ORIGINAL RESEARCH

Adaboost.MRT: Boosting regression for multivariate estimation

Nikolai Kummer [*], Homayoun Najjaran

School of Engineering, University of British Columbia, Canada

Received: August 1, 2014 **Accepted:** October 20, 2014 **Online Published:** October 29, 2014
DOI: 10.5430/air.v3n4p64 **URL:** http://dx.doi.org/10.5430/air.v3n4p64

Abstract

Adaboost.RT is a well-known extension of Adaboost to regression problems, which achieves increased accuracy by iterative training of weak learners on different subsets of data. At each iteration, the prediction error is compared against a threshold, which is used to increase or decrease the weight of the sample for the next iteration. Adaboost.RT is susceptible to noise and contains a singularity in its misclassification function, which results in reduced accuracy for output values near zero. We propose Adaboost.MRT, which extends Adaboost.RT to multivariate output, addresses the singularity in the misclassification function and reduces noise sensitivity. A singularity-free, variance-scaled misclassification function is proposed that generates diversity in the training sets. Adaboost.MRT boosts multivariate regression by assigning each output variable a weight for each sample in the training data. To avoid fitting to outliers, the sampling weights for the training sets are averaged across all output variables. The threshold parameter is extended to accommodate the multivariate output and experiments suggest that for small amounts of output variables, the threshold can be tuned for each output variable individually. Comparisons on six singlevariate output datasets show that the proposed Adaboost.MRT outperforms Adaboost.RT on datasets with values near zero or with large noise and displays a similar accuracy otherwise. Experiments with three multivariate output datasets show that Adaboost.MRT performs similar or better than bagging and a simple averaging ensemble.

Key Words: Adaboost, Regression, Multivariate, Ensemble learning, Artificial neural networks, Boosting

1 Introduction

Ensemble prediction and learning is a widely used technique in machine learning and artificial intelligence to improve prediction accuracy through the combination of multiple predictors/learners.[1] Ensembles display better classification accuracy than each individual predictor if the ensemble consists of diverse predictors.[2] Predictors are said to be diverse if they have different error characteristics (e.g., mean, variance) on data points.[3] The rationale behind the use of ensembles acknowledges that it may be hard to find an individual optimal learner and may be more practical to select a combination of sub-optimal or "weaker" learners. While the exact performance increase is difficult to determine, ensembles perform better than the ensemble's weakest learner. Published work reports increased accuracy of ensembles over single classifiers.[2,4,5]

Bagging (bootstrap aggregating) and boosting are two ensemble methods that use multiple predictors to improve the accuracy for regression and classification problems. The size of the available training data is increased and diversified by repeated sampling, with replacement, from the original dataset. In bagging, the samples are chosen with equal weights. In boosting sampling occurs sequentially with weights that are updated based on the accuracy of pre-

[*]**Correspondence:** Nikolai Kummer; Email: nikolai.kummer@ubc.ca; Address: School of Engineering, University of British Columbia, Kelowna, BC.,V1V 1V7, Canada

vious predictions. Boosting methods are generally learned in series, whereas bagging can be readily parallelized for faster learning. These methods work best on unstable learners such as the neural network or decision trees technique. There is some evidence suggesting that boosting has increased accuracy over bagging.[5,6] However in Ref.5 and Ref.7, the authors present examples of boosting's performance degradation in the presence of noise. The sequential sampling of boosting techniques allows the ensemble to focus on harder examples, but may result in over-fitting. For an overview on boosting methods, see Ref.8.

1.1 Adaboost methods

Adaboost, one of the first practical boosting methods, was introduced by Freund and Schapire.[9,10] In Adaboost, weak learners are combined, "boosted", to improve ensemble accuracy and produce a "strong" classifier. In classification, weak learners exhibit only a small correlation between the prediction and the true value. Specifically, a weak learner has prediction accuracy of just over 50%, which is only slightly better than random guessing. Weak learners are used in ensembles, because a complex function can be described by many general trends, which can result in a strong approximation of that function. It is also often difficult to select an individual optimal learner and boosting simplifies that selection.

In Adaboost, the training data is iteratively sampled, with replacement, to train the weak learner. The predictive performance of the weak learner hypothesis is evaluated and the sampling distribution weights are updated, giving more weight to the misclassified examples. The next learner samples from the updated distribution and the learning process is repeated. Training set diversity is introduced by training the next weak learner on the previously misclassified, or "hard", examples. The final output of Adaboost, is a weighted combination of all hypotheses, with the weights corresponding to each individual hypothesis' prediction error.

Adaboost featured some practical advantages: its low implementation complexity and the introduction of only a single tuning parameter, the number of iterations T. However, the accuracy that can be expected is dependent on the base learning algorithm and the available data. Boosting is susceptible to noise and Adaboost will fail if there is insufficient data, or if the prediction hypothesis is too weak.[10]

Following the introduction of Adaboost, the research effort shifted to the extension of the algorithm to accommodate a wider set of problems and to address its short-comings. Adaboost focused on binary classification but has been extended to multi-class classification (Adaboost.M1, Adaboost.M2[11]). In Ref.12, Sun et al. present cost-sensitive modifications that handle class-imbalances in pattern recognition, which are data with significant differences in prior probabilities.

Boosting is prone to over-fitting and is therefore noise and outlier sensitive.[7,13,14] Strategies that address this noise sensitivity generally fall into two major categories:[13] (1) revision of the weight updating mechanism to correspond to a modified loss function; (2) filtering of outliers or modification of the weights that are assigned to noisy data in the final hypothesis. Following the first strategy, ND-Adaboost[13] was introduced, which addresses boosting's noise sensitivity and improves performance by a noise detection mechanism. Along the second strategy is the AveBoost algorithm,[15,16] in which the weighting distributions of the current and previous iteration are averaged. This averaging reduces the impact of outliers on the final hypothesis by limiting the weight that these outliers can be assigned. The reduction of the sampling weight of outliers reduced the likelihood that learners will pick these values for the following iterations and thus reduced the chance of fitting to outliers.

1.2 Adaboost for regression

The successful application of Adaboost for classification problems led to the extension of Adaboost to regression problems. Adaboost's modular nature allows for improvements of the regressors and for the adaptability to specific problems.[8] For an overview of regression boosting methods see.[17]

Adaboost.RT (where RT is an abbreviation for regression threshold) is based on Adaboost.M1, which was modified for regression and was introduced by Solomatine and Shrestha.[18,19] Adaboost.RT creates a sample from the available training data using the sampling weight vector $\boldsymbol{D_t}$. The base learning algorithm uses this sample to create a hypothesis that links the input to the output data. This hypothesis is applied to all the data to create predictions. The absolute relative error is calculated for each prediction and compared against a threshold ϕ. The absolute relative error is the percent error of the prediction compared to the true value. The threshold ϕ is used to classify the predicted values as correct or incorrect. This threshold comparison modifies the regression task to a classification problem. The sampling weight of incorrectly predicted samples is increased for the next iteration. Adaboost.RT displayed better performance than a single artificial neural network (ANN) at the cost of the introduction of a new tuning parameter ϕ. Adaboost.RT did not have an early stopping criterion like similar regression boosting methods such as Adaboost.R2 and the number of iterations can be chosen depending on the required accuracy (up to a limit). Follow-up work compared performance to Adaboost.R2, bagging, and artificial neural networks among others and reported better results with Adaboost.RT.[19] Performance comparisons of Adaboost.RT with various other algorithms can be found in Ref.19 and Ref.20.

Unlike other Adaboost-based regression techniques, Adaboost.RT requires the selection of an additional tuning parameter ϕ.[19]The parameter ϕ affects the accuracy of the ensemble, as it influences the sampling weights of the instances and thereby modifies the diversity of the training data. A small threshold will classify the majority of predicted values as incorrect and result in a near uniform sampling distribution, which will resemble bagging. A threshold that is set too large will result in fitting to extreme outliers. An adaptive work-around for the problem has been proposed in Ref.21 and 22, by recursive adjustment of ϕ. However, the equations presented in Ref.22 enable the value of ϕ to become negative, which is physically meaningless and force the algorithm to fail. From Ref.22, the value of ϕ at iteration t+1 is

$$\begin{cases}\phi_{t+1} = \phi_t(1-\lambda), while\ e_t < e_{t-1} \\ \phi_{t+1} = \phi_t(1+\lambda), while\ e_t > e_{t-1}\end{cases} \tag{1}$$

where e_t is the root mean squared error (RMSE) at iteration t and by definition $e_t \geq 0$. The value λ is

$$\lambda = r\left|\frac{e_t - e_{t-1}}{e_t}\right| \tag{2}$$

where r is a tuning constant. The term 1-λ will be negative, leading to a negative ϕ, if $e_{t-1} > 2e_t$ (for r=1). This problem can be managed by selecting the value of r sufficiently small to let $\lambda < 1$. The reduction of the magnitude of r also limits the algorithm's ability to change and adapt the value of ϕ. The method shifts the tuning parameter from ϕ to r. The adaptive method will however not fail as long as

$$\frac{e_t}{r} > |e_t - e_{t-1}| \tag{3}$$

and as long as $e_t \neq 0$.

An additional limitation of Adaboost.RT is the definition of the misclassification function $Er_t(i)$ as the *absolute relative error*. This error function is a direct implementation of the Adaboost.M1 algorithm and classifies a sample prediction based on the percent error of the prediction $f_t(x_i)$ compared to the true value y_i. The absolute relative error is calculated at each iteration and is defined as

$$Er_t(i) = ARE_t(i) = \left|\frac{f_t(x_i) - y_i}{y_i}\right| \tag{4}$$

This definition of the error function contains a singularity when the true value y_i=0.

Values near zero receive an artificially high absolute relative error and are always classified as incorrect and their sampling weight increases. Values near zero will be selected more frequently for training and the accuracy of the values away from the zero-crossing will decrease. The performance of the entire ensemble is degraded when the output variable contains a zero-crossing in the output data. This issue is illustrated in Figure 1, with training data sampled from the noise-less function $y = x$. Despite the lack of noise, the overall accuracy of the algorithm is degraded. Training instances near y=0 are repeatedly sampled and identified to a high accuracy, but the values away from the zero crossing, experience a very high relative error.

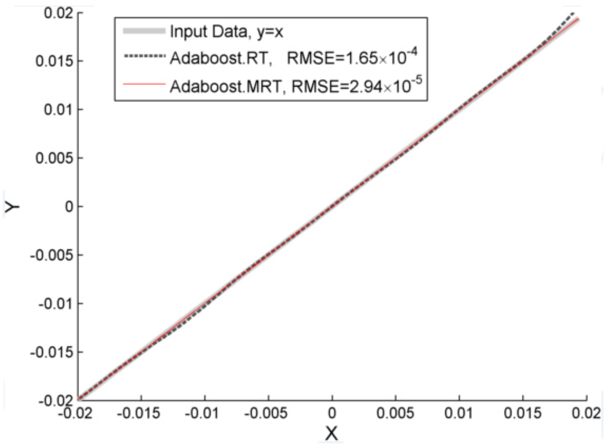


Figure 1: Performance comparison of Adaboost.RT and Adaboost.MRT for estimation of the noise-less function $y = x$. The accuracy of Adaboost.RT is negatively affected by the singularity in the misclassification function.

Some possible remedies include a data transformation, by the addition of a constant to the output data[19] or the use of the absolute error misclassification function defined by

$$Er_t(i) = |f_t(x_i) - y_i| \tag{5}$$

The absolute error function, makes the determination of a threshold ϕ difficult, since it is directly linked to the numerical value of the output data. Absolute error-based misclassification functions attempt to increase accuracy at each sample. Given noisy training data the general shape of the function is more important than the accuracy at each sample to avoid over-fitting to noise.

Despite its short-comings, Adaboost.RT is shown in literature to provide good results, with a few papers detailing applications to real problems: in Ref.21 and 22 for molten steel temperature prediction, in Ref.23 for music emotion classification with a regression approach, in Ref.24 for software reliability modelling and in Ref.25 for intensity estimation of facial action units. These methods were used to estimate a single output variable only. To the best of the author's knowledge, the Adaboost algorithm has not been used for multivariate output regression.

1.3 Contributions

This paper presents Adaboost.MRT (Adaboost Multivariate Regression Threshold), a multivariate regression algorithm which is based on Adaboost.RT. Adaboost.MRT takes in a set of labelled training instances $S = \{(\boldsymbol{x_n}, \boldsymbol{y_n}), n = 1, 2, \cdots, m\}$, where $\boldsymbol{y_n}$ is an R-dimensional vector of output data and $\boldsymbol{x_n}$ is a multi-dimensional vector of input data. The regression model $f_t(x)$ is built over T learning iterations, with training data generated by iterative sampling from the m available instances.

The first major contribution is the improvement of the misclassification function of Adaboost.RT. The singularity in the absolute relative error function of the original algorithm potentially degrades the accuracy of the ensemble. In the proposed formulation, a variance-scaled error function is selected that allows for zero-crossings in the output data without the need for output data transformation. The variance-scaled error function always classifies a portion of the predictions as incorrect at each iteration, which also avoids fitting to outliers. As part of a variance-scaled misclassification function it was also necessary to change the output of the ensemble from a weighted output as, in Adaboost.RT, to a simple average output.

The second major contribution is the extension of Adaboost.RT to accommodate multivariate output. The proposed method modifies the sampling distribution into $\boldsymbol{D_{y,t}}$ an $R \times m$ vector which maintains the weight distributions of all R output variables for each of the m instances. Every entry in $\boldsymbol{D_{y,t}}$ tracks how well each of the R output variables have been predicted by the previous t learners. This new distribution is called the output error distribution and is a measure of the prediction error at each output variable. At each iteration, the instance sampling weight $\boldsymbol{D_t}$ is the average of the weights of the R output variables within the instance. Similar to AveBoost,[15, 16] this averaging reduces the impact that an outlier within the R output variables has on the weight of the training instance. This strategy is aimed at reducing the algorithm's noise and outlier sensitivity when dealing with multivariate output data.

Adaboost.MRT addresses Adaboost's noise and outlier sensitivity in two ways: (i) by the redefinition of the misclassification function (Equation (6)) and by (ii) averaging the sampling distribution of the output variables for multivariate output (Equation (10)). The variance-scaled misclassification function classifies the least accurate samples as incorrect at each iteration based on ϕ. The misclassification of the least correct predictions generates training set diversity in successive training iterations. The averaging of the sampling distribution across the output variables, for multivariate output, prevents a single outlier in $\boldsymbol{y}$ from overly increasing the value of a particular instance. This averaging limits the impact of outliers in one output variable on the entire sample distribution.

Adaboost.MRT is tested and validated against Adaboost.RT and their base learning algorithm, the artificial neural network, on six commonly used singlevariate output datasets. Performance comparisons for Adaboost.RT against other methods can be found in Ref.18-20. Thus, the singlevariate experiments in this paper only focus on comparing the performance of Adaboost.MRT against Adaboost.RT. The Friedman #1, #2, and #3 data sets are commonly used for regression performance evaluation. The Gabor and the $sinc|x|$ function were used due to their zero-crossing in the output data. The yacht hydrodynamics dataset from the UCI Machine Learning database was used to assess the accuracy on real data. The noise sensitivity of Adaboost.RT and Adaboost.MRT was compared in an experiment that varied the noise-to-signal magnitude ratio for the Friedman #3 dataset and the $sinc|x|$ function.

For multivariate output, Adaboost.MRT is tested and validated against bagging, the simple averaging of 10 neural networks trained under similar conditions, and a single iteration of the base learning algorithm. The bagging and the simple averaging method contain 10 neural networks and were included to test whether Adaboost.MRT improves the way that each learner is trained. The performance comparison was done on three multivariate output datasets.

Following the introduction, the Adaboost.MRT algorithm is presented in Section 2. Experiments with singlevariate output are presented in Section 3. Multivariate output experiments are presented in Section 4 and concluding remarks are found in Section 5.

2 Adabost.MRT algorithm

The proposed Adaboost.MRT algorithm is capable of estimating vector outputs $\boldsymbol{y}$ from vector inputs $\boldsymbol{x}$. There are three main phases: (i) the *initialization phase*, (ii) the training phase, and (iii) the output phase.

During the initialization phase the following parameters have to be selected: a base learning algorithm, the number of iterations T, and the threshold parameter ϕ. The base learning algorithm is the method that will create a hypothesis $f(\boldsymbol{x}) \rightarrow \boldsymbol{y}$. Previous work used M5 model trees or neural networks as the base learner.[19] Adaboost.MRT boosts the accuracy of weak learners, by training multiple of these consecutively and providing each with a different portion of all the available training data. The number of iterations T has to be selected during the initialization phase. Previous work on Adaboost.RT (which the proposed algorithm is based on) shows that there is limited accuracy decrease beyond T=10 iterations.[19] The threshold ϕ is used to increase the sampling weight of incorrectly predicted samples (explained in Section 2.2). The value of ϕ affects the accuracy of the entire ensemble. A section of the available training data should be used to train Adaboost.MRT for different values of ϕ. The value of ϕ that gives the lowest error on the small sample, should be used to generate a final hypothesis for your data (see Section 3.2 for scalar output and Section 4.3 for vector output).

During the training phase T different learners are trained. Each learner is passed a small subset of the entire data and generates a hypothesis $f_t(\boldsymbol{x}) \rightarrow \boldsymbol{y}$. This hypothesis is then used on the entire training data to evaluate how well the learner predicts the data. Each correctly predicted sample (based on ϕ) has its sampling weight decreased and each incorrectly predicted sample has its sampling weight increased, making it more likely to be selected for the next iteration.

The *output phase* is used once an entire ensemble has been trained. The output of the ensemble is an averaged output of the T learners.

An overview of the Adaboost.MRT procedure is given in Algorithm #1. The initialization phase is detailed in Section 2.1. Section 2.2 describes the steps that occur at each iteration during learning and Section 2.3 describes the way that Adaboost.MRT outputs predictions.

2.1 Initialization phase

Adaboost.MRT is passed *m* training instances and each instance consist of vector input data $\boldsymbol{x}$ and vector output data $\boldsymbol{y} = (y_1, y_2, \cdots, y_R)$, where R is the number of output variables in the vector y and r denotes the index of the output variable $(r = 1 \cdots R)$ within $\boldsymbol{y}$. The number of iterations T is also defined, where t denotes the index of each learner $(t = 1 \cdots T)$.

The weights in $\boldsymbol{D_t}$ (an $m \times 1$ vector) are initialized as a uniform distribution and correspond to the sampling weight of each instance in S, which describes an instances likelihood to be selected for training. This distribution $\boldsymbol{D_t}$ is updated at each iteration to assign more weight to the misclassified predictions. Hard examples in the training data will be selected more frequently than samples that are accurately predicted, which generates predictors in the ensemble that are specialized on harder examples.

The output error distribution $\boldsymbol{D_{y,t}}$ is an $R \times m$ vector, which contains the prediction accuracy for each output of the R output variables for all m training instances. The output error distribution is an extension of the Adaboost.RT algorithm to the multivariate output case and is initialized as $1/m$. Prior to training, a threshold parameter vector $\phi = [\phi_1 \cdots \phi_R]$ is initialized. These thresholds are used to classify the predictions as correctly or incorrectly predicted. The optimal threshold value ϕ_{opt} that results in the lowest RMSE for a given dataset is selected from experiments in which the value of ϕ is varied (see Sections 3.2 and 4.3).

2.2 Iteration phase

A regression model $f_t(\boldsymbol{x}) \rightarrow \boldsymbol{y}$ is created at each iteration from N training examples (where $N \leq m$). The N samples are randomly selected, with replacement, from the original m training instances using the sampling weights $\boldsymbol{D_t}$. Let i denote the index of the instance with $i = 1, 2, \cdots, m$. The regression model is used on all m instances to predict the output $\boldsymbol{y}$. This prediction is used in the variance-scaled error function

$$Er_t^{(r)}(i) = \frac{|f_t(x_i) - y_{r,i}|}{\sigma_t^{(r)}} \tag{6}$$

Equation (6) calculates the absolute error of the prediction and divides it by the standard deviation of the absolute error for all m predictions $\sigma_t^{(r)}$ of an output variable. The resulting quantity is a measure of prediction error for instance i of output variable r, compared to all predictions at this iteration. Equation 6 is defined for all $\sigma_t^{(r)} > 0$, which is true for most predictions that have error.

The error function output is an $R \times m$ dimensional vector and is compared to the threshold ϕ. The weights in $\boldsymbol{D_{t,r}}$ of the misclassified samples are added to create the misclassification error rate $\varepsilon_t^{(r)}$, an $R \times 1$ dimensional vector. The misclassification error rate is a measure of how many "hard" examples the regression model misclassified. In the context of a variance-scaled $Er_t(r)$, "hard" examples are not necessarily outliers, they are examples that previous regression model iterations have classified as incorrect. A large value means that the regression model classified a lot of the “hard” examples as incorrect. The error rate is used in

$$\beta_{t,r} = (\varepsilon_t^{(r)})^n \tag{7}$$

to calculate the weight updating parameter $\beta_{t,r}$ using the power coefficient n. The power coefficient governs the sampling weight increase for incorrect predictions for the next iteration. In Ref.19 a power coefficient of n=2 and n=3 was suggested, but the authors warn that a large power coefficient may result in fitting to specific examples and may fail to generalize. If n is set too large, the algorithm will fit to outliers in a manner similar to Adaboost’s zero-crossing problem (see Section 1.2).

The weight updating parameter $\beta_{t,r}$ updates the weights in the output error distribution in $\boldsymbol{D_{y,t+1}^{(r)}(i)}$ by

$$\boldsymbol{D_{y,t+1}}^{(r)}(i) = \frac{\boldsymbol{D_{y,t}^{(r)}}(i)}{Z_t} \times \begin{cases} \beta_{t,r}, & if\ Er_t^{(r)}(i) \leq \phi_r \\ 1, & otherwise \end{cases} \tag{8}$$

The weights of the correctly classified examples in Equation (8) are multiplied by $\beta_{t,r}$. If the current iteration misclassified a lot examples (i.e. little additional diversity or bad prediction), $\beta_{t,r}$ will be close to 1 and the weights in $\boldsymbol{D_{t,r}}$ will remain nearly unchanged. Conversely, if the prediction is good, the weights of correctly classified examples are reduced. The sampling weight distribution $\boldsymbol{D_t}$ for the next iteration is updated as the average of the output error distribution $\boldsymbol{D_{y,t}}$ for each instance in m given by

$$\boldsymbol{D_{t+1}}(i) = \frac{1}{R}\sum_{k=1}^{r} \boldsymbol{D_{y,t}}^{(r=k)}(i) \tag{9}$$

Algorithm 1 Generalized Procedure for the Adaboost Multivariate Regression Threshold (Adaboost.MRT) algorithm, capable of estimation of R sized output vectors.

```
1. Input:
    • Sequence of $m$ examples $< \boldsymbol{x_1}, \boldsymbol{y_1} > \cdots < \boldsymbol{x_m}, \boldsymbol{y_m} >$ where the output $\boldsymbol{y} = (y_1, y_2, \ldots, y_R) \in \mathbb{R}$
    • Weak learning algorithm (``Weak Learner")
    • Integer $T$ specifying number of iterations or machines
    • Threshold vector $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_R), (0 < \boldsymbol{\phi})$ used to classify samples as correct and incorrect
2. Initialize:
    • Machine number or iteration, $t = 1$
    • Set the sampling weight distribution $\boldsymbol{D_t}(i) = 1/m$ for all $i$
    • Set the output error distribution $\boldsymbol{D_{y,t}}^{(r)}(i) = 1/m$ for all $i$
    • Set misclassification error rate vector of size $R$ to $\varepsilon_t^r = 0$
3. Iterate while t ≤ T:
    • Sample $N$ examples from sequence of $m$, using weights $D_t$ with replacement (where $N \leq m$)
    • Give these $N$ examples to the Weak Learner
    • Build the regression model $f_t(\boldsymbol{x}) \to \boldsymbol{y}$
    • Calculate the error function on $all$ $m$ examples for each output variable:
          $$Er_t^{(r)}(i) = \frac{|f_t(x_i) - y_{r,i}|}{\sigma_t^{(r)}}$$
      where $\sigma_t^{(r)}$ is the sample standard deviation of $(f_t(x_i) - y_{r,i})$
    • Calculate the misclassification error rate for every output of $f_t(x)$:
          $$\varepsilon_t^{(r)} = \sum_{i:Er_t^{(r)}(i) > \phi_r} \boldsymbol{D_{y,t}^{(r)}}(i)$$
    • Set the weight updating parameter
          $$\beta_{t,r} = \left(\varepsilon_t^{(r)}\right)^n$$
      where $n$= power coefficient (e.g. , linear, square or cubic)
    • Update the output error distribution $D_{y,t}$ as:
          $$\boldsymbol{D_{y,t+1}}^{(r)}(i) = \frac{\boldsymbol{D_{y,t}}^{(r)}(i)}{Z_t} \times \begin{cases} \beta_{t,r} & , if\ Er_t^{(r)}(i) \leq \phi_r \\ 1 & , otherwise \end{cases}$$
      where $Z_t$ is a normalization factor such that $\boldsymbol{D_{y,t+1}}^{(r)}$ will be a distribution.
    • Update the sample weight distribution as the average of the error rate of each output variable $r$ for each $i$:
          $$D_{t+1}(i) = \frac{1}{R}\sum_{k=1}^{r} D_{y,t}^{(r=k)}(i)$$
    • Set $t = t + 1$.

4. Output the final hypothesis:
          $$\boldsymbol{y}(\boldsymbol{x}) = \frac{\sum_t f_t(\boldsymbol{x})}{T}$$
```

2.3 Output phase

The final hypothesis $\boldsymbol{y}$ of Adaboost.MRT is the averaged output of all the T learners.

$$y(\boldsymbol{x}) = \frac{\sum_t f_t(\boldsymbol{x})}{T} \tag{10}$$

During learning, T different learners are trained and Adaboost.MRT diversifies the training data to created diverse learners. Adaboost.RT outputs a weighted sum, based on the accuracy of each learner, which may however lead to fitting to noise in noisy data sets. Previous iterations of the Adaboost.MRT algorithm considered the use of the weight updating parameter $\beta_{t,r}$, to scale the output. The parameter $\beta_{t,r}$ is derived from a variance-scaled quantity and is thus not a good predictor of accuracy. The increase in accuracy for a simple averaging output can be attributed to an increase of diversity in the learners.

3 Performance evaluation for scalar output datasets

3.1 Datasets

The performance of Adaboost.MRT was tested on six datasets and compared to that of Adaboost.RT, a single iteration of their weak learner i.e., the artificial neural network (ANN), and a simple averaging ensemble. The averaging ensemble consisted of 10 neural networks that were passed N randomly selected sampled. The output of the averaging ensemble was an average of all 10 neural networks. This type of ensemble was included to demonstrate the creation of diverse learners by Adaboost.MRT and to showcase that it is more accurate than simply T different learners.

Six datasets (five of which appeared in Ref.20 and 26) were used for performance comparison. Each dataset consists of vector inputs and scalar outputs. The six datasets are the Friedman #1, #2, and #3 datasets, the Gabor function, the $sinc|x|$ function, and the yacht hydrodynamics dataset from the UCI Machine Learning Repository.[27] These test sets are commonly used for performance evaluation in regression literature. The Friedman #1, #2 and #3 have been used in boosting regression in Ref. 6,19,20,26,28,29.

The Friedman #1 dataset has 10 input variables, 5 of which do not contribute to the output. The Friedman #2 and #3 test set model the impedance and phase shift in an alternating circuit.[29] The Gabor and the $sinc|x|$ function have been selected for the zero-crossing in the output data. The yacht hydrodynamics dataset was selected for function estimation of real measured data. The equations and parameters for the test sets are shown in Table 1. Uniformly distributed noise ε was added to the output variable y in the training data. The noise magnitude parameters are shown in Table 2. The RMSE was calculated against the noiseless function.

Table 1: Dataset equations used in the performance evaluation

Dataset	Function	Input Variables	Size, m
Friedman #1	$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5$	$x_i \sim \boldsymbol{U}[0,1]$, $i = 1 \dots 10$	5000
Friedman #2	$y = \sqrt{x_1^2 + \left(x_2 x_3 - \left(\frac{1}{x_2 x_4}\right)\right)^2}$	$x_1 \sim \boldsymbol{U}[0,100]$ $x_2 \sim \boldsymbol{U}[40\pi, 560\pi]$ $x_3 \sim \boldsymbol{U}[0,1]$ $x_4 \sim \boldsymbol{U}[1,11]$	5000
Friedman #3	$y = tan^{-1}\left(\frac{x_2 x_3 - \frac{1}{x_2 x_4}}{x_1}\right)$	$x_1 \sim \boldsymbol{U}[0,100]$ $x_2 \sim \boldsymbol{U}[40\pi, 560\pi]$ $x_3 \sim \boldsymbol{U}[0,1]$ $x_4 \sim \boldsymbol{U}[1,11]$	3000
Gabor	$y = \frac{\pi}{2} exp[-2(x_1^2 + x_2^2)] \cos[2\pi(x_1 + x_2)]$	$x_1 \sim \boldsymbol{U}[0,1]$ $x_2 \sim \boldsymbol{U}[0,1]$	3000
$sinc\|x\|$	$y = sinc\|x\| = \frac{\sin\|x\|}{\|x\|}$	$x \sim \boldsymbol{U}[-2\pi, 2\pi]$	5000
Yacht	Yacht Hydrodynamics Dataset	$x_i, i = 1 \dots 6$	308

The neural networks are trained by the back-propagation algorithm in a MATLAB implementation. The neural network consisted of one input, one hidden, and one output layer with the number of hidden nodes H shown in Table 2 for each dataset. The architecture of the neural network was not optimized for accuracy, as the relative performance of the tested algorithms was more important. The power coefficient n=1 was used in the Adaboost.MRT and Adaboost.RT algorithms to reduce the probability of fitting to outliers.

3.2 Optimization of the threshold parameter φ

The threshold parameter ϕ affects the accuracy for Adaboost.MRT and Adaboost.RT. An experimental search for an optimum value of threshold ϕ is presented in this section. The optimum value ideally gives the lowest error. For clarity, the thresholds will be referred to as ϕ_{RT} and ϕ_{MRT} for Adaboost.RT and Adaboost.MRT, respectively.

The threshold parameter search used a k-fold cross-validation with k=10. In a k-fold cross-validation the data is partitioned into k equal folds and the algorithm is trained on all data except one fold for each of the k folds.[30] The data contained in the fold left out of the training data is used to calculate prediction error. The yacht dataset used 4-fold cross-validation due to the low number of samples.

All experiments used the same data for each value of ϕ. The search range for ϕ_{RT} was [0.1,0.16], with steps of 0.01 and the search range for ϕ_{MRT} was [0.05,0.01,. . . ,1.5] with steps of 0.1. The resulting mean RMSE is shown in Figure 2 and the standard deviation is not shown for clarity. The training data for the search was only generated once. There will be variation in the results shown in Figure 2 for a different set of generated data but it was found that the trends presented in the figure can still be used to find a best ϕ, which results in a low prediction error. The best values for ϕ that were found are shown in Table 2.

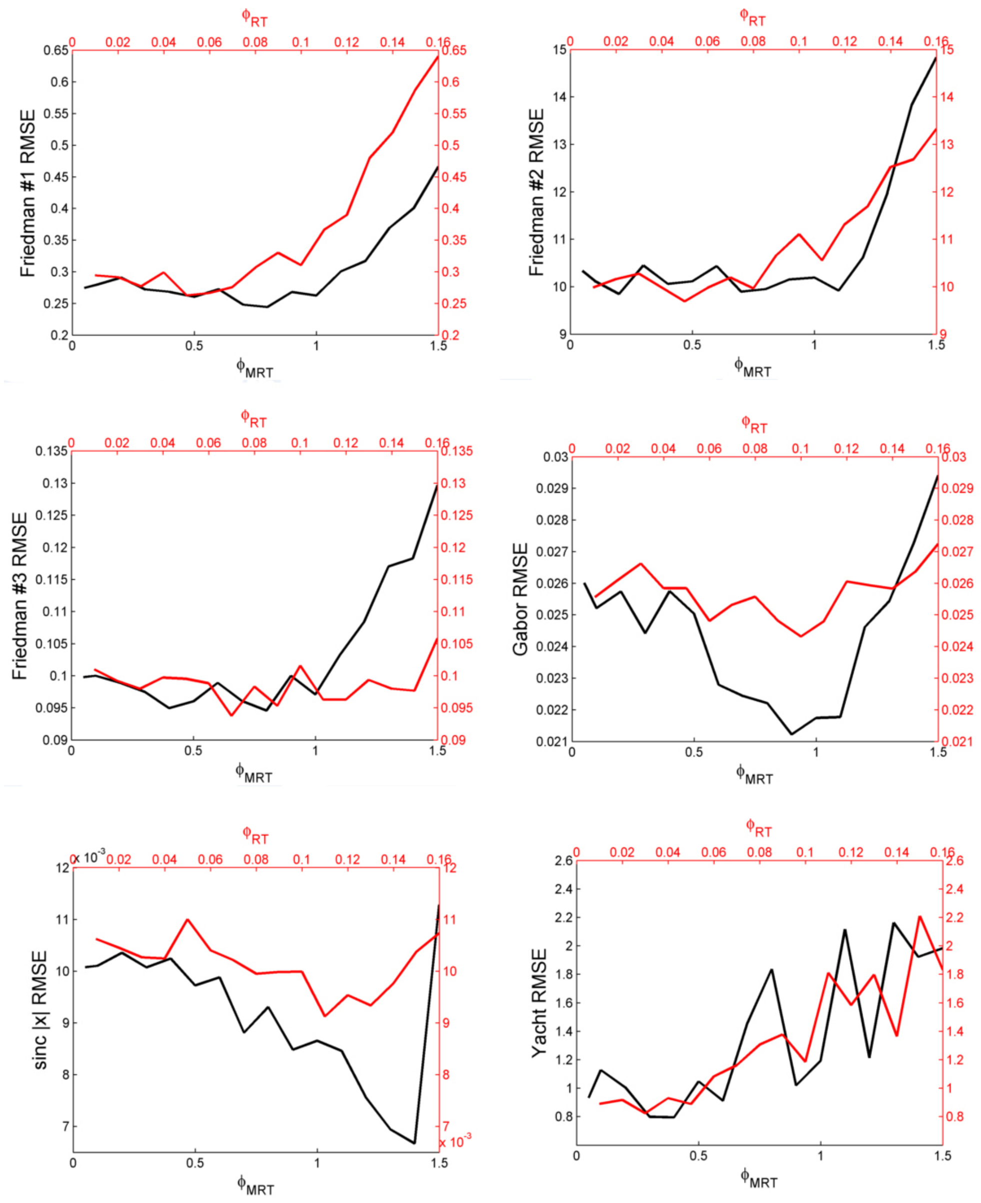


Figure 2: Variation of ϕ_{MRT}(black) and ϕ_{RT}(red) and the resulting mean RMSE for a single cross-validation test for six singlevariate output datasets.

Table 2: Training parameters for the datasets, showing the number of nodes in the hidden layers, the Adaboost.RT threshold ϕ_{RT}, the Adaboost.MRT threshold ϕ_{MRT} and the sample size N for each iteration. The power coefficient n=1 was used for all tests.

Dataset	Nodes, H	ϕ_{RT}	ϕ_{MRT}	Sample Size, N	Noise, ε
Friedman #1	10	0.05	0.8	2000	$U[-2,2]$
Friedman #2	20	0.04	0.9	2000	$U[-120,120]$
Friedman #3	95	0.07	0.5	1200	$U[-0.4,0.4]$
Gabor	45	0.10	0.9	1200	$U[-0.2,0.2]$
$sinc\|x\|$	10	0.10	1.3	2000	$U[-0.2,0.2]$
Yacht	10	0.03	0.4	150	-

There are significant performance gains in the $sinc|x|$ and the Gabor function prediction for a wide range of ϕ_{MRT}. This performance gain can be attributed to the zero-crossing in the data, which causes over-fitting in the prediction of Adaboost.RT. The main factor that affects the selection of ϕ is the prediction error distribution. Large errors were observed for ϕ_{MRT}>1.5, which can be attributed to lack of diversity in the training data, since the learners will only fit to extreme outliers if the threshold is too large.

3.3 Performance comparison for singlevariate datasets

The parameters from Table 2 were used to compare the prediction performance of Adaboost.RT, Adaboost.MRT, a single iteration of the base learning algorithm, and a simple averaging ensemble. The root mean square error (RMSE) was used as the metric for performance comparison of the ensemble methods. The data for the comparison was generated once and the learners were trained 30 times. At each of the 30 repetitions the data was randomly shuffled. Of the total number of data points m, 9/10 was used for training and 1/10 was used for testing. Previous work suggests that there is little reduction in RMSE past T=10 iterations for Adaboost.MRT,[31] thus T=10 iterations was used for all experiments.

The resulting mean RMSE and the standard deviation of the 30 repetitions is shown in Table 3. The results in Table 3 differ slightly from Figure 2, especially for the $sinc|x|$ function. The optimum threshold search used a cross-validation and the data was also regenerated for the performance evaluation. Adaboost.MRT has the lowest numerical mean for four of the six datasets and outperforms Adaboost.RT on five of the six datasets (except the Yacht dataset). The simple averaging ensemble outperforms all other ensembles on the Friedman #3 dataset by a small margin.

Adaboost.MRT significantly outperforms Adaboost.RT on the $sinc|x|$, and the Gabor dataset, due to the zero-crossing in the output data. The Friedman #2 dataset does not contain a zero-crossing; however the addition of noise likely resulted in the introduction of some, thus letting Adaboost.MRT outperform Adaboost.RT.

These results show that Adaboost.MRT should be considered over Adaboost.RT for output that contains a zero-crossing (based on $sinc|x|$ and Gabor performance). In some cases, the noise in the system can introduce zero crossings, which may reduce the accuracy of Adaboost.RT (Friedman #2). Adaboost.MRT has outperformed Adaboost.RT in five out of six of the experiments while maintaining the same number of tuning parameters.

Table 3: Performance comparison of Adaboost.RT, Adaboost.MRT, a single iteration of the ANN, and a simple averaging ensemble trained on the same data. The lowest values for each dataset are shown in bold font.

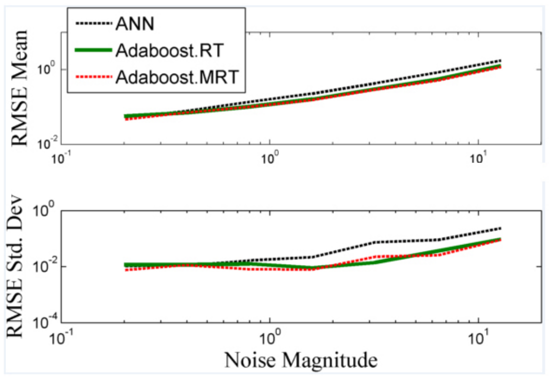
Dataset	Method	RMSE Mean	RMSE Std. Dev.
Friedman #1	Adaboost.RT	0.2936	0.0367
	Adaboost.MRT	**0.2674**	0.0373
	Averaging Ensemble	0.2873	**0.0347**
	Single ANN	0.3664	0.1765
Friedman #2	Adaboost.RT	10.221	0.845
	Adaboost.MRT	**9.174**	**0.714**
	Averaging Ensemble	9.485	0. 827
	Single ANN	12.237	1.9853
Friedman #3	Adaboost.RT	0.0965	0.0091
	Adaboost.MRT	0.0964	0.0096
	Averaging Ensemble	**0.0941**	**0.0083**
	Single ANN	0.1251	0.0187
$sinc\|x\|$	Adaboost.RT	0.00994	0.00126
	Adaboost.MRT	**0.00618**	**0.00084**
	Averaging Ensemble	0.00966	0.00099
	Single ANN	0.01268	0.00474
Gabor	Adaboost.RT	0.02564	0.00150
	Adaboost.MRT	**0.02126**	**0.00137**
	Averaging Ensemble	0.02422	0.00170
	Single ANN	0.03294	0.00437
Yacht	Adaboost.RT	**0.8204**	**0.2661**
	Adaboost.MRT	0.9170	0.3693
	Averaging Ensemble	0.9250	0.3890
	Single ANN	1.2360	1.6643

3.4 Performance comparison with noisy data

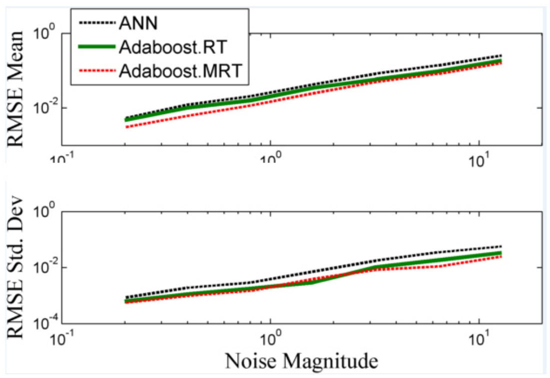
The redefinition of the error function from the absolute relative error to the variance-scaled error function should reduce the RMSE in the presence of noise. Noise increases the probability of near-zero values, which result in over-fitting with Adaboost.RT. This claim is investigated in this section using two datasets with an increasing signal-to-noise ratio (SNR). The Friedman #3 dataset was used, since the performance for Adaboost.RT and Adaboost.MRT were similar in Table 3. The $sinc|x|$ function was selected to test performance gains with an already present zero-crossing.

The noise was uniformly distributed with zero mean. The noise amplitude was varied from a value of 0.2 and incrementally doubled to 12.8 resulting in seven different noise magnitudes. The parameters for the learners are shown in Table 2. The performance was assessed under the same conditions as Section 3.3, except with only 10 repetitions.

The average RMSE values were recorded for Adaboost.MRT, Adaboost.RT, and the ANN. The results for both datasets are shown in Figure 2. In the presence of noise Adaboost.MRT achieves an approximately 5% smaller RMSE than Adaboost.RT for the Friedman #3 dataset (see Figure 3 (a)) and a larger RMSE reduction for the $sinc|x|$ function (see Figure 3 (b)) for a large range of noise magnitudes.



(a) Friedman #3 dataset



(b) sinc|x| dataset

Figure 3: RMSE mean and standard deviation for a range of noise magnitudes for the Friedman #3 and the $sinc|x|$ datasets.

4 Performance evaluation for vector output datasets

4.1 Datasets

This section investigates the proposed extension of Adaboost.MRT to accommodate vector output data. The proposed Adaboost.MRT is compared against a single iteration of its base learner, bagging, and a simple averaging ensemble. Three datasets were used for the comparison: two nonlinear functions and a dataset from the UCI repository. Table 4 shows the equations used to generate the datasets. A low number of hidden nodes was selected for the multivariate output experiments to test Adaboost.MRT's ability to boost the accuracy of weak learners.

Table 4: Dataset equations for the multivariate output performance evaluations.

Dataset	Function	Input Variables	Size, *m*	Sample Size, *N*
3D Cone	$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{(h-z)}{h} r_d \cos\theta \\ \frac{(h-z)}{h} r_d \sin\theta \end{bmatrix}$	$h = 10,\ r_d = 10$ $z \sim \boldsymbol{U}[0,10]$ $\theta \sim \boldsymbol{U}[-\pi,\pi]$	1000	900
Swiss Roll	$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \theta\cos\theta \\ d \\ \theta\sin\theta \end{bmatrix}$	$d \sim \boldsymbol{U}[0,10]$ $\theta \sim \boldsymbol{U}[0,4\pi]$	500	200
Energy Efficiency	$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f(x_1, x_2, \dots x_i)$	$x_i\ , i = 1 \dots 8$	768	576

The first dataset is a function that describes a cone in 3D space, with two input variables $\boldsymbol{x} = [x_1, x_2] = [z, \theta]$ and two output variables $\boldsymbol{y} = [y_1, y_2] = [x, y]$. The constant parameter h indicates the height of the cone and r_d is the cone radius when z=0. Uniformly distributed noise $\varepsilon \sim \boldsymbol{U}[-2, 2]$ was added to the output variables. The neural network base learner consisted of one hidden layer and H=5 hidden nodes.

The second dataset is the Swiss Roll function with two inputs $\boldsymbol{x} = [x_1, x_2] = [d, \theta]$ and three outputs $\boldsymbol{y} = [y_1, y_2, y_3] = [x, y, z]$. Uniformly distributed noise $\varepsilon =$

$\boldsymbol{U}[-1.5, 1.5]$ was added to both output variables. The RMSE was calculated against the noiseless output variables. The neural network consisted of H=3 hidden nodes.

The energy efficiency dataset[32] from the UCI Machine Learning Repository predicts the heating and cooling load on buildings depending on their parameters such as wall area, roof area, surface area, etc. The dataset has eight real-valued input variables and two real-valued output variables. Uniformly distributed noise $\varepsilon \sim \boldsymbol{U}[-5, 5]$ was added to the Energy Efficiency dataset output variables. The base learner neural network consisted of H=5 hidden nodes.

4.2 Comparison methods

Adaboost.MRT was compared to a single iteration of its base learner, to a simple averaging ensemble, and bagging. Each of the methods used the same base learner, the artificial neural network (ANN). The bagging ensemble and the simple averaging ensemble have previously been used in Ref.20 for method validation. Adaboost.MRT combines the output of multiple weak learners into a strong learner by manipulating the training data, thus it was compared against other boosting and bagging ensemble methods only.

The bagging method has been chosen because it is an ensemble method similar to boosting. Each sample has the same likelihood to be picked at each iteration. The sampling likelihood of a sample in boosting is independent of the performance of the previous learner. The bagging ensemble consists of T=10 individual learners. The training data for each leaner is sampled from the available data with replacement and the output of the ensemble is averaged. The sample size N was 900, 200, and 576 for the 3D cone, the Swiss roll, and the energy efficiency dataset, respectively.

The simple averaging ensemble consists of 10 iterations of the base learning algorithm. At each iteration, the averaging ensemble is passed N examples, which were randomly chosen with equal likelyhood. The output of the ensemble is an averaged output of the 10 learners. The averaging ensemble is used as comparison to investigate whether Adaboost.MRT is better than just the average output of multiple learners.

The data for the comparison was generated once and reshuffled after each test. Each test took the m examples and used 1/10$^{\text{th}}$ for validation and 9/10$^{\text{th}}$ for learning. The test was repeated 30 times and a final RMSE mean and standard deviation of the 30 test was reported.

4.3 Adaboost.MRT threshold parameter selection

The parameter ϕ for the multivariate case is a multi-dimensional vector $\phi = [\phi_1 \cdots \phi_R]$, which results in multiple tuning parameters that affect the expected accuracy of the ensemble. The best or optimal value ϕ_{opt} for each output variable for each dataset is experimentally determined in this paper.

The best value ϕ_r for output y_r was determined by keeping all other $\phi \neq r$ constant while changing ϕ_r from 0.1 to 1.4. For each value of ϕ_r a 10-fold cross-valdiation was peformed and a mean RMSE calculated. From the results, a general trend was determined and a "best" ϕ_r was selected. This process was repeated for each of the R output variables. The best determined value for the 3D Cone data set was $\phi = [1.1, 1.2]$, for the Swiss Roll dataset $\phi = [0.6, 0.2, 0.6]$ and for the Energy Efficiency dataset $\phi = [0.7, 0.9]$. This threshold selection becomes more complex and time-consuming as more output variables are added. Difficulty was experienced when tuning the Swiss Roll dataset, due to the three output variables, which all affect the RMSE. Further research into automated threshold selection methods (via an optimization, for example) is warranted.

4.4 Performance comparison for multivariate datasets

The RMSE mean and standard deviation for the performance comparison is shown in Table 5, with the lowest value for the mean and standard deviation RMSE shown in bold font. All ensembles outperform a single iteration of their base learner, except for Adaboost for the Swiss Roll dataset y output variable, in which the single ANN outperformed the ensembles.

Simple averaging performs well when a strong learner, such as a neural network with a large number of hidden nodes, is used. Adaboost.MRT, a boosting method, outperforms bagging and the simple averaging ensemble in the performed experiments.

Some difficulties with relative accuracy increases were experienced with the Swiss roll dataset. As the accuracy of one variables was increased, the accuracy of others diminished. This points towards a difficulty with multivariate datasets in achieving the lowest RMSE for all output variables.

Overall, Adaboost.MRT performs well, when pitted against similar type of ensembles. Adaboost.MRT is capable of boosting the accuracy of weak learners at the cost of introducing a tuning parameter. More investigation into the threshold parameters selection is warranted, as previously suggested for Adaboost.RT.[19]

Table 5: Performance comparison of Adaboost.MRT to bagging, a simple averaging ensemble, and a single iteration of the base learner for the 3D Cone, the Swiss Roll, and the Energy Efficiency datasets. Lowest values are shown in bold font.

Function	Output Variable	Method	RMSE Mean	RMSE Std. Dev.
3D Cone	x	Adaboost.MRT	**0.1391**	0.0237
		Bagging	0.1656	0.0276
		Simple Average	0.1608	**0.0176**
		Single ANN	0.2212	0.0305
	y	Adaboost.MRT	**0.1328**	**0.0202**
		Bagging	0.1709	0.0226
		Simple Average	0.1690	0.0205
		Single ANN	0.2446	0.0363
Swiss Roll	x	Adaboost.MRT	**2.706**	**0.367**
		Bagging	2.849	0.390
		Simple Average	2.973	0.571
		Single ANN	3.184	1.428
	y	Adaboost.MRT	0.868	0.302
		Bagging	0.458	0.169
		Simple Average	0.369	**0.143**
		Single ANN	**0.294**	0.146
	z	Adaboost.MRT	**3.158**	0.290
		Bagging	3.663	**0.193**
		Simple Average	3.602	0.258
		Single ANN	3.952	0.511
Energy Efficiency	y_1	Adaboost.MRT	**2.526**	**0.154**
		Bagging	2.701	0.155
		Simple Average	2.748	0.155
		Single ANN	3.025	0.607
	y_2	Adaboost.MRT	**2.905**	**0.151**
		Bagging	3.039	0.171
		Simple Average	3.076	0.181
		Single ANN	3.286	0.468

5 Conclusions

In this paper Adaboost.MRT, a boosting algorithm for multivariate output regression, was introduced. The proposed algorithm is based on Adaboost.RT and addresses the singularity in the misclassification function by introducing a variance-scaled misclassification function and modifies the ensemble output from a weighted to an averaged output. To the best of the author's knowledge, Adaboost.MRT is the first algorithm that extends Adaboost to multivariate regression, resulting in a boosted accuracy for machine learning with multivariate outputs and weak learners. Adaboost.MRT has presented some insensitivity to noise in the training data, which led to an increase in accuracy. Multivariate regression is extensively used in machine learning for data mining, system identification and advanced control.

Experiments with six commonly used datasets showed that Adaboost.MRT outperforms Adaboost.RT on training sets that contain high magnitude noise and training sets with a zero-crossing in the output data and performed similarly to Adaboost.RT on others. Experiments suggest that the proposed method is applicable to a wider range of cases than Adaboost. RT.

Experiments on Adaboost.MRT on three multivariate output functions have been performed and its performance compared to bagging, a simple averaging ensemble, and a single iteration of the base learning algorithm. In most cases, there was some error reduction in the RMSE or comparable performance over the other methods at the cost of the introduction of a new tuning parameter.

Experiments suggest that for a small number of output variables the RMSE of an output variable is most sensitive to the threshold parameter of the specific output variable, a claim that breaks down as more output variables are added. For a very small number of output variables the threshold can be selected and tuned for each variable individually. Further testing on larger sets and more diverse combinations of output variables is warranted.

In Adaboost.MRT, the expected accuracy depends on the base learning algorithm and the noise distribution of the predictions. Following the selection of the base learning algorithm, the values of ϕ and T need to be chosen. There is a reduction in accuracy increase beyond T = 10 iterations. The optimum value of ϕ can be experimentally determined and remains consistent for a given prediction error distribution. Future work should focus on the selection of the threshold parameter ϕ. The optimal parameter value is likely linked to the distribution of the prediction error. Further research on an adaptive threshold parameter selection method that automates the threshold selection, or an investigation in the relation between the noise distribution and the optimal parameter value, is warranted.

References

[1] G. Brown, J. Wyatt, and P. Tiňo. Managing diversity in regression ensembles. The Journal of Machine Learning Research. 2005; 6: 1621–1650.

[2] L. Hansen and P. Salamon. Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1990; 12(10): 993–1001. http://dx.doi.org/10.1109/34.58871

[3] T. Dietterich. Ensemble methods in machine learning. Multiple classifier systems. 2000: 1–15.

[4] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. Machine Learning. 1999; 37(3): 297–336. http://dx.doi.org/10.1023/A:1007614523901

[5] R. Maclin and D. Opitz. Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research. 1999; 11: 169–198.

[6] H. Drucker. Improving regressors using boosting techniques. International Conference on Machine Learning. 1997; 97: 107–115.

[7] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine Learning. 2000; 40(2): 139–157. http://dx.doi.org/10.1023/A:1007607513941

[8] G. Ridgeway. The state of boosting. Computing Science and Statistics. 1999: 172–181.

[9] Y. Freund and R. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences. 1995; 55(1): 119–139.

[10] Y. Freund and R. Schapire. A short introduction to boosting. Journal-Japanese Society For Artificial Intelligence. 1999; 14(5): 771–780.

[11] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. in International Conference on Machine Learning. 1996; 96: 148–156.

[12] Y. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang. Cost-sensitive boosting for classification of imbalanced data. Pattern Recognition. 2007 Dec; 40(12): 3358–3378. http://dx.doi.org/10.1016/j.patcog.2007.04.009

[13] J. Cao, S. Kwong, and R. Wang. A noise-detection based AdaBoost algorithm for mislabeled data. Pattern Recognition. 2012; 45(2): 4451–4465. http://dx.doi.org/10.1016/j.patcog.2012.05.002

[14] P. Melville and R. J. Mooney. Creating diversity in ensembles using artificial data. Information Fusion. 2005 Mar; 6(1): 99–111. http://dx.doi.org/10.1016/j.inffus.2004.04.001

[15] N. Oza. Boosting with averaged weight vectors. in Multiple Classifier Systems. 2003; 4: 15–24.

[16] N. Oza. Aveboost2: Boosting for noisy data. in Multiple Classifier Systems. 2004: 31–40.

[17] G. Ridgeway, D. Madigan, and T. Richardson. Boosting methodology for regression problems. in Proceedings of the International Workshop on AI and Statistics. 1999: 152–161.

[18] D. P. Solomatine and D. L. Shrestha. AdaBoost. RT: a boosting algorithm for regression problems. in IEEE International Joint Conference on Neural Networks. 2004; 2: 1163–1168.

[19] D. Shrestha and D. Solomatine. Experiments with AdaBoost. RT, an improved boosting scheme for regression. Neural computation. 2006 Jul; 18(7): 1678–710. PMid:16764518 http://dx.doi.org/10.1162/neco.2006.18.7.1678

[20] D. Wang and M. Alhamdoosh. Evolutionary extreme learning machine ensembles with size control. Neurocomputing. 2013 Feb; 102: 98–110. http://dx.doi.org/10.1016/j.neucom.2011.12.046

[21] H. Tian, Z. Mao, and Y. Wang. Hybrid modeling of molten steel temperature prediction in LF. ISIJ international. 2008; 48(1): 58–62. http://dx.doi.org/10.2355/isijinternational.48.58

[22] H. Tian and Z. Mao. An Ensemble ELM Based on Modified AdaBoost.RT Algorithm for Predicting the Temperature of Molten Steel in Ladle Furnace. IEEE Transactions on Automation Science and Engineering. 2010 Jan; 7(1): 73–80. http://dx.doi.org/10.1109/TASE.2008.2005640

[23] Y.-H. Yang, Y.-C. Lin, Y.-F. Su, and H. H. Chen. Music emotion classification: A regression approach. in Multimedia and Expo. 2007: 208–211.

[24] E. O. Costa, A. Trinidad, R. Pozo, and S. R. Vergilio. A genetic programming approach for software reliability modeling. IEEE Transactions on Reliability. 2010; 59(1): 222–230. http://dx.doi.org/10.1109/TR.2010.2040759

[25] A. Savran, B. Sankur, and M. Taha Bilge. Regression-based intensity estimation of facial action units. Image and Vision Computing. 2012 Oct; 30(10): 774–784. http://dx.doi.org/10.1016/j.imavis.2011.11.008

[26] Z. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. Artificial Intelligence. 2002 May; 137(1): 239–263.

[27] M. Bache, K. Lichman. UCI Machine Learning Repository http://archive.ics.uci.edu/ml. University of California, Irvine, School of Information and Computer Sciences[Internet]. 2013. Available from: http://archive.ics.uci.edu/ml

[28] J. H. Friedman. Multivariate adaptive regression splines. The Annals of Statistics. 1991: 1–67.

[29] L. Breiman. Bagging predictors. Machine learning. 1996; 24(2): 123–140. http://dx.doi.org/10.1023/A:1018054314350

[30] R. R. Bouckaert. Choosing between two learning algorithms based on calibrated tests. in International Conference on Machine Learning (ICML-03). 2003: 51–58.

[31] N. Kummer. Translational visual servoing control of quadrotor helicopters. University of British Columbia. 2013.

[32] A. Tsanas and A. Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. Energy and Buildings. 2012 Jun; 49: 560–567. http://dx.doi.org/10.1016/j.enbuild.2012.03.003