Transform Regression and the Kolmogorov Superposition Theorem

Edwin Pednault
IBM T. J. Watson Research Center
1101 Kitchawan Road, P.O. Box 218
Yorktown Heights, NY 10598 USA
pednault@us.ibm.com

Abstract

This paper presents a new predictive modeling algorithm that draws inspiration from the Kolmogorov superposition theorem. An initial version of the algorithm is presented that combines gradient boosting, generalized additive models, and decision-tree methods to construct models that have the same overall mathematical structure as Kolmogorov's superposition equation. Improvements to the algorithm are then presented that significantly increase its rate of convergence. The resulting algorithm, dubbed "transform regression," generates surprisingly good models compared to those produced by the underlying decision-tree method when the latter is applied directly. Transform regression is highly scalable and a parallelized database-embedded version of the algorithm has been implemented as part of IBM DB2 Intelligent Miner Modeling.

Keywords: Gradient boosting, Generalized additive modeling, Decision trees.

1. Introduction

In many respects, decision trees and neural networks represent diametrically opposed classes of learning techniques. A strength of one is often a weakness of the other. Decision-tree methods approximate response surfaces by segmenting the input space into regions and using simple models within each region for local surface approximation. The strengths of decision-tree methods are that they are nonparametric, fully automated, and computationally efficient. Their weakness is that statistical estimation errors increase with the depth of trees, which ultimately limits the granularity of the surface approximation that can be achieved for fixed sized data.

In contrast, neural network methods fit highly flexible families of nonlinear parametric functions to entire surfaces to construct global approximations. The strength of this approach is that it avoids the increase in estimation error that accompanies segmentation and local model fitting. The weakness is that fitting nonlinear parametric functions to data is computationally demanding, and these demands are exacerbated by the fact that several network architectures often need to be trained and evaluated in order to maximize predictive accuracy.

This paper presents a new modeling approach that attempts to combine the strengths of the methods described above—specifically, the global fitting aspect of neural networks with the automated, computationally efficient, and nonparametric aspects of decision trees. To achieve this union, this new modeling method draws inspiration from the Kolmogorov superposition theorem:

Theorem (Kolmogorov, 1957). For every integer dimension $d \geq 2$, there exist continuous real functions $h_{ij}(x)$ defined on the unit interval $U = [0,1]$, such that for every continuous real function $f(x_1,\ldots,x_d)$ defined on the d-dimensional unit hypercube U^d, there exist real continuous functions $g_i(x)$ such that

$$f(x_1,\ldots,x_d) = \sum_{i=1}^{2d+1} g_i\left(\sum_{j=1}^{d} h_{ij}(x_j)\right) .$$

Stronger versions of this theorem have also been reported (Lorentz, 1962; Sprecher, 1965). The theorem is interesting because it states that even the most complex multivariate functions can be decomposed into combinations of univariate functions, thereby enabling cross-product interactions to be modeled without introducing cross-product terms in the model.

Hecht-Nielson (1987) has noted that the superposition equation can be interpreted as a three-layer neural network and has suggested using the theorem as basis for understanding multilayer neural networks. Girosi and Poggio (1989), on the other hand, have criticized this suggestion for several reasons, one being that applying Kolmogorov's theorem would require the inductive learning of nonparametric activation functions. Neural network methods, by contrast, usually assume that the activation functions are given and the problem is to learn values for the weights that appear in the networks. Although the usual paradigm for training weights can be extended to incorporate the learning of smooth parametric activation functions (i.e., by including their parameters in the partial derivatives that are calculated during training), the incorporation of nonparametric learning methods into the training paradigm was seen as problematic.

Nonparametric learning, on the other hand, is a key strength of decision-tree methods. The learning of nonparametric activation functions thus provides a starting point for combining the global-fitting aspects neural network methods with nonparametric learning aspects of decision-tree methods.

In the sections that follow, an initial algorithm is presented and then subsequently refined that uses decision-tree methods to inductively learn instantiations of the g_i and h_{ij} functions that appear in Kolmogorov's superposition equation so as to make the equation a good predictor of underlying response surfaces. In this respect, the initial algorithm and its refinement are inspired by, but are not mathematically based upon, Kolmogorov's theorem. The g_i and h_{ij} functions that are created by the algorithms presented here are quite different from those that are constructed in the various proofs of Kolmogorov's theorem and its variants. The latter are highly nonsmooth fractal functions that in some respects are comparable to hashing functions (Girosi and Poggio, 1989). Moreover, Kolmogorov's theorem requires that the h_{ij} functions be universal for a given dimension d; that is, the h_{ij} functions are fixed for each dimension d and only the g_i functions depend on the specific function f. The initial algorithm presented below, on the other hand, heuristically constructs both g_i and h_{ij} functions that depend on training data.

It is important to note that neither universality nor the specific functional forms of the g_i and h_{ij} functions that appear in the various proofs of Kolmogorov's theorem are necessary in order to satisfy the superposition equation. For example, consider the function $f(x,y) = xy$. This function can be rewritten in superpositional form as $f(x,y) = 0.25(x + y)^2 - 0.25(x - y)^2$. In this case, $h_{11}(x) = h_{21}(x) = x$, $h_{12}(y) = y$, $h_{22}(y) = -y$, $h_{ij}(z) = 0$ for $i,j > 2$, $g_1(z) = 0.25z^2$, $g_2(z) = -0.25z^2$, and $g_i(z) = 0$ for $i > 2$. Although these particular g_i and h_{ij} functions satisfy the superposition equation, they do not satisfy the preamble of the theorem because the above h_{ij} functions are not universal for all continuous functions $f(x,y)$. Nor do the above g_i and h_{ij} functions at all resemble the g_i and h_{ij} functions that are constructed in the proofs of Kolmogorov's theorem and its variants. In general, for any given function $f(x_1,\ldots,x_d)$, there can exist a wide range of g_i and h_{ij} functions that satisfy the superposition equation without satisfying the preamble of the theorem.

Taking the above observations to heart, the initial algorithm presented below likewise ignores the preamble of Kolmogorov's theorem and instead focuses on the mathematical structure of the superposition equation itself. Decision-tree methods and greedy search heuristics are used to construct g_i and h_{ij} functions based on training data in an attempt to make the superposition equation a good predictor. The approach contrasts with previous work on direct application of the superposition theorem (Neruda, Štědrý, & Drkošová, 2000; Sprecher 1996, 1997, 2002). One difficulty with direct application is that the g_i and h_{ij} functions that need to be constructed are extremely complex and entail very large computational overheads to implement, even when the target function is known (Neruda, Štědrý, & Drkošová, 2000). Another problem is that noisy data is highly problematic. The approach presented here avoids both these issues, but it is heuristic in nature. Although there are no mathematical guarantees of obtaining good predictive models using this approach, the initial algorithm and its refinements nevertheless produce very good results in practice. Thus, one of the contributions of this paper is to demonstrate that the mathematical form of the superposition theorem is interesting in and of itself, and can be heuristically exploited to obtain good predictive models.

The initial algorithm is based on heuristically interpreting Kolmogorov's superposition equation as a gradient boosting model (Friedman, 2001, 2002) in which the base learner constructs generalized additive models (Hastie & Tibshirani, 1990) whose outputs are then nonlinearly transformed to remove systematic errors in their residuals. To provide the necessary background to motivate this interpretation, gradient boosting and generalized additive modeling are first briefly overviewed in Sections 2 and 3. The initial algorithm is then presented in Section 4.

The initial algorithm, however, has very poor convergence properties. Sections 5 and 6 therefore present improvements to the initial algorithm to obtain much faster rates of convergence, yielding a new algorithm called transform regression. Faster convergence is achieved by modifying Friedman's gradient boosting framework so as to introduce a nonlinear form of Gram-Schmidt orthogonalization. The modification requires generalizing the mathematical forms of the models to allow constrained multivariate g_i and h_{ij} functions to appear in the resulting models in order to implement the orthogonalization method. The resulting models thus depart from the pure univariate form of Kolmogorov's superposition equation, but the benefit is significantly improved convergence properties. The nonlinear Gram-Schmidt orthogonalization technique is another contribution of this paper, since it can be combined with other gradient boosting algorithms to obtain similar benefits, such as Friedman's (2001, 2002) gradient tree boosting algorithm.

Section 7 presents evaluation results that compare the performance of the transform regression algorithm to the underlying decision-tree method that is employed. In the evaluation study, transform regression often produced better predictive models than the underlying decision-tree method when the latter was applied directly. This result is interesting because transform regression uses decision trees in a highly constrained manner.

Section 8 discusses some of the details of a parallelized database-embedded implementation of transform regression that was developed for IBM DB2 Intelligent Miner Modeling.

Section 9 presents conclusions and discusses possible directions for future work.

2. Gradient boosting

Gradient boosting (Friedman, 2001, 2002) is a method for making iterative improvements to regression-based predictive models. The method is similar to gradient descent except that, instead of calculating gradient directions in parameter space, a learning algorithm (called the *base learner*) is used to estimate gradient directions in function space. Whereas with conventional gradient descent each iteration contributes an additive update to the current vector of parameter values, with gradient boosting each iteration contributes an additive update to the current regression model.

When the learning objective is to minimize total squared error, gradient boosting is equivalent to Jiang's LSBoost.Reg algorithm (Jiang, 2001, 2002) and to Mallat and Zhang's matching pursuit algorithm (Mallat and Zhang, 1993). In the special case of squared-error loss, the gradient directions in function space that are estimated by the base learner are models that predict the residuals of the current overall model. Model updating is then accomplished by summing the output of the current model with the output of the model for predicting the residuals, which has the effect of adding correction terms to the current model to improve its accuracy.

The resulting gradient boosting algorithm is summarized in Table 1. If the base learner is able to perform a true least-squares fit on the residuals at each iteration, then the multiplying scalar α will always be equal to one. In the general case of an arbitrary loss function, the "residual error" (a.k.a., *pseudo-residuals*) at each iteration is equal to the negative partial derivative of the loss function with respect to the model output for each data record. In this more general setting, line search is usually needed to optimize the value of α.

Table 1. The gradient boosting method for minimizing squared error.

```
Let the current model $M$ be zero everywhere;
Repeat until the current model $M$ does not appreciably change:
    Use the base learner to construct a model $R$ that predicts the residual error of $M$, ensuring that $R$ does not overfit the data;
    Find a value for scalar $\alpha$ that minimizes the loss (i.e., total squared error) for the model $M + \alpha R$;
    Update $M \leftarrow M + \alpha R$;
```

When applying gradient boosting, overfitting can be an issue and some means for preventing overfitting must be employed (Friedman, 2001, 2002; Jiang, 2001, 2002). For the algorithms presented in this paper, a portion of the training data is reserved as a holdout set which the base learner employs at each iteration to perform model selection for the residual models. Because it is also possible to overfit the data by adding too many boosting stages (Jiang, 2001, 2002), this same holdout set is also used to prune the number of gradient boosting stages. The algorithms continue to add gradient boosting stages until a point is reached at which either the net improvement obtained on the training data as a result of an iteration falls below a preset threshold, or the prediction error on the holdout set has increased steadily for a preset number of iterations. The current model is then pruned back to the boosting iteration that maximizes predictive accuracy on the holdout set.

3. Generalized additive models

Generalized additive models (Hastie & Tibshirani, 1990), is a method for constructing regression models of the form:

$$\tilde{y} = y_0 + \sum_{j=1}^{d} h_j(x_j) \ .$$

Hastie and Tibshirani's *backfitting* algorithm is typically used to perform this modeling task. Backfitting assumes the availability of a learning algorithm called a *smoother* for estimating univariate functions. Traditional examples of smoothers include classical smoothing algorithms that use kernel functions to calculate weighted averages of training data, where the center of the kernel is the point at which the univariate function is to be estimated and the weights are given by the shapes of the kernel functions. However, in general, any learning algorithm can be used as a smoother, including decision tree methods. An attractive aspect of decision tree methods is that they can explicitly handle both numerical and categorical input features, whereas classical kernel smoothers require preprocessing to construct numerical encodings of categorical features.

With the backfitting algorithm, the value of y_0 would first be set to the mean of the target variable and a smoother would then be applied to successive input variables x_j in round-robin fashion to iteratively (re)estimate the h_j functions until convergence is achieved. The resulting algorithm is summarized in Table 2.

Overfitting and loop termination can be handled in a similar fashion as for gradient boosting. The initialization of the h_j functions can be accomplished by setting them to be zero everywhere. Alternatively, one can obtain very good initial estimates by applying a

Table 2. The backfitting algorithm.

```
Set $y_0$ equal to the mean of the target variable $y$;
For $j = 1,\dots,d$ initialize the function $h_j(x_j)$;
Repeat until the functions $h_j(x_j)$ do not appreciably change:
    For $j = 1,\dots,d$ do the following:
        Use the smoother to construct a model $H_j(x_j)$ that predicts the following target value using only input feature $x_j$:
            new target $= y - y_0 - \sum_{k \neq j} h_k(x_k)$
        Update $h_j(x_j) \leftarrow H_j(x_j)$;
```

Table 3. A greedy one-pass additive modeling algorithm.

```
Set $y_0$ equal to the mean of the target variable $y$;
For $j = 1,\dots,d$ use the smoother to construct a model $H_j(x_j)$ that predicts the target value $(y - y_0)$ using only input feature $x_j$:
Calculate linear regression coefficients $\lambda_j$ such that $\sum_j \lambda_j H_j(x_j)$ is a best predictor of $(y - y_0)$;
For $j = 1,\dots,d$ set $h_j(x_j) = \lambda_j H_j(x_j)$;
```

greedy one-pass approximation to backfitting that independently applies the smoother to each input x_j and then combines the resulting models using linear regression. This one-pass additive modeling algorithm is summarized in Table 3.

Remarkably, the greedy one-pass algorithm shown in Table 3 can often produce surprisingly good models in practice without additional iterative backfitting. The one-pass algorithm also has the advantage that overfitting can be controlled in the linear regression calculation, either via feature selection or by applying a regularization method. This overfitting control is available in addition to overfitting controls that may be provided by the smoother. Examples of the latter include kernel-width parameters in the case of classical kernel smoothers and tree pruning in the case of decision-tree methods.

4. An initial algorithm

As mentioned earlier, inspiration for the initial algorithm presented below is based on interpreting Kolmogorov's superposition equation as a gradient boosting model in which the base learner constructs generalized additive models whose outputs are then nonlinearly transformed to remove systematic errors in their residuals. To motivate this interpretation, suppose that we are trying to infer a predictive model for y as function of inputs $x_1,\dots,x_d$ given a set of noisy training data $\{\langle x_1,\dots,x_d, y\rangle\}$. As a first attempt, we might try constructing a generalized additive model of the form

$$\tilde{y}_1 = \sum_{j=1}^{d} h_{1j}(x_j) = y_{10} + \sum_{j=1}^{d} \hat{h}_{1j}(x_j) \ , \tag{1}$$

where

$$h_{1j}(x_j) = \hat{h}_{1j}(x_j) + \tfrac{1}{d} y_{10} \ . \tag{2}$$

This modeling task can be performed by first applying either the backfitting algorithm shown in Table 2 or the greedy one-pass algorithm shown in Table 3, and then distributing the additive constant y_{10} that is obtained equally among the transformed inputs as per Equation 2.

Independent of whether backfitting or the greedy one-pass algorithm is applied, residual nonlinearities can still exist in the relationship between the additive model output $\tilde{y}_1$ and the target value y. To remove such nonlinearities, the same smoother used for additive modeling can again be applied, this time to linearize $\tilde{y}_1$ with respect to y. The resulting combined model would then have the form

$$\hat{y}_1 = g_1(\tilde{y}_1) = g_1\left(\sum_{j=1}^{d} h_{1j}(x_j)\right) \ . \tag{3}$$

To further improve the model, gradient boosting can be applied by using the above two-stage modeling technique as the base learner. The resulting gradient boosting model would then have the form

$$\tilde{y}_i = \sum_{j=1}^{d} h_{ij}(x_j) = \sum_{j=1}^{d} \left(\hat{h}_{ij}(x_j) + \tfrac{1}{d} y_{i0}\right) \tag{4a}$$

$$\hat{y}_i = g_i(\tilde{y}_i) = g_i\left(\sum_{j=1}^{d} h_{ij}(x_j)\right) \tag{4b}$$

$$\hat{y} = \sum_i \hat{y}_i = \sum_i g_i\left(\sum_{j=1}^{d} h_{ij}(x_j)\right) \ . \tag{4c}$$

Equations 4a and 4b define the stages in the resulting gradient boosting model. Equation 4a defines the generalized additive models $\tilde{y}_i$ that are constructed in each boosting stage, while Equation 4b defines the boosting stage outputs $\hat{y}_i$. Equation 4c defines the output

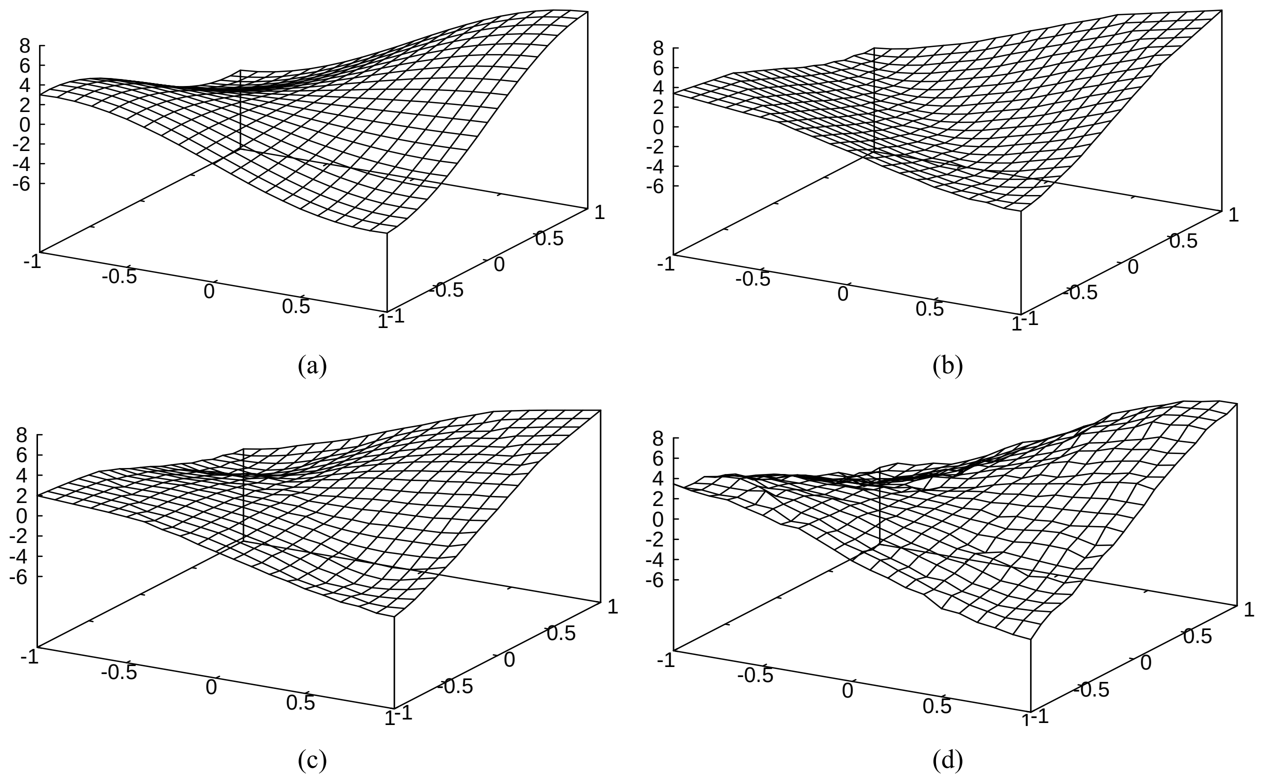


Figure 1. An example of the modeling behavior of the initial algorithm. (a) The target function. (b) The model output after one gradient boosting stage. (c) After two boosting stages. (d) After ten boosting stages.

$\hat{y}$ of the overall model, which is the sum of the boosting stage outputs $\hat{y}_i$. The resulting algorithm will thus generate predictive models that have the same mathematical form as Kolmogorov's superposition equation. The algorithm itself is summarized in Table 4.

Table 4. The initial algorithm.

```
Let the current model $M$ be zero everywhere;
Repeat until the current model $M$ does not appreciably change:
    At the $i$'th iteration, construct an additive model $\tilde{y}_i$ that that predicts the residual error of $M$ using the raw features $x_1,\ldots,x_d$ as input, ensuring that $\tilde{y}_i$ does not overfit the data;
    Construct an additive model $\hat{y}_i$ that predicts the residual error of $M$ using the output of model $\tilde{y}_i$ as the only input feature, ensuring that $\hat{y}_i$ does not overfit the data;
    Update $M \leftarrow M + \hat{y}_i$;
```

To demonstrate the behavior of the initial algorithm, the ProbE linear regression tree (LRT) algorithm (Natarajan & Pednault, 2002) was used as the smoother in combination with the one-pass greedy additive modeling algorithm shown in Table 3. The LRT algorithm constructs decision trees with multivariate linear regression models in the leaves. To prevent overfitting, LRT employs a combination of tree pruning and stepwise linear regression techniques to select both the size of the resulting tree and the variables that appear in the leaf models. Predictive accuracy on a holdout data set is used as the basis for making this selection.

In its use as a smoother, the LRT algorithm was configured to construct splits only on the feature being transformed. In addition, in the case of numerical features, the feature being smoothed was also allowed to appear as a regression variable in the leaf models. The resulting h_{ij} functions are thus piecewise linear functions for numerical features x_j, and piecewise constant functions for categorical features x_j.

For the linear regression operation in Table 3, forward stepwise regression was used for feature selection and the same holdout set used by LRT for tree pruning was used for feature pruning to prevent overfitting.

For illustration purposes, synthetic training data was generated using the following target function:

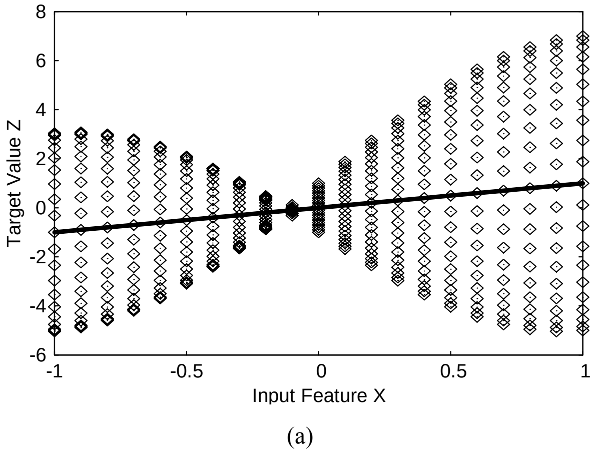
$$z = f(x,y) = x + y + \sin\left(\frac{\pi \cdot x}{2}\right) \cdot \sin\left(\frac{\pi \cdot y}{2}\right) \quad (5)$$

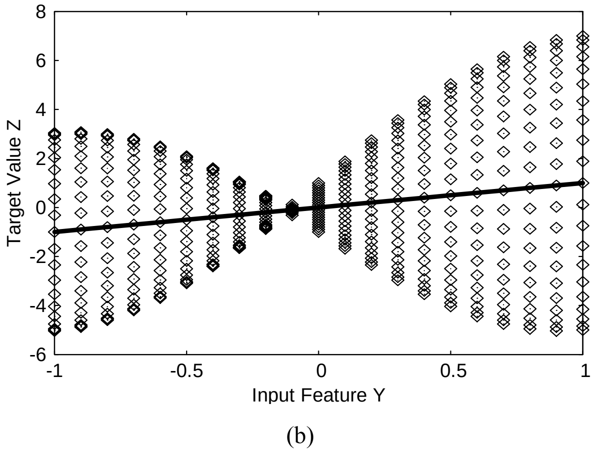
Data was generated by sampling the above function in the region $\langle x, y \rangle \in [-1,1]^2$ at grid increments of 0.01. This data was then subsampled at increments of 0.1 to create a test set, with the remaining data randomly divided into a training set and a holdout set.

Figure 1 illustrates the above target function and the predictions on the test set after one, two, and ten boosting stages. As can be seen in Figure 1, the algorithm is able to model the cross-product interaction expressed in Equation 5, but the convergence of the algorithm is very slow. Even after ten boosting stages, the root mean squared error is 0.239, which is quite large given that no noise was added to the training data.

Nevertheless, Figure 1 does illustrate the appeal of the Kolmogorov superposition theorem, which is its implication that cross-product interactions can be modeled without explicitly introducing cross-product terms into the model. Figure 2 illustrates how the initial algorithm is able to accomplish the same solely by making use of the mathematical structure of the superposition equation. Figures 2a and 2b show scatter plots of the test data as viewed along the x and y input features, respectively. Also plotted in Figures 2a and 2b as solid curves are the feature transformations $\hat{h}_{1x}(x)$ and $\hat{h}_{1y}(y)$, respectively, that were constructed from the x and y inputs. Figure 2c shows a scatter plot of the test data as viewed along the additive model output $\tilde{y}_1$, together with the output of the $g_1(\tilde{y}_1)$ function plotted as a solid curve.

As can be seen in Figures 2a and 2b, the first stage feature transformations $\hat{h}_{1x}(x)$ and $\hat{h}_{1y}(y)$ extract only the linear terms in the target function. From the point of view of these transformations, the cross-product relationship appears as heteroskedastic noise. However, as shown in Figure 2c, from the point of view of the additive model output $\tilde{y}_1$, the cross-product relationship appears as residual systematic error together with lower heteroskedastic noise. This residual systematic error is modeled by the $g_1(\tilde{y}_1)$ transformation of the first boosting stage. The resulting output produces the first approximation of the cross-product interaction shown in Figure 1b. As this example illustrates, the nonlinear transformations g_i in Equation 4b (and in Kolmogorov's theorem) have the effect of modeling cross-product interactions without requiring that explicit cross-product terms be introduced into the models.





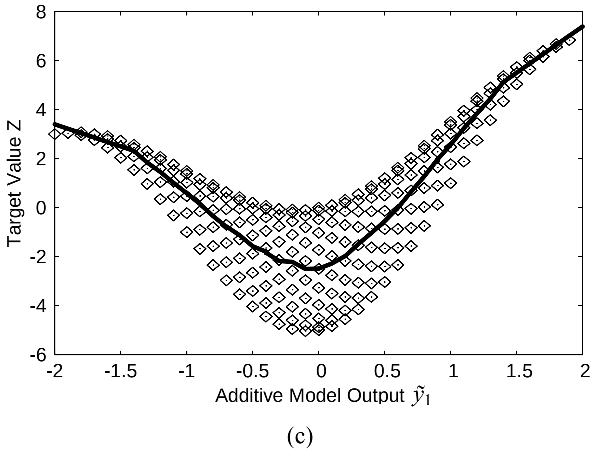


Figure 2. The test data as seen from various points in the first boosting stage. (a) Test data and $h_{1x}(x)$ plotted against the x axis. (b) Test data and $h_{1y}(y)$ plotted against the y axis. (c) Test data and $g_1(\tilde{y}_1)$ plotted against the derived $\tilde{y}_1$ axis.

5. Orthogonalized, feed-forward gradient boosting

A necessary condition that must be satisfied to maximize the rate of convergence of gradient boosting for squared-error loss is that the boosting stage outputs must be mutually orthogonal. To see why, let R_i be the output of the i'th boosting stage. Then the current model M_i obtained after i boosting iterations is given by

$$M_i = \sum_{j=1}^{i} \alpha_j R_j \quad .$$

If some of the boosting stage outputs are not mutually orthogonal, then under normal circumstances there will exist coefficients λ_i such that the model M'_i given by

$$M'_i = \sum_{j=1}^{i} \lambda_j \alpha_j R_j$$

will have a strictly smaller squared error on the training data than model M_i. These coefficients can be estimated by performing a linear regression of the boosting stage outputs. Additional boosting iterations would therefore need to be performed on model M_i just to match the squared error of M'_i. Hence, the rate of convergence will be suboptimal in this case.

To illustrate, suppose the base learner performs a univariate linear regression using one and only one input, always picking the input that yields the smallest squared error. Gradient boosting applied to this base learner produces an iterative algorithm for performing multivariate linear regression that is essentially equivalent to coordinate descent. Suppose further that we have a training set comprising two inputs, x and y, and a target value $f(x,y) = x+y$ with no noise added. If the inputs are orthogonal (i.e., if the dot product between x and y is zero), then only two boosting iterations will be needed to fit the target function to within round-off error. However, if the inputs are not orthogonal, more than two iterations will be needed, and the rate of convergence will decrease as the degree to which they are not orthogonal increases (i.e., as the projection of one input data vector onto the other increases).

In order to improve the rate of convergence, the gradient boosting method needs to be modified so as to increase the degree of orthogonality between boosting stages. One obvious approach would be to apply Gram-Schmidt orthogonalization to the boosting stage outputs. Gram-Schmidt orthogonalization is a technique used in QR decomposition and related linear regression algorithms. Its purpose is to convert simple coordinate descent into an optimal search method for linear least-squares optimization by modifying the coordinate directions of successive regressors. In the case of gradient boosting, the coordinate directions are the defined by the boosting stage outputs. However, the usual Gram-Schmidt orthogonalization procedure assumes that all coordinates are specified up front and at the start of the procedure. Gradient boosting, on the other hand, constructs coordinates dynamically in a stagewise fashion, so an incremental orthogonalization procedure is needed.

Although not as computationally efficient as the traditional Gram-Schmidt procedure, incremental orthogonalization can be readily accomplished simply by replacing the αR term in the gradient boosting algorithm shown in Table 1 with a linear regression of the current boosting stage output R and all previous boosting stage outputs. Adding this step to the procedure produces the orthogonalized gradient boosting method shown in Table 5. In the case of squared-error loss, the line search step in Table 5 to find an optimal scalar α is actually not needed because this scalar will equal one by construction. However, the line search is included in Table 5 to indicate how orthogonalized gradient boosting generalizes to arbitrary loss functions. When orthogonalized gradient boosting is applied to the univariate linear regression base learner described above, an optimum rate of convergence is achieved and the resulting overall algorithm is closely related to the QR decomposition algorithm for linear regression.

Table 5. Orthogonalized gradient boosting for squared-error loss.

```
Let the current model $M$ be zero everywhere;
Repeat until the current model $M$ does not appreciably change:
    At the $i$'th iteration, use the base learner to construct a model $R_i$ that predicts the residual error of $M$, ensuring that $R_i$ does not overfit the data;
    Use least-squares linear regression to find coefficients $\lambda_k$, $k \le i$, such that $\Sigma\lambda_k R_k$ best fits the residual error of $M$.
    Find a value for scalar $\alpha$ that minimizes the loss function (i.e., total squared error) for the model $M + \alpha \Sigma\lambda_k R_k$;
    Update $M \leftarrow M + \alpha \Sigma\lambda_k R_k$;
```

Although the orthogonalized gradient boosting method in Table 5 is expedient, it does not address the underlying source of non-orthogonality between boosting stage outputs, which is the base learner itself. A more refined solution would be to strengthen the base learner so that it produces models whose outputs are already orthogonal with respect to previous boosting stages. To accomplish that, the approach presented here involves first modifying the gradient boosting method

Table 6. Feed-forward gradient boosting for squared-error loss.

```
Let the current model $M$ be zero everywhere;
Repeat until the current model $M$ does not appreciably change:
    At the $i$'th iteration, use the base learner to construct a model $R_i$ that predicts the residual error of $M$ using all previous boosting stage outputs $R_1,\dots,R_{i-1}$ as additional inputs, while ensuring that $R_i$ does not overfit the data;
    Find a value for scalar $\alpha$ that minimizes the loss (i.e., total squared error) for the model $M + \alpha R_i$;
    Update $M \leftarrow M + \alpha R_i$;
```

shown in Table 1 so that, at each iteration, all previous boosting stage outputs are made available to the base learner as additional inputs. This modification yields the feed-forward gradient boosting method shown in Table 6.

The primary motivation for feed-forward gradient boosting is to enable the base learner to perform an implicit Gram-Schmidt orthogonalization instead of the explicit orthogonalization performed in Table 5. In particular, by making previous boosting stage outputs available to the base learner, it might be possible to modify the base learner so that it achieves a nonlinear form of Gram-Schmidt orthogonalization, in contrast to the linear orthogonalization step in Table 5. In the next section, it will be shown how such a modification can in fact be made to the additive-modeling base learner used in the initial algorithm.

An additional but no less important effect of feed-forward gradient boosting is that it further strengthens weak base learners by expanding their hypothesis spaces through function composition. If in the first iteration the base learner considers models of the form $M_1(x_1,\dots,x_d)$, then in the second iteration it will consider models of the form $M_2(x_1,\dots,x_d, M_1(x_1,\dots,x_d))$. Unless the base learner's hypothesis space is closed under this type of function composition, feed-forward gradient boosting will have the side effect of expanding the base learner's hypothesis space without modifying its mode of operation. The strengthening of the base learner produced by this expansion effect can potentially improve the accuracy of the models that are constructed.

Although feed-forward gradient boosting is motivated by orthogonality considerations, the method steps defined in Table 6 are not sufficient to guarantee orthogonality for arbitrary base learners, since some base learners might not be able to make full use of the outputs of previous boosting stages in order to achieve orthogonality. In such cases, the orthogonalized feed-forward gradient boosting method shown in Table 7 can be employed.

Table 7. Orthogonalized feed-forward gradient boosting for squared-error loss.

```
Let the current model $M$ be zero everywhere;
Repeat until the current model $M$ does not appreciably change:
    At the $i$'th iteration, use the base learner to construct a model $R_i$ that predicts the residual error of $M$ using the previous boosting stage outputs $R_1,\dots,R_{i-1}$ as additional inputs, while ensuring that $R_i$ does not overfit the data;
    Use least-squares linear regression to find coefficients $\lambda_k$, $k \le i$, such that $\Sigma\lambda_k R_k$ best fits the residual error of $M$.
    Find a value for scalar $\alpha$ that minimizes the loss function (i.e., total squared error) for the model $M + \alpha\, \Sigma\lambda_k R_k$;
    Update $M \leftarrow M + \alpha\, \Sigma\lambda_k R_k$;
```

6. The transform regression algorithm

In the case of the initial algorithm presented in Section 4, the base learner itself can be modified to take full advantage of the outputs of previous boosting stages. In particular, two modifications are made to the initial algorithm in order to arrive at the transform regression algorithm.

Because feed-forward gradient boosting permits boosting stage outputs to be used as input features to subsequent stages, the first modification is to convert the g_i functions in Equation 4 into h_{ij} functions by eliminating Equation 4b and by using the outputs of the additive models in Equations 4a as input features to all subsequent gradient boosting stages. This modification is intended mainly to simplify the mathematical form of the resulting models by exploiting the fact that feed-forward gradient boosting explicitly makes boosting stage outputs available to subsequent stages, which enables the h_{ij} functions to perform dual roles: transform the input features and transform the additive model outputs.

The second modification is to introduce multivariate h_{ij} functions by further allowing the outputs of the additive models in Equations 4a to appear as additional inputs to the h_{ij} functions in all subsequent stages. This modification is intended to push the orthogonalization of model outputs all the way down to the construction of the h_{ij} feature transformation functions. As discussed in Section 4, the ProbE linear regression tree (LRT) algorithm (Natarajan & Pednault, 2002) was used in the initial algorithm to construct the univariate g_i and h_{ij} functions that appear in Equation 4. The LRT algorithm, however, is capable of constructing multivariate linear regression models in the leaves of trees, and not just univariate linear regression models as was needed for the initial algorithm. By allowing previous boosting stage outputs to appear as regressors in the leaves of these trees, the output of each leaf model will then be orthogonal to those previous boosting stage outputs when the leaf conditions are satisfied. Hence, the overall nonlinear transformations defined by the trees will be orthogonal to the previous boosting stage outputs and any linear combination of the trees will likewise be orthogonal.

With the above changes, the mathematical form of the resulting transform regression models is given by the following system of equations:

$$\hat{y}_1 = \sum_{j=1}^{d} h_{1j}(x_j) \tag{6a}$$

$$\begin{aligned}\hat{y}_i = & \sum_{j=1}^{d} h_{ij}\left(x_j \middle| \hat{y}_1,\ldots,\hat{y}_{i-1}\right) \\ & + \sum_{k=d+1}^{d+i-1} h_{ik}\left(\hat{y}_{k-d} \middle| \hat{y}_1,\ldots,\hat{y}_{i-1}\right) \quad , \ i > 1\end{aligned} \tag{6b}$$

$$\hat{y} = \sum_{i} \hat{y}_i \quad , \tag{6c}$$

where the notation $h_{ij}\left(x_j \middle| \hat{y}_1,\ldots,\hat{y}_{i-1}\right)$ is used to indicate that function h_{ij} is meant to be a nonlinear transformation of x_j and that this transformation is allowed to vary as a function of $\hat{y}_1,\ldots,\hat{y}_{i-1}$. Likewise for the $h_{ik}\left(\hat{y}_{k-d} \middle| \hat{y}_1,\ldots,\hat{y}_{i-1}\right)$ functions that appear in Equation 6b. Note that the latter are the counterparts to the g_i functions in Equation 4b. In concrete terms, when applying the ProbE LRT algorithm to construct $h_{ij}\left(x_j \middle| \hat{y}_1,\ldots,\hat{y}_{i-1}\right)$ and $h_{ik}\left(\hat{y}_{k-d} \middle| \hat{y}_1,\ldots,\hat{y}_{i-1}\right)$, the LRT algorithm is constrained to split only on the features being transformed (i.e., x_i and $\hat{y}_{k-d}$, respectively) with all other inputs (i.e., $\hat{y}_1,\ldots,\hat{y}_{i-1}$) allowed to appear as regressors in the linear regression models in the leaves of the resulting trees. Of course, the features being transformed can likewise appear as regressors in the leaf models if they are numeric. Equation 6a corresponds to the first boosting stage while Equation 6b corresponds to all subsequent stages. Equation 6c defines the output of the overall model. The resulting algorithm is summarized in Table 8.

Although Equation 6 departs from the mathematical form of Kolmogorov's superposition equation, the above modifications dramatically improve the rate of convergence of the resulting algorithm. Figure 3 illustrates the increased rate of convergence of the transform regression algorithm compared to the initial algorithm when transform regression is applied to the same data as in Figures 1 and 2. In this experiment, the ProbE linear regression tree (LRT) algorithm was again used, this time exploiting its ability to construct multivariate regression models in the leaves of trees. As with the initial algorithm, one-pass greedy additive modeling was used with stepwise linear regression, with the holdout set used to estimate generalization error in order to avoid overfitting.

Table 8. The transform regression algorithm.

```
Let the current model $M$ be zero everywhere;
Repeat until the current model $M$ does not appreciably change:
    At the $i$'th iteration, construct an additive model $\hat{y}_i$ that that predicts the residual error of $M$ using both the raw features $x_1,\ldots,x_d$ and the outputs $\hat{y}_1,\ldots,\hat{y}_{i-1}$ of all previous boosting stages as potential input features to $\hat{y}_i$, allowing the feature transformation to vary as a function of the previous boosting stage outputs $\hat{y}_1,\ldots,\hat{y}_{i-1}$, and ensuring that $\hat{y}_i$ does not overfit the data;
    Update $M \leftarrow M + \hat{y}_i$;
```

As shown in Figure 3a, because the g_i functions have been removed, the first stage of transform regression extracts the two linear terms of the target function, but not the cross-product term. As can be seen in Figure 3d, the first boosting stage therefore has a higher approximation error than the first boosting stage of the initial algorithm. However, for all subsequent boosting stages, transform regression outperforms the initial algorithm, as can be seen in Figures 3b-d. As this example demonstrates, by modifying the gradient boosting algorithm and the base learner to achieve orthogonality of gradient boosting stage outputs, a dramatic increase the rate of convergence can be obtained.

7. Experimental evaluation

Table 9 shows evaluation results that were obtained on eight data sets that were used to compare the performance of the transform regression algorithm to the underlying LRT algorithm. Also shown are results for the first gradient boosting stage of transform regression, and for the stepwise linear regression algorithm that is used both in the leaves of linear regression trees and in the greedy one-pass additive modeling method. The first four data sets are available from the UCI Machine Learning Repository and the UCI KDD Archive. The last four are internal IBM data sets.

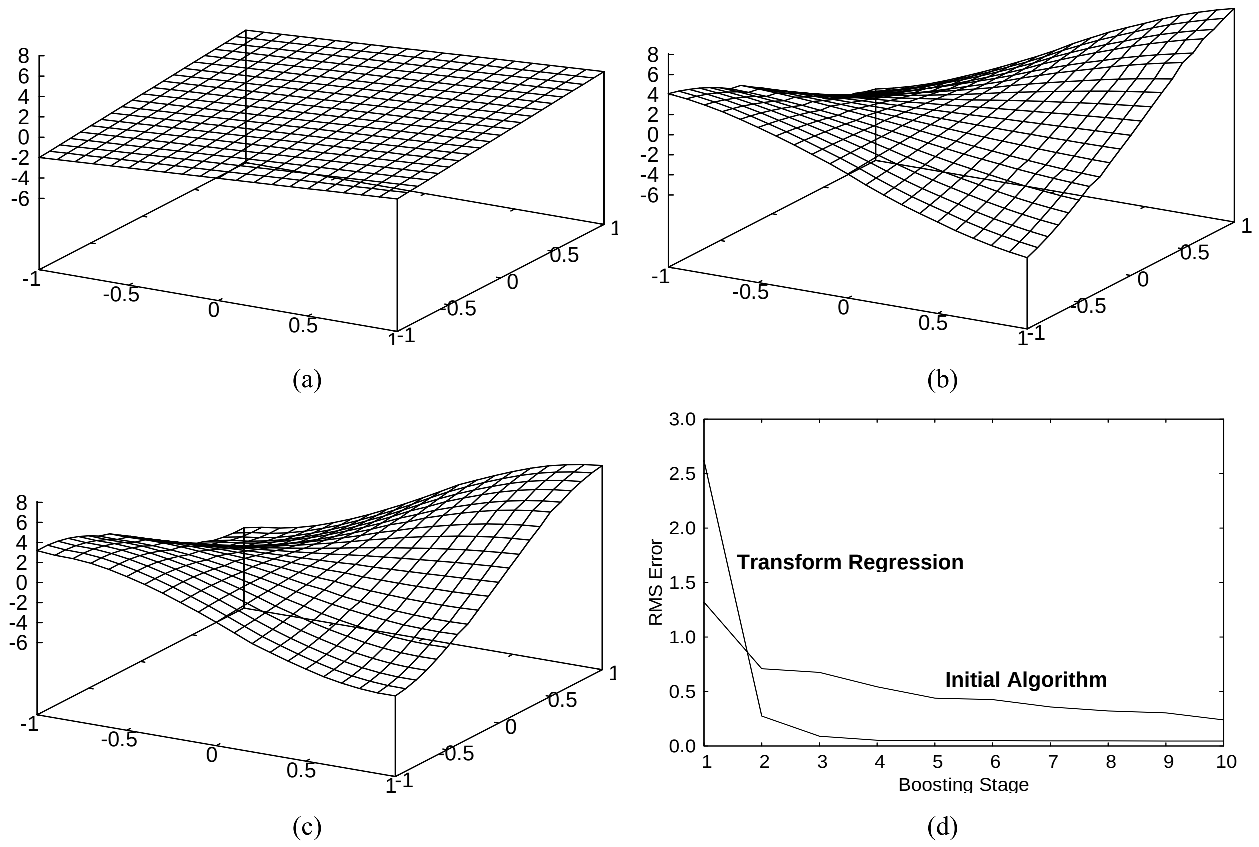


Figure 3. An example of the modeling behavior of the transform regression algorithm. (a) Model output after one gradient boosting stage. (b) After two stages. (c) After three stages. (d) RMS errors of successive gradient boosting stages.

Because all data sets have nonnegative target values, and because all but one (i.e., KDDCup98 TargetD) have 0/1 target values, comparisons were made based on Gini coefficients of cumulative gains charts (Hand, 1997) estimated on holdout test sets. Cumulative gains charts (a.k.a., lift curves) are closely related to ROC curves, except that gains charts have the benefit of being applicable to continuous nonnegative numeric target values in addition to 0/1 categorical values. Gini coefficients are normalized areas under cumulative gains charts, where the normalization produces a value of zero for models that are no better than random guessing, and a value of one for perfect predictors. Gini coefficients are thus closely related to AUC measurements (i.e., the areas under ROC curves). Note, however, that models that are no better than random guessing have an AUC of 0.5 but a Gini coefficient of zero.

Table 9. Gini coefficients for different data sets and algorithms. For each data set, the best coefficient is highlighted in bold, the second best in italics.

DATA SET	TRANS REG	FIRST BOOST STAGE	LIN REG TREES	STEP LIN REG
ADULT	**0.655**	0.559	*0.566*	0.429
COIL	**0.431**	*0.382*	0.311	0.373
KDD-98 B	**0.217**	*0.216*	0.160	0.164
KDD-98 D	**0.157**	*0.140*	0.102	0.000
A	*0.536*	0.468	**0.541**	0.162
D	*0.536*	**0.543**	0.447	0.409
M	**0.690**	*0.682*	0.638	0.380
R	**0.508**	0.481	*0.491*	0.435

As can be seen in Table 9, transform regression produces better models than the underlying LRT algorithm on all data sets but one, and for the one exception the LRT model is only slightly better. Remarkably, the first gradient boosting stage also produces better models than the LRT algorithm on a majority of the data sets. In one case, the first stage model is also better than the overall transform regression model, which indicates an overfitting problem with the prototype implementation used for these experiments.

8. Computational considerations

In addition to its other properties, transform regression can also be implemented as a computationally efficient parallelizable algorithm Such an implementation is achieved by combining the greedy one-pass additive modeling algorithm shown in Table 3 with the ProbE linear regression tree (LRT) algorithm (Natarajan & Pednault, 2002). As per Table 3, each boosting stage is calculated in two phases. The first phase constructs initial estimates of the feature transformation functions h_{ij} and h_{ik} that appear in Equations 6a and 6b. The second phase performs a stepwise linear regression on these initial feature transformations in order to select the most predictive transformations and to estimate their scaling coefficients, as per Table 3.

The ProbE LRT technology enables the computations for the first phase to be performed using only a single pass over the data. It also enables the computations to be data-partition parallelized for scalability. The LRT algorithm incorporates a generalized version of the bottom-up merging technique used in the CHAID algorithm (Biggs, deVille, and Suen 1991; Kass 1980). Accordingly, multiway splits are first constructed for each input feature. Next, data is scanned to estimate sufficient statistics for the leaf models in each multiway split. Finally, leaf nodes and their sufficient statistics are merged in a bottom-up pairwise fashion to produce trees for each feature without further accessing the data. For categorical features, the category values define the multiway splits. For numerical features, the feature values are discretized into intervals and these intervals define the multiway splits. Although the CHAID method considers only constant leaf models, the approach can be generalized to include stepwise linear regression models in the leaves (Natarajan & Pednault, 2002). In the case of linear regression, the sufficient statistics are mean vectors and covariance matrices.

By calculating sufficient statistics simultaneously for both training data and holdout data, the tree building and tree pruning steps can be performed using only these sufficient statistics without any further data access. Linear regression tree estimates of the h_{ij} and h_{ik} feature transformation functions can therefore be calculated using only a single pass over both the training and holdout data at each iteration. In addition, because the technique of merging sufficient statistics can be applied to any disjoint data partitions, the same merging method used during tree building can be used to merge sufficient statistics that are calculated in parallel on disjoint data partitions (Dorneich et al., 2006). This merging capability enables data scans to be readily parallelized.

In the first phase, the stepwise linear regression models that appear in the leaves of the feature transformation trees are relatively small. At each iteration, the maximum number of regressors is equal to the iteration number. In the second phase, no trees are constructed but a large stepwise linear regression is performed instead. In this case, the number of regressors is equal to the number of transformation trees (i.e., the number of input features plus the iteration index minus one). As with the first phase, the mean vectors and covariance matrices that define the sufficient statistics for the linear regression can be calculated using only a single pass over the training and holdout data. The sufficient statistics can likewise be calculated in parallel on disjoint data partitions, with the results then merged using the same merging technique used for tree building.

The above implementation techniques produce a scalable and efficient algorithm. These techniques have been incorporated into a parallelized version of transform regression that is now available in IBM DB2 Intelligent Miner Modeling, which is IBM's database-embedded data mining product (Dorneich et al., 2006).

9. Conclusions

Although the experimental results presented above are by no means an exhaustive evaluation, the consistency of the results clearly demonstrate the benefits of the global function-fitting approach of transform regression compared to the local fitting approach of the underlying linear regression tree (LRT) algorithm that is employed. Transform regression uses the LRT algorithm to construct a series of global functions that are then combined using linear regression. Although this use of LRT is highly constrained, in many cases it enables better models to be constructed than with the pure local fitting of LRT. In this respect, transform regression successfully combines the global-fitting aspects of learning methods such as neural networks with the nonparametric local-fitting aspects of decision trees.

Transform regression is also computationally efficient. Only two passes over the data are required to construct each boosting stage: one pass to build linear regression trees for all input features to a boosting stage, and another pass to perform the stepwise linear regression that combines the outputs of the resulting trees to form an additive model. The amount of computation that is required per boosting stage is therefore between one to two times the amount of computation needed by the LRT algorithm to build a single level of a conventional linear regression tree when the LRT algorithm is applied outside the transform regression framework.

Another aspect of transform regression is that it demonstrates how Friedman's gradient boosting framework can be enhanced to strengthen the base learner and improve the rate of convergence. One enhancement is to use the outputs of boosting stages as first-class input features to subsequent stages. This modification has the effect of expanding the hypothesis space through function composition of boosting stage models. Another enhancement is to modify the base learner so that it produces models whose outputs are linearly orthogonal to all previous boosting stage outputs. This orthogonality property improves the efficiency of the gradient descent search performed by the boosting algorithm, thereby increasing the rate of convergence. In the case of transform regression, this second modification involved using boosting stage outputs as additional multivariate inputs to the feature transformation functions h_{ij} and h_{ik}. This very same approach can likewise be used in combination with Friedman's tree boosting algorithm by replacing his conventional tree algorithm with the LRT algorithm. It should likewise be possible to extend the approach presented here to other tree-based boosting techniques.

Transform regression, however, is still a greedy hill-climbing algorithm. As such, it can get caught in local minima and at saddle points. In order to avoid local minima and saddle points entirely, additional research is needed to further improve the transform regression algorithm. Several authors (Kůrková, 1991, 1992; Neruda, Štědrý, & Drkošová, 2000; Sprecher 1996, 1997, 2002) have been investigating ways of overcoming the computational problems of directly applying Kolmogorov's theorem. Given the strength of the results obtain here using the form of the superposition equation alone, research aimed at creating a combined approach could potentially be quite fruitful.

References

Biggs, D., de Ville, B., and Suen, E. (1991). A method of choosing multiway partitions for classification and decision trees. *Journal of Applied Statistics*, 18(1):49-62.

Dorneich, A., Natarajan, R., Pednault, E., & Tipu, F. (2006). Embedded predictive modeling in a parallel relational database, to appear in *Proceedings of the 21st ACM Symposium on Applied Computing*, April 2006, Dijon, France.

Friedman, J.H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics* **29**(5):1189-1232.

Friedman, J.H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis* **38**(4):367-378.

Girosi, F. & Poggio, T. (1989). Representation properties of networks: Kolmogorov's theorem is irrelevant. *Neural Computation* **1**(4):465-469.

Hand, D.J. (1997). *Construction and Assessment of Classification Rules.* New York: John Wiley and Sons.

Hastie, T. & Tibshirani, R. (1990). *Generalized Additive Models*. New York: Chapman and Hall.

Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem. *Proc. IEEE International Conference on Neural Networks, Vol. 3*, 11-14.

Jiang, W. (2001). Is regularization unnecessary for boosting? *Proc. 8th Intl. Workshop on AI and Statistics*, 57-64. San Mateo, California: Morgan Kaufmann.

Jiang, W. (2002). On weak base hypotheses and their implications for boosting regression and classification. *Annals of Statistics* **30**(1):51-73.

Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics* 29(2):119-127.

Kolmogorov, A.N. (1957). On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, **144**(5):953-956. Translated in *American Mathematical Society Translations Issue Series 2*, **28**:55-59 (1963).

Kůrková, V. (1991). Kolmogorov's theorem is relevant. *Neural Computation* **3**(4):617-622.

Kůrková, V. (1992). Kolmogorov's theorem and multilayer neural networks. *Neural Networks* **5**(3):501-506.

Lorentz, G.G. (1962). Metric entropy, widths, and superposition of functions. *American Mathematical Monthly*, **69**:469-485.

Mallat, S.G. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Trans. Signal Processing* **41**(12):3397-3415.

Natarajan, R. & Pednault, E.P.D. (2002). Segmented regression estimators for massive data sets. *Proc. Second SIAM International Conference on Data Mining*, available online at www.siam.org.

Neruda, R., Štědrý, A., & Drkošová J. (2000). Towards feasible learning algorithm based on Kolmogorov theorem. *Proc. International Conference on Artificial Intelligence, Vol. II*, pp. 915-920. CSREA Press.

Sprecher, D.A. (1965). On the structure of continuous functions of several variables. *Transactions American Mathematical Society*, **115**(3):340-355.

Sprecher, D.A. (1996). A numerical implementation of Kolmogorov's superpositions. *Neural Networks* **9**(5):765-772.

Sprecher, D.A. (1997). A numerical implementation of Kolmogorov's superpositions II. *Neural Networks* **10**(3):447-457.

Sprecher, D.A. (2002). Space-filling curves and Kolmogorov superposition-based neural networks. *Neural Networks* **15**(1):57-67.