as a weak learning machine. We used linear and square loss function to calculate the weight updating parameter. The performance of this boosting method for all test data sets is shown in Table 3. In general, the overall RMSE is satisfactory. There is a significant increase of performance over Laser and CPU data set as RMSE is reduced from 4.04 to 2.69 (33.4% reduction) and 34.65 to 24.45 (29.4% reduction) respectively. For the other three data sets the performance improvement was less significant: from 16.9% for Friedman#1 to almost no improvement for Auto-Mpg. However, for SieveQ3 and SieveQ6 the performance deteriorated.

The reason for deteriorating performance of *AdaBoost.R2* can be seen in boosting of outliers and noise. Noisy examples that obviously have large prediction error are strongly boosted while those with smaller prediction error are weakly boosted as weight updating parameter depends upon the amount of prediction error. Intuitively, some examples will typically generate extraordinarily high prediction errors thus resulting in the low overall performance of the committee machine.

Boosting method of Avnimelech and Intrator

Experiments with boosting method of Avnimelech and Intrator were conducted as well. 15% of the training set was assigned to the *Set 1* and the rest were split to two equal sets similar to the experiments performed by Avnimelech and Intrator (1999) with Laser data set. As in *AdaBoost.RT*, big error γ is selected using cross validation set for the *Group 1* of data sets and using training set for the *Group 2*. We used the *Modified Boost2* version of their boosting method. The results are reported in Table 3. The performance of boosting is worse than that of the single machine for all data sets except SieveQ3. The reason could be that there is no sufficient data available for training due to the partition of training data into three subsets. Furthermore, non-optimal partition of training data and selection of big error leads to deteriorate the performance.

***BEM* boosting method**

Finally, the experiments with the *BEM* boosting method (Feely, 2000) were repeated. Similar to other threshold based boosting algorithms, *BEM* value is calibrated using cross validation (for the *Group 1* of data set) and training data set (for the *Group 2*). The results are reported in Table 3. The *BEM* method is more accurate than the single machine for the benchmark and Laser data sets. RMSE on test data sets is reduced by 33% for CPU data set and 22.8% for Friedman #1 data set. However, for hydrological data sets this method suffers from overfitting: RMSE in training drops dramatically while the RMSE in testing is much higher than that of the single machine. The reason for overfitting is that after several iterations the training data consists of a large number of copies of just a few examples (examples having higher errors) and, consequently, fails to generalize. Furthermore, it is noticed that *BEM* boosting method is highly sensitive to the value of *BEM*. The computation cost is very high as compared to other boosting algorithms since the size of training data increases after few iterations.

6.3 Bagging

The experiments with bagging were performed as well. Since Breiman (1996b) noted that the most of the improvement from bagging is evident within ten replications and since it was interesting to see the performance improvement that can be brought by a single order of magnitude increase in computation cost, the number of machines was set to ten. The results of these experiments are presented in Table 3. Bagging outperformed the *AdaBoost.RT* in only one data set out of seven, namely Auto-Mpg (by only 3.4%). On the other 6 data sets, *AdaBoost.RT* supersedes 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 *Group 1* of data sets. Bagging, however always improves the performance over a single machine, although only marginally. The two-tail sign test also indicates that bagging is significantly better than the single machine at the 99% confidence level. It is worthwhile to mention that the bagging seems to be always reliable and consistent but may be not the most accurate method.

6.4 Neural Networks

Finally, an ANN model (multi-layer perceptron) was also set up to compare the techniques described above against a popular machine learning method. The hyperbolic tangent function was used for the hidden layer and the linear transfer function – for the output layer. For all the data sets, default value of 0.2 and 0.7 was taken as learning and momentum rate respectively. The number of epochs was 1000. Number of hidden nodes was selected according to the number of attributes and in most of cases it is less than the number of attributes. We used NeuroSolutions and NeuralMachine software.

The results are presented in the Table 3. While comparing to MT, it is observed that for the three out of seven data sets performance of the ANN is worse. For example, in case of SieveQ6, RMSE value achieved using the ANN is 32.87 as compared to 26.55 using MT. This may be due to using non-optimal values of ANN parameters. However, for Freidman#1 ANN outperformed MT considerably (RMSE is lower by more than 30%). Interestingly, for CPU data set, ANN is considerably better than all other methods.

6.5 Use of Different Combining Methods for Boosting

It has been already mentioned above that *AdaBoost.RT* uses the weighted average while combining outputs of the machines in the ensemble. In contrast, *AdaBoost.R2* uses the weighted median.

In order to compare these two competent boosting algorithms, with different schemes for combining outputs, we repeated experiments for the *Group 1* of data sets using both weighted average and median. The results are presented in the Table 4. When using the weighted average in *AdaBoost.R2*, *AdaBoost.RT* leads for all three data sets. Furthermore, if weighted median is used for *AdaBoost.RT*, *AdaBoost.R2* leads only on one data set. If we examine 30 experiments in this group of data sets, it is observed that *AdaBoost.RT* outperformed *AdaBoost.R2* for 23 data sets using weighted average while *AdaBoost.R2* outperformed *AdaBoost.RT* for only 12 data sets using weighted median as the combination

scheme. This means that *AdaBoost.RT* outperforms *AdaBoost.R2* regardless of the combination scheme on the data sets considered.
[*Insert* Table 4 *approximately here*]

6.6 Comparison of All Methods

From the performance comparison as presented in the Table 3, it has been found that results are mixed, i.e. there is no machine that beats all other machines on all data sets. Since the results presented in the Table 3 are average for 10 independent runs, it is also possible to count how many times one machine beats another in 70 independent runs for all data sets (10 runs for each data set). Table 5 presents the qualitative performance of one machine over another in percentage (also called *qualitative scoring matrix*). Its element $QSM_{i,j}$ should be read as the number of times the performance of i^{th} machine (header row in Table 5) beats the j^{th} machine (header column) in 100 independent runs for all data sets, and is given by the following formula:

$$QSM_{i,j} = \begin{cases} \dfrac{Countif\ RMSE_{K,i} > RMSE_{K,j}}{Nt} & \forall K,\quad i \neq j \\ 0, & i = j \end{cases} \tag{7}$$

where $RMSE_{K,i}$ is root mean squared error of the i^{th} machine (rows' headers in the first column in Table 5), $RMSE_{K,j}$ is root mean squared error of the j^{th} machine (columns' header in the first row), Nt is total number of independent runs for all data sets, and K = 1, 2,..., Nt. The following relation holds:

$$QSM_{i,j} + QSM_{j,i} = 1 \qquad \forall\, i, j \ \text{ and } \ i \neq \text{j} \tag{8}$$

[*Insert* Table 5 *approximately here*]

For example, the value of 77 appearing in the 2nd row and the 1st column means that *AdaBoost.RT* beats MT 77% of times on average. Table 5 demonstrates that *AdaBoost.RT* is better than all other machined considered 69% of times, while 62%, 47%, and 24% for *AdaBoost.R*, *BEM* boosting method, and method of Avnimelech and Intrator respectively.

If one is interested to analyze the relative performance of the algorithms more precisely, then a measure reflecting the relative performance is needed. For this reason we used the so-called *quantitative scoring matrix*, or, simply, *scoring matrix* (Solomatine and Shrestha, 2004) as shown in Table 6. It shows the average relative performance (in %) of one technique (either single machine or committee machine) over another one for all the data sets. The element of scoring matrix $SM_{i,j}$ should be read as average performance of the i^{th} machine (header row in Table 6) over the j^{th} machine (header column) and is calculated as follows:

$$SM_{i,j} = \begin{cases} \dfrac{1}{N} \sum_{k=1}^{N} \dfrac{RMSE_{k,j} - RMSE_{k,i}}{\max(RMSE_{k,j}, RMSE_{k,i})}, & i \neq j \\ 0, & i = j \end{cases} \tag{9}$$

where N is the number of data sets. The value of 13.7 appearing in the 2nd row and the 1st column (Table 6) means that the performance of *AdaBoost.RT* over the MT is 13.7% higher, if averaged over all seven data sets considered by summing up all

the elements' values row-wise one can determine the overall score of each machine. It can be clearly observed from the Table 5 and Table 6 that *AdaBoost.RT* on average scores highest, which implies that on the used data sets *AdaBoost.RT* is comparatively better than the other techniques considered.
[*Insert* Table 6 *approximately here*]

6.7 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 (1997) performed some experiments using *AdaBoost.R2* with Friedman#1 and Boston housing data sets. He obtained RMSE of 1.69 and 3.27 for Friedman#1and Boston housing data sets respectively. Our results are consistent with his results, however there are certain procedural differences in experimental settings. He used 200 training examples, 40 pruning samples and 5000 test example per run and 10 runs to determine the best loss functions. In our experiments we used only 990 training examples and 510 test examples, and the number of iterations was only 10 against his 100. We did not optimize the loss functions and moreover the weak learning machine is completely different, as we used M5 model tree (potentially more accurate), whereas he used a regression tree.
[*Insert* Table 7 *approximately here*]

We also conducted experiments with Laser data set using ANNs as a weak learning machine to compare with the previous work of Avnimelech and Intrator (1999). The architecture of neural networks is the same as described before. The hidden layer consisted of 6 nodes. The number of epochs was limited to 500. The results are reported in Table 7 and shows that *AdaBoost.RT* is superior as compared to all other boosting methods. RMSE of *AdaBoost.RT* is 2.48 as compared to 2.84 of *AdaBoost.R2*, 3.13 of Avnimelech and Intrator's method, and 3.42 of *BEM* method. To compare to this latter work, the normalized mean squared error (NMSE, mean squared error divided by the variance across the target value) was also calculated. They obtained NMSE value of 0.0047 while ours was 0.0044 using their method on Laser data set. There were some procedural differences in experimental setting: they used 5 different partitions of 8000 training examples, 2000 test examples, whereas we used 10 different partitions of 7552 training examples and 2518 test examples. In spite of these differences it can be said that our result is consistent with theirs.

7 Conclusions

Experiments with a new boosting algorithm *AdaBoost.RT* for regression problems were presented and analyzed. Unlike several recent boosting algorithms for regression problems that follow an idea of gradient descent, *AdaBoost.RT* builds the regression model by simply changing the distribution of the sample, and is a variant of *AdaBoost.M1* modified for regression. The training examples are projected into two classes by comparing the accuracy of prediction with the preset relative error threshold. The idea of using error threshold is analogous to insensitivity range used, e.g., in support vector machines. Loss function is computed using relative error rather than absolute error; in our view it is justified

in many real-life applications. Unlike most of the other boosting algorithms, the *AdaBoost.RT* algorithm does not have to stop when the error rate is greater than 0.5. The modified weight updating parameter not only ensures that the value of machine weight is non-negative, but also gives relatively higher emphasis on the harder examples. Boosting method of Avnimelech and Intrator (1999) suffers from the problem of data shortage, while *BEM* boosting method (Feely, 2000) is time consuming to handle large data sets. In this sense *AdaBoost.RT* would be a preferred option.

If compared to *AdaBoost.R2*, however, *AdaBoost.RT* needs a parameter to be selected prior and this introduces additional complexity that has to be handled as in other threshold based boosting algorithms (for example, by Avnimelech and Intrator and *BEM*). The algorithmic structure of *AdaBoost.RT* is such that it updates the weight updating parameter (used to calculate the probability to be chosen for the next machine) by the same value. This feature ensures that the outliers (noisy examples) do not dominate the training sets for the subsequent machines.

The experimental results demonstrated that *AdaBoost.RT* algorithm outperforms the single machine (i.e. weak learning machine for which a M5 model tree was used) for all of the data sets considered. The two-tail sign test also indicates that *AdaBoost.RT* is better than a single machine with the confidence level higher than 99%. Comparing with the other boosting algorithms, it was observed that *AdaBoost.RT* surpasses them on most of the data sets considered. Qualitative and quantitative performance measures (scoring matrix) also give an indication of the higher accuracy of *AdaBoost.RT.* However, for more accurate statistically significant comparison, more experiments would be 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

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

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

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

Breiman, L. (1996b). Bagging Predictors. *Machine Learning*, 24 (2), 123-140.

Breiman, L. (1996c). Bias, variance, and arcing classifiers, *Technical Report 460*, Statistics Department, University of California,Berkeley, CA.

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

Cherkassky, V., & Ma, Y. (2004). Comparison of Loss Functions for Linear Regression. *Proc. of the International Joint Conference on Neural Networks* (pp. 395-400). Budapest, Hungary.

Drucker, H. (1997). Improving Regressor using Boosting Techniques. In Douglas

H. Fisher, Jr (Eds.), *Proc. of the 14th Intenational Conferences on Machine Learning* (pp. 107-115). Morgan Kaufmann.

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

Duffy, N., & Helmbold, D.P. (2000). Leveraging for regression. *Proc.of the 13th Annual Conference on Computational Learning Theory* (pp. 208-219). San Francisco: Morgan Kaufmann.

Efron, B., & Tibshirani, R. (1993). An Introduction to the Bootstrap. New York: Chapman and Hall.

Feely, R. (2000). Predicting Stock Market Volatility Using Neural Networks. *B.A (Mod) Dissertation.* Department of Computer Science, Trinity College Dublin.

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

Freund, Y., & Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55 (1), 119-139.

Friedman, J. (1991). Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19 (1), 1-82.

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

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

Kearns, M., & Vazirani, U.V. (1994). An Introduction to Computational Learning Theory. Cambridge, USA: MIT Press.

Namee, B.M., Cunningham, P, Byrne, S., & Corrigan, O.I. (2000). The problem of bias in training data in regression problems in mediacal decision support [online]. *Pádraig Cunningham's Online publications*, TCD-CS-2000-58. Available: http://www.cs.tcd.ie/Padraig.Cunningham/online-pubs.html.

NeuralMachine (Neural network software). Available: http://www.data-machine.com.

NeuroSolutions (Neural network software). Available: http://www.nd.com.

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

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

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

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

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.

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

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

Solomatine, D.P., & Shrestha, D.L. (2004). AdaBoost.RT: a Boosting Algorithm for Regression Problems. *Proc. of the International Joint Conference on Neural Networks* (pp. 1163-1168). Budapest, Hungary.

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

Valiant, L.G. (1984). A theory of the learnable. *Communications of the ACM*, 27 (11), 1134-1142.

Vapnik, V. (1995). The Nature of Statistical Learning Theorey. Springer, New York.

Weigend A.S., & Gershenfeld G. (1993). Time Series Prediction: Forecasting the Future and Understanding the Past. Proceedings of the NATO Advanced Research Workshop on Comparitive Time Seris Analysis, Sante Fe, New Mexico.

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

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

Appendix A. *AdaBoost.M1* algorithm

1. Input:
- Sequence of m examples $(x_1, y_1),\ldots,(x_m, y_m)$, where labels $y \in Y = \{1,\ldots, k\}$
- Weak learning algorithm W*eak Learner*
- Integer T specifying number of iterations (machines)

2. Initialize:
- Machine number or iteration $t = 1$
- Distribution $D_t(i) = 1/m \qquad \forall\, i$
- Error rate $\varepsilon_t = 0$

3. Iterate while error rate $\varepsilon_t < 0.5$ or $t \le T$
- Call *Weak Learner*, providing it with distribution D_t
- Get back a hypothesis $h_t : X \to Y$
- Calculate the error rate of h_t: $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$
- Set $\beta_t = \varepsilon_t /(1-\varepsilon_t)$
- Update distribution D_t as

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

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

Appendix B. *AdaBoost.R2* algorithm

1. Input:
- Sequence of m examples $(x_1, y_1),\ldots,(x_m, y_m)$, where output $y \in R$
- Weak learning algorithm *Weak Learner*
- Integer T specifying number of iterations (machines)

2. Initialize:
- Machine number or iteration $t = 1$
- Distribution $D_t(i) = 1/m \qquad \forall\, i$
- Average loss function $\overline{L_t} = 0$

3. Iterate while average loss function $\overline{L_t} < 0.5$ or $t \le T$
- Call *Weak Learner*, providing it with distribution D_t
- Build the regression model: $f_t(x) \rightarrow y$
- Calculate the loss for each training example as: $l_t(i) = \left| f_t(x_i) - y_i \right|$
- Calculate the loss function $L_t\ (i)$ for each training example using three different functional forms as:

  Linear: $L_t(i) = \dfrac{l_t(i)}{Denom_t}$; Square law: $L_t(i) = \left[\dfrac{l_t(i)}{Denom_t}\right]^2$

  Exponential: $L_t(i) = 1 - \exp\left[-\dfrac{l_t(i)}{Denom_t}\right]$

  where $Denom_t = \max\limits_{i=1\ldots m}(l_t(i))$
- Calculate an average loss as: $\overline{L_t} = \sum\limits_{i=1}^{m} L_t(i) D_t(i)$
- Set $\beta_t = \overline{L_t}/(1-\overline{L_t})$
- Update distribution D_t as: $D_{t+1}(i) = \dfrac{D_t(i)\beta_t^{(1-L_t(i))}}{Z_t}$

  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:

$$f_{fin}(x) = \inf\left[y \in Y : \sum_{t:\, f_t(x) \le y} \log\frac{1}{\beta_t} \ge \frac{1}{2}\sum_t \log\frac{1}{\beta_t} \right]$$

Appendix C. Boosting method of Avnimelech and Intrator (1999)

1. Spilt the training examples to three sets *Set 1*, *Set 2* and *Set 3* such a way that the *Set 1* should be smaller than the other two sets.
2. Train the first machine on *Set 1.*
3. Assign to the second machine all the examples from *Set 2* on which first machine has a *big error* and a similar number of examples from this *Set 2* on which it does not have a *big error* and train the second machine on it.
4. Assign the training examples to third machine according to the following different versions:

- *Boost1*: All examples on which exactly one of the first two machines has a *big error.*
- *Boost2*: Data set constructed for *Boost1* + all examples on which both machines have a *big error*, but these errors have different sign.
- *Modified Boost 2*: Data set constructed for *Boost2* + all examples on which both machines have a *big error*, but there is a "big" difference between the magnitude of errors.

5. Combine the outputs of the three machines using median as final prediction from the ensemble.

Appendix D. *BEM* boosting method (Feely, 2000)

1. Input:
- Sequence of m examples $(x_1, y_1),\ldots,(x_m, y_m)$, where output $y \in R$
- Weak learning algorithm *Weak Learner*
- Integer T specifying number of iterations
- *BEM* for demarcating correct and incorrect predictions

2. Initialize:
- Machine number or iteration $t = 1$
- Distribution $D_t(i) = 1/m \qquad \forall\, i$
- Error count $\text{errCount}_t = 0$

3. Iterate while $t \leq T$
- Call *Weak Learner*, providing it with distribution D_t
- Build the regression model: $f_t(x) \rightarrow y$
- Calculate absolute error $AE_t(i)$ for each training example
- Calculate the error count of $f_t(x)$: $errCount_t = \sum_{i:\, AE_t(i) > BEM} whole\ \ training\ \ data$
- Calculate the *Upfactor* and *Downfactor as:*

$$Upfactor_t = \frac{m}{errCount_t}$$

$$Downfactor_t = \frac{1}{Upfactor_t}$$

- Update distribution D_t as

$$D_{t+1}(i) = D_t(i)\begin{Bmatrix} Upfactor_t & if\ \ AE_t\ (i) > BEM \\ Downfactor_t & otherwise \end{Bmatrix}$$

- Sample new training data according to the distribution D_{t+1}
- Set $t = t + 1$

4. Combine outputs from individual machine

Table Captions

Table 1: Data sets summary. CV is cross validation data set.

Table 2: Comparison of performance (RMSE) of *AdaBoost.RT* on test data for the *Group 1* of data sets with and without cross validation. The results are averaged for 10 different runs (each run for each subset of data sets).

Table 3: Comparison of performance (RMSE) of different machines on test data. The results are averaged for 10 different runs. RT, R2, AI and BEM represent boosting algorithms *AdaBoost.RT*, *AdaBoost.R2,* method of Avnimelech and Intrator (1999) and *BEM* boosting methods respectively.

Table 4: Comparison of performance (RMSE) of *AdaBoost.R2* and *AdaBoost.RT* using different combining schemes in test data set for the *Group 1* of data sets. The results are averaged for 10 different runs.

Table 5: Qualitative scoring matrix for different machines.

Table 6: Quantitative scoring matrix for different machines.

Table 7: Comparison of performance (RMSE) of boosting algorithms on Laser data set (test data) using ANN as a weak learning machine. The results are averaged for 10 different runs.

Figures Captions

Figure 1: Block diagram of a committee machine. Flows of type A distribute data between the machines. Flows of type B appear when machines pass unclassified data subsets to the subsequent machines thus making a hierarchy.

Figure 2: Weight updating parameter (left y-axis) or weight of the machine (right y-axis) and error rate ε (*AdaBoost.RT*) or average loss Lta (*AdaBoost.R2*). *BetaR2* and *BetaRT* represent the weight updating parameters for *AdaBoost.R2* and *AdaBoost.RT* respectively. *WR2* and *WRT* represent the weights of the machine for *AdaBoost.R2* and *AdaBoost.RT* respectively.

Figure 3: Error rate and relative error threshold for selected data sets. Error rate for all data sets drops exponentially to relative error threshold.

Figure 4: Performance of M5 model trees against the numbers of linear models for one of the Boston housing data sets.

Figure 5: Performance of *AdaBoost.RT* against different number of machines for one of the Friedman#1 data sets.

Figure 6: Performance of *AdaBoost.RT* against different values of relative error threshold for one of the Friedman#1 data sets.

Figure 7: Performance and error rate of *AdaBoost.RT* against different numbers of machines for one of the Friedman#1 data sets. Although error rate is higher than 0.5 for machine 7, root mean squared error is still decreasing after machine 7.

Figure 8: Histogram of the output variable *y* in one of the Friedman#1 data sets.

Figure 9: Comparison of average absolute error using MT and *AdaBoost.RT* in different ranges for one of the Friedman#1 data sets. It is noticed that *AdaBoost.RT* outperforms MT on all ranges except one.

Figure 10: Histogram of absolute error using MT and *AdaBoost.RT* for one of the Friedman#1 test data sets. It is noticed that *AdaBoost.RT* does well on the hard examples as the frequency of larger absolute errors is reduced.

Table 1: Data sets summary. CV is cross validation data set.

	Data set	Training	CV	Test	Attributes	
					Continuous	Discrete
Group 1	SieveQ3	1554	300	300	7	0
	SieveQ6	1554	300	300	3	0
	Boston housing	401	80	25	13	1
Group 2	Auto-Mpg	262	-	136	5	3
	CPU	137	-	72	7	1
	Friedman#1	990	-	510	6	0
	Laser	7552	-	2518	17	0

Table 2: Comparison of performance (RMSE) of *AdaBoost.RT* on test data for the *Group 1* of data sets with and without cross validation. The results are averaged for 10 different runs (each run for each subset of data sets).

Data set	MT	*AdaBoost.RT*	
		With cross valid.	Without cross valid.
SieveQ3	14.66	**13.81**	15.36
SieveQ6	27.01	**26.37**	26.91
Boston housing	3.50	**3.23**	3.38

Table 3: Comparison of performance (RMSE) of different machines on test data. The results are averaged for 10 different runs. RT, R2, AI and BEM represent boosting algorithms *AdaBoost.RT*, *AdaBoost.R2,* method of Avnimelech and Intrator (1999) and *BEM* boosting methods respectively.

Data Set	MT	RT	R2	AI	BEM	Bagging	ANN
SieveQ3	14.67	**13.81**	15.01	14.14	22.3	13.93	17.09
SieveQ6	26.55	**26.37**	28.36	26.82	35.15	26.47	32.87
B. Housing	3.62	**3.23**	**3.23**	4.11	3.44	3.24	3.54
Auto-Mpg	2.97	2.92	2.91	3.09	3.00	**2.86**	3.79
CPU	34.65	26.52	24.45	46.7	23.2	32.64	**13.91**
Friedman #1	2.19	1.72	1.82	2.24	1.69	2.06	**1.51**
Laser	4.04	2.76	**2.69**	4.5	2.87	3.35	3.95

Table 4: Comparison of performance (RMSE) of *AdaBoost.R2* and *AdaBoost.RT* using different combining schemes in test data set for the *Group 1* of data sets. The results are averaged for 10 different runs.

Data set	Weighted average		Weighted median	
	AdaBoost.R2	*AdaBoost.RT*	*AdaBoost.R2*	*AdaBoost.RT*
SieveQ3	21.92	**13.81**	15.01	**13.73**
SieveQ6	32.15	**26.37**	28.36	**26.30**
Boston housing	3.57	**3.23**	**3.23**	3.39

Table 5: Qualitative scoring matrix for different machines.

Machine	MT	RT	R2	AI	BEM	Bagging	ANN	**Total**
MT	0	23	30	73	43	30	47	**41**
RT	77	0	60	90	67	60	59	**69**
R2	70	40	0	80	61	61	57	**62**
AI	27	10	20	0	34	16	43	**25**
BEM	57	33	39	66	0	46	44	**47**
Bagging	70	40	39	84	54	0	56	**57**
ANN	53	41	43	57	56	44	0	**49**
Total	**59**	**31**	**38**	**75**	**53**	**43**	**51**	0

Table 6: Quantitative scoring matrix for different machines.

Machine	MT	RT	R2	AI	BEM	Bagging	ANN	**Total**
MT	0.0	-13.7	-12.0	7.5	-4.3	-6.9	-5.8	**-35.2**
RT	13.7	0.0	1.4	19.5	8.7	7.5	5.9	**56.7**
R2	12.0	-1.4	0.0	17.6	7.9	5.9	4.2	**46.2**
AI	-7.5	-19.5	-17.6	0.0	-10.1	-13.7	-10.8	**-79.2**
BEM	4.3	-8.7	-7.9	10.1	0.0	-1.6	-4.2	**-8.0**
Bagging	6.9	-7.5	-5.9	13.7	1.6	0.0	0.3	**9.2**
ANN	5.8	-5.9	-4.2	10.8	4.2	-0.3	0.0	**10.3**

Table 7: Comparison of performance (RMSE) of boosting algorithms on Laser data set (test data) using ANN as a weak learning machine. The results are averaged for 10 different runs.

Boosting algorithm	RMSE	NMSE
ANN	4.07	0.008042
AdaBoost.RT	**2.48**	**0.002768**
AdaBoost.R2	2.84	0.003700
AI	3.13	0.004402
BEM	3.42	0.005336

FIGURES 1 – 10
(they are also provided without captions in a separate file)

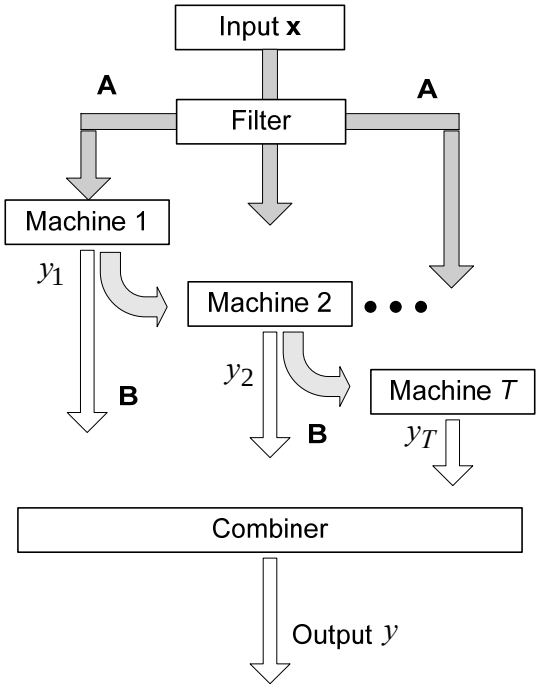

Figure 1: Block diagram of a committee machine. Flows of type A distribute data between the machines. Flows of type B appear when machines pass unclassified data subsets to the subsequent machines thus making a hierarchy.

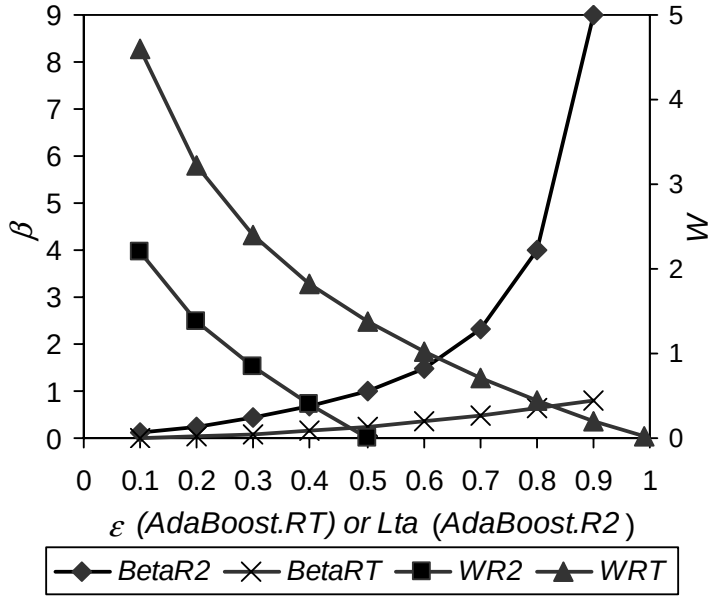

Figure 2: Weight updating parameter (left y-axis) or weight of the machine (right y-axis) and error rate ε (*AdaBoost.RT*) or average loss *Lta* (*AdaBoost.R2*). *BetaR2* and *BetaRT* represent the weight updating parameters for *AdaBoost.R2* and *AdaBoost.RT* respectively. *WR2* and *WRT* represent the weights of the machine for *AdaBoost.R2* and *AdaBoost.RT* respectively.

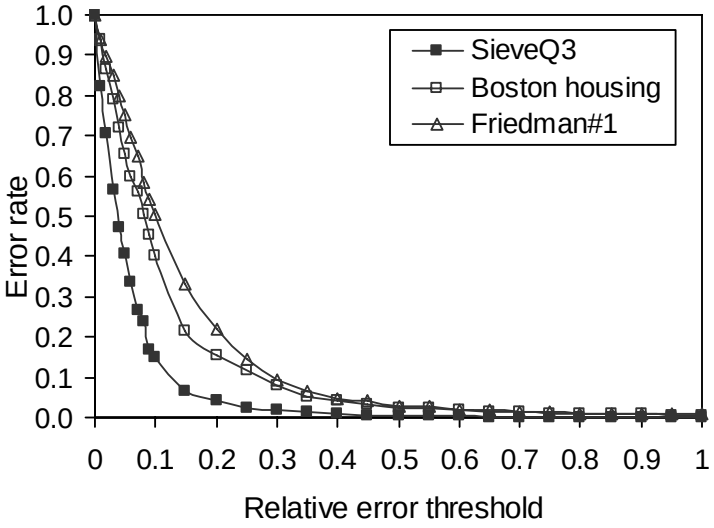

Figure 3: Error rate and relative error threshold for selected data sets. Error rate for all data sets drops exponentially to relative error threshold.

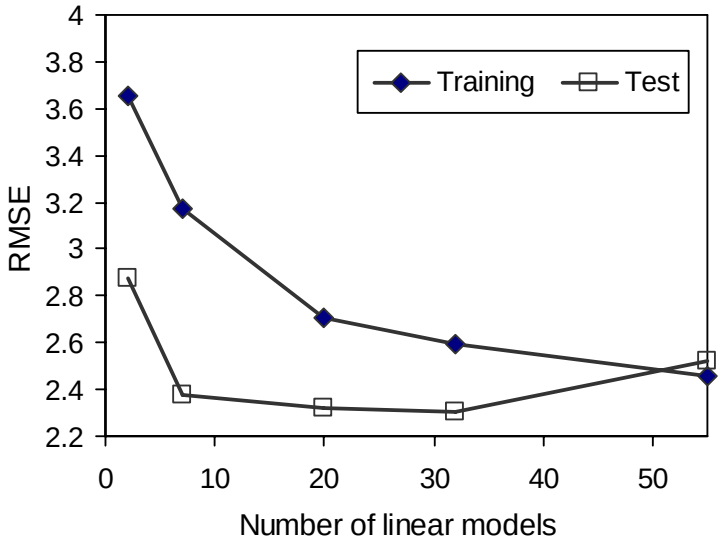

Figure 4: Performance of M5 model trees against the numbers of linear models for one of the Boston housing data sets.

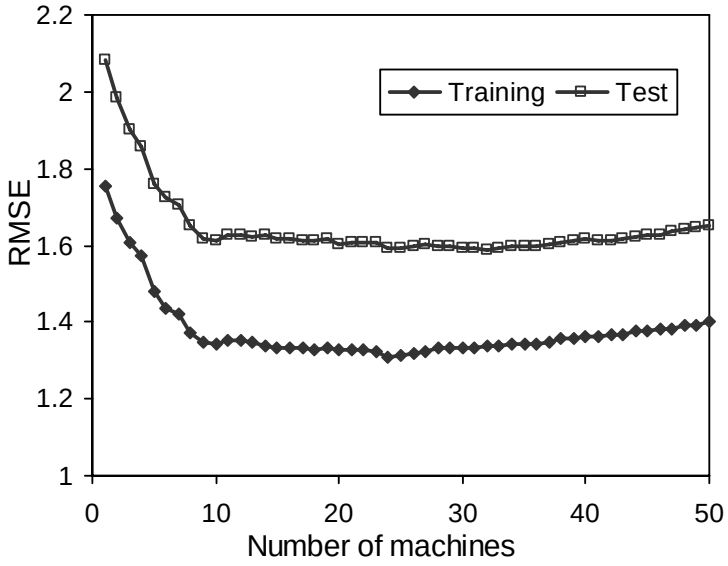

Figure 5: Performance of *AdaBoost.RT* against different number of machines for one of the Friedman#1 data sets.

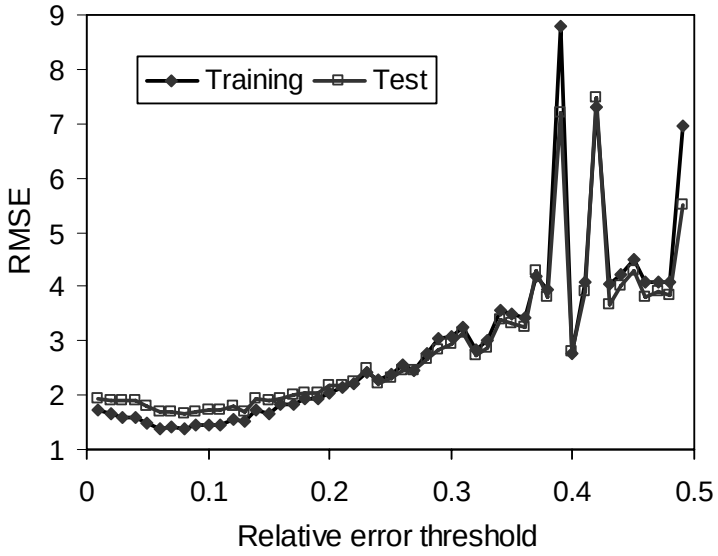

Figure 6: Performance of *AdaBoost.RT* against different values of relative error threshold for one of the Friedman#1 data sets.

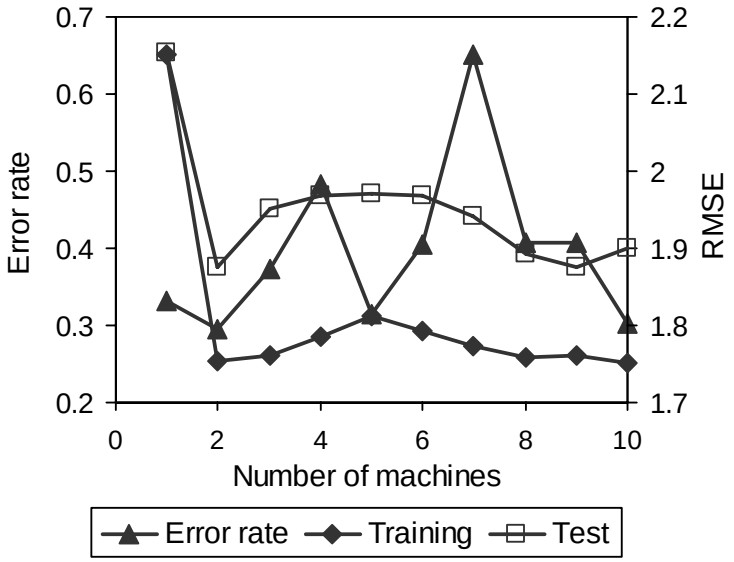

Figure 7: Performance and error rate of *AdaBoost.RT* against different numbers of machines for one of the Friedman#1 data sets. Although error rate is higher than 0.5 for machine 7, root mean squared error is still decreasing after machine 7.

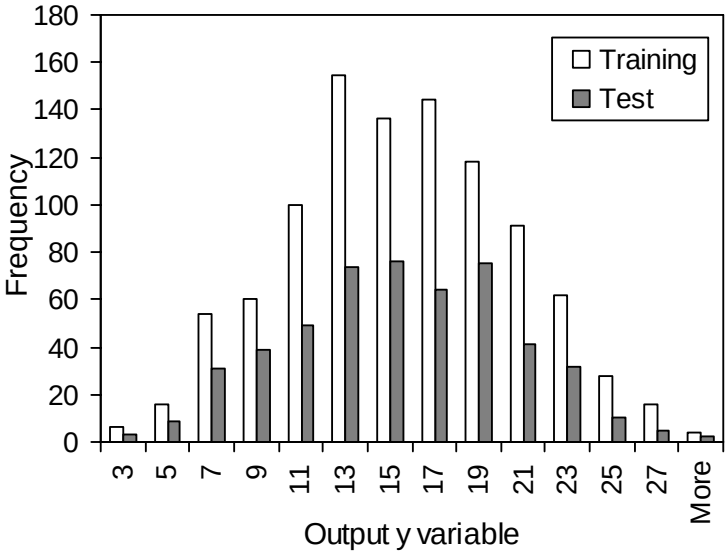

Figure 8: Histogram of the output variable *y* in one of the Friedman#1 data sets.

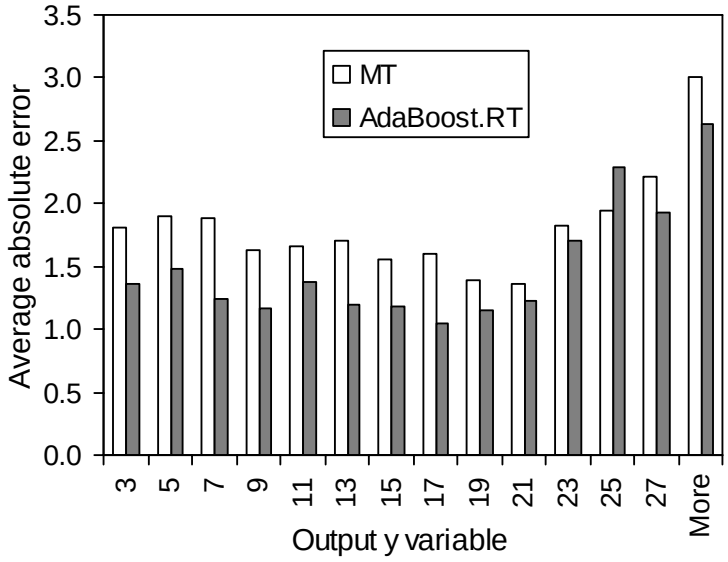

Figure 9: Comparison of average absolute error using MT and *AdaBoost.RT* in different ranges for one of the Friedman#1 data sets. It is noticed that *AdaBoost.RT* outperforms MT on all ranges except one.

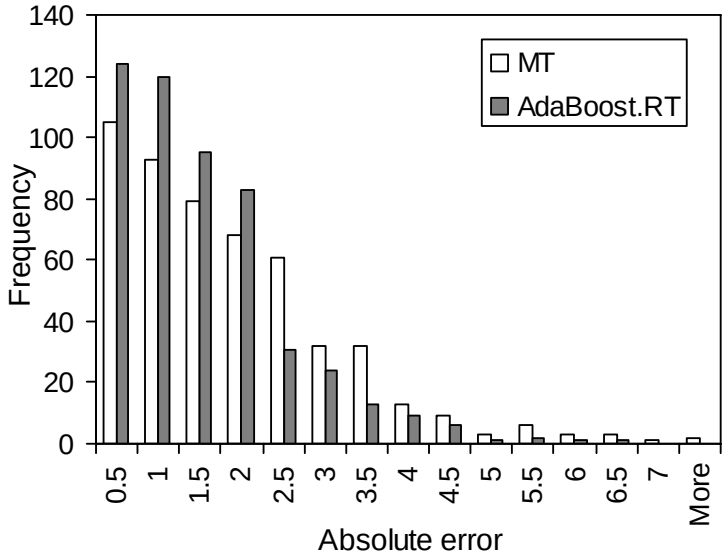

Figure 10: Histogram of absolute error using MT and *AdaBoost.RT* for one of the Friedman#1 test data sets. It is noticed that *AdaBoost.RT* does well on the hard examples as the frequency of larger absolute errors is reduced.