

# 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  
*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 with 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 outside the transform regression framework.

## 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 approach draws inspiration from the Kolmogorov superposition theorem [1]:

**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  $f(x_1, \dots, x_d)$  defined on the  $d$ -dimensional unit hypercube  $U^d$ , there exist real continuous functions  $g_i(x)$  such that

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

Stronger versions of this theorem have also been reported [2,3]. Hecht-Nielson [3] 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. This suggestion, however, has been criticized [5] 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 neural network methods with decision-tree methods.

In the sections that follow, an initial algorithm is presented 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 is inspired by, but is not mathematically based upon, Kolmogorov’s theorem. Improvements to the initial algorithm are then presented to achieve a much faster rate of convergence. Evaluation results are also presented that compare the performance of the transform regression algorithm to the underlying decision-tree method that is employed.

## 2 An initial algorithm

Inspiration for the initial algorithm presented in this section is based on interpreting Kolmogorov’s superposition equation as a gradient boosting model [6,7] in which the “base learner” constructs generalized additive models [8] 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  $x_1, \dots, x_d$  given a set of training vectors  $\{(x_1, \dots, x_d, y)\}$ . As a first attempt, we might try constructing a generalized additive model of the form

$$\tilde{y}_1 = \sum_{j=1}^d h_{1j}(x_j) . \tag{1}$$

This modeling task could be performed, for example, using Hastie and Tibshirani’s iterative backfitting algorithm [8]. Backfitting assumes the availability of a nonparametric learning technique, called a “smoother,” for estimating univariate functions. With backfitting, a smoother would be repeatedly applied to successive input variables to iteratively (re)estimate the  $h_{1j}$  functions until convergence is achieved. A faster, though weaker, approach would be to use a greedy one-pass method that independently estimates univariate

models  $\hat{h}_{1j}$  for each input feature (i.e., without iterative backfitting) and then combines the outputs of these models using linear regression to obtain a model of the form

$$\tilde{y}_1 = \sum_{j=1}^d h_{1j}(x_j) = \lambda_{10} + \sum_{j=1}^d \lambda_{1j} \hat{h}_{1j}(x_j) = \sum_{j=1}^d \left( \lambda_{1j} \hat{h}_{1j}(x_j) + \frac{1}{d} \lambda_{10} \right). \quad (2)$$

Independent of which of the above methods is used for generalized additive modeling, systematic errors can still appear in the relationship between the additive model output  $\tilde{y}_1$  and the target value  $y$ . To remove such errors, the same nonparametric learning technique (i.e., smoother) can again be applied, this time to linearize  $\tilde{y}_1$  with respect to  $y$ . The resulting combined model will then have the form

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

To further improve the model, Friedman's gradient boosting method [6,7] can be applied by using the above two-stage modeling technique as the base learner. The resulting gradient boosting model will then have the form

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

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

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

Equations 4a and 4b define the gradient boosting stages. Equation 4a defines the generalized additive models  $\tilde{y}_i$  that are constructed in each boosting stage. The  $h_{ij}$  functions could be estimated using either backfitting or the greedy one-pass approximation described above—Equation 4a reflects both choices. Equation 4b defines the boosting stage outputs  $\hat{y}_i$ , which are nonlinear transformations of the corresponding additive model outputs  $\tilde{y}_i$ . Equation 4c defines the output  $\hat{y}$  of the overall model, which is the sum of the boosting stage outputs  $\hat{y}_i$ .

The gradient boosting model in Equation 4 is trained one boosting stage at a time. For the first stage ( $i = 1$ ), the original target values that appear in the training data are used to first learn an additive model (Equation 4a) and then a nonlinear transformation of its output (Equation 4b). In each successive boosting stage ( $i > 1$ ), the target values that are used are the differences between the original target values  $y$  and the sum  $\hat{y}_1 + \dots + \hat{y}_{i-1}$  of the outputs of the previous boosting stages.

Note that the above modeling approach defines a class of algorithms that generate predictive models that have the same mathematical form as Kolmogorov's superposition equation. However, the  $g_i$  and  $h_{ij}$  functions that appear in these models will clearly not have the forms defined in the proofs of the various versions of the superposition theorem. Instead, their forms would depend on the nonparametric learning method (i.e., smoother) that is used to estimate univariate functions.

In the experiments reported below, the ProbE linear regression tree algorithm [9] was used for univariate function estimation, the one-pass greedy approach to additive modeling was used in conjunction with stepwise linear regression, and a holdout validation set was used in both of these methods to estimate generalization error in order to avoid overfitting. To

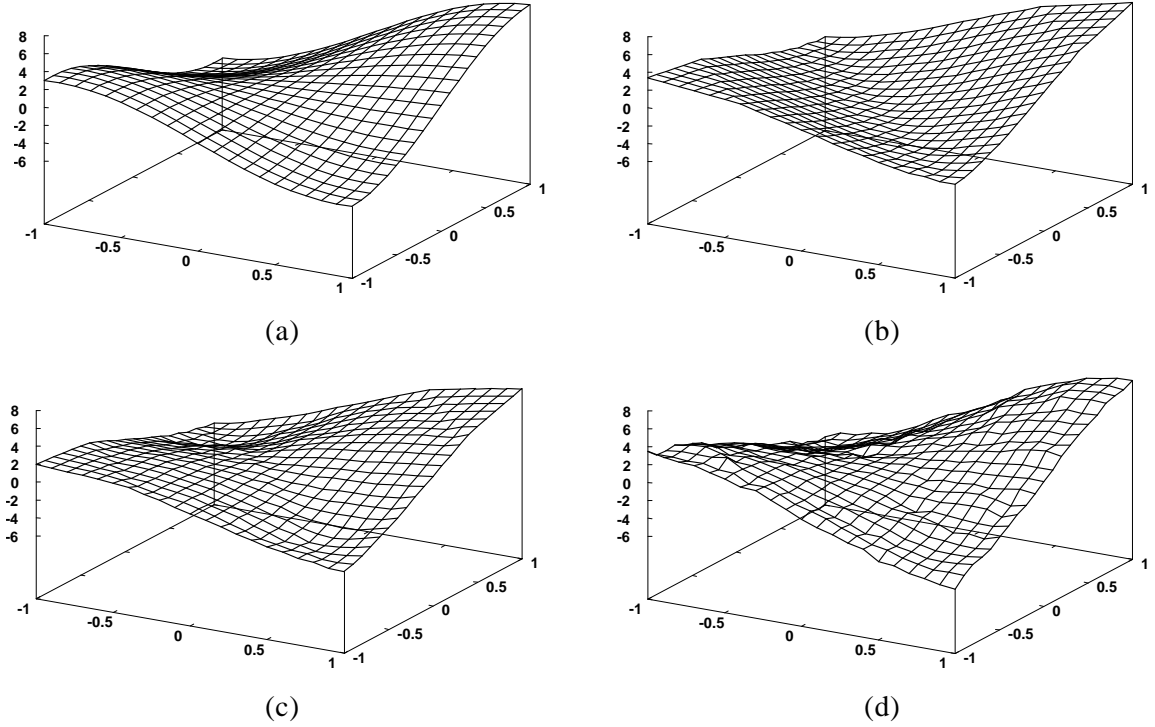


Figure 1: An example of the modeling behavior of the initial algorithm.  
(a) The target function. (b) The model output after one gradient boosting stage.  
(c) After two boosting stages. (d) After ten boosting stages.

demonstrate the behavior of the resulting concrete algorithm, the algorithm was applied to synthetic training data that was generated using the following target function:

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

Synthetic 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 divided into a training set and a holdout validation set for model pruning purposes to avoid overfitting.

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 term 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.

Figure 2 illustrates how the initial algorithm is able to model cross-product interactions without explicitly introducing cross-product terms into the model. Figures 2a and 2b show scatter plots of the test data as viewed along the  $x$  and  $y$  axes, respectively. Also plotted in Figures 2a and 2b as solid curves are the feature transformations  $\hat{h}_{1x}(x)$  and  $\hat{h}_{1y}(y)$  constructed from the  $x$  and  $y$  inputs, respectively. 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, only the linear terms in the target function are extracted by the first stage feature transformations  $\hat{h}_{1x}(x)$  and  $\hat{h}_{1y}(y)$ . From the point of view

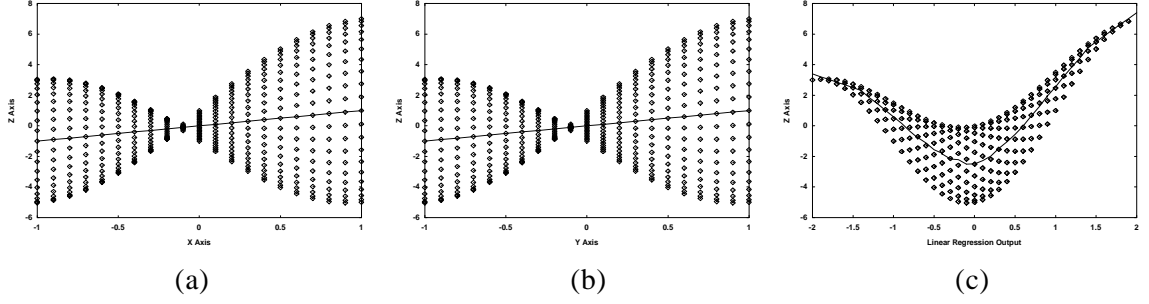


Figure 2: The test data as seen from various points in the first boosting stage. (a) Test data and  $\hat{h}_{1,x}(x)$  plotted against the  $x$  axis. (b) Test data and  $\hat{h}_{1,y}(y)$  plotted against the  $y$  axis. (c) Test data and  $g_1(\hat{y}_1)$  plotted against the derived  $\hat{y}_1$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

of these transformations, the cross-product relationship appears only as heteroskedastic noise. However, as shown in Figure 2c, from the point of view of the additive model output  $\hat{y}_1$ , the cross-product relationship appears as residual systematic error together with lower heteroskedastic noise. This residual systematic error is modeled by the  $g_1(\hat{y}_1)$  transformation, which gives rise to the first boosting stage 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) are essential for modeling cross-product interactions.

### 3 The transform regression algorithm

To improve the rate of convergence, two modifications are made to the above algorithm in order to arrive at the transform regression algorithm. The first modification is to convert the  $g_i$  functions into  $h_{ij}$  functions by eliminating Equation 4b and by using the outputs of the additive models in Equations 4a as first-class input features to all subsequent gradient boosting stages. 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. With these changes, the mathematical form of the resulting transform regression models is given by the following system of equations:

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

$$\hat{y}_i = \sum_{j=1}^d h_{ij}(x_j | \hat{y}_1, \dots, \hat{y}_{i-1}) + \sum_{k=d+1}^{d+i-1} h_{ik}(\hat{y}_{k-d} | \hat{y}_1, \dots, \hat{y}_{i-1}) \quad , \quad i > 1 \quad (6b)$$

$$\hat{y} = \sum_i \hat{y}_i \quad , \quad (6c)$$

where the notation  $h_{ij}(x_j | \hat{y}_1, \dots, \hat{y}_{i-1})$  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  $\hat{y}_1, \dots, \hat{y}_{i-1}$ . Likewise for the  $h_{ik}(\hat{y}_{k-d} | \hat{y}_1, \dots, \hat{y}_{i-1})$  functions that appear in Equation 6b. The latter are the counterparts to the  $g_i$  functions in Equation 4b. Equation 6a corresponds to the first boosting stage while Equation 6b corresponds to all subsequent stages. Because Equation 6b requires multivariate transformations, univariate learning techniques (i.e., smoothers) must be replaced with multivariate techniques in order to construct the additive models in this equation.

Although the above changes depart from the mathematical form of Kolmogorov’s superposition equation, they improve the rate of convergence of the resulting algorithm by making better use of the information that is extracted by the gradient boosting stages defined in Equations 6a and 6b. The output of a boosting stage can be viewed as a derived feature that has been identified by the base learner as being highly predictive of the target values for that boosting stage. The first modification of using the outputs of boosting stages as first-class input features to subsequent stages enables the subsequent stages to take full advantage of the predictive power of these derived features, and hence to do a better job of modeling. The second modification of further using the outputs of boosting stages as additional multivariate inputs to the feature transformation functions  $h_{ij}$  and  $h_{ik}$  provides a supplementary mechanism for modeling cross-product interactions, in addition to the modeling capability provided by the  $h_{ik}$  functions. This use likewise contributes to doing a better job of modeling.

To obtain a concrete algorithm, the ProbE linear regression tree (LRT) algorithm [9] was again used, this time exploiting its ability to construct multivariate regression models in the leaves of trees. When transforming a given feature, the LRT algorithm was constrained to split only on that feature; however, all inputs to the feature transformation were allowed to be included in the linear regression models in the leaves of the resulting tree. As with the initial algorithm, one-pass greedy additive modeling was used with stepwise linear regression, and a holdout validation set was used to estimate generalization error in order to avoid overfitting.

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 for Figures 1 and 2. As shown in Figure 3a, because the  $g_i$  functions have been removed, the first stage of transform regression extracts the two linear terms in the target function, but not the cross-product term. The first boosting stage therefore has a higher approximation error than the first boosting stage of the initial algorithm, as can be seen in Figure 3d. However, for all subsequent boosting stages, transform regression outperforms the initial algorithm, as can be seen in Figures 3b-d. As this example demonstrates, using gradient boosting stage outputs as additional inputs to subsequent boosting stages can produce a considerable increase in the rate of convergence.

## 4 Experimental evaluation

Table 1 shows evaluation results that were obtained on eight data sets used to compare the performance of the transform regression algorithm to the underlying LRT algorithm that it employs. 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. 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 [10] that were calculated on holdout test sets.

On all data sets but one, transform regression produces better models than the underlying LRT algorithm, 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.

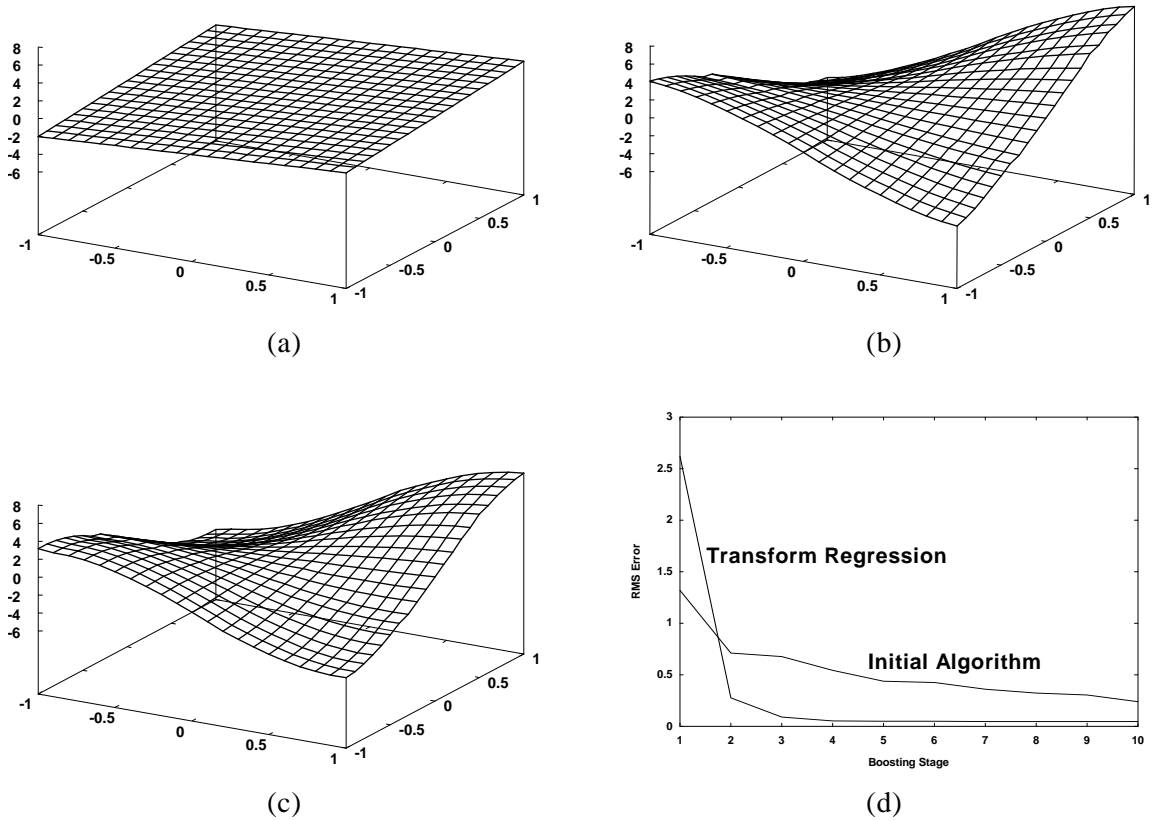


Figure 3: An example of the modeling behavior of the transform regression algorithm. (a) Model output after one gradient boosting stage. (b) After two stages. (c) After three stages. (d) RMS errors of successive gradient boosting stages.

Table 1: 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	Transform Regression	First Boosting Stage	Linear Regression Trees	Stepwise Linear Regression
Adult	<b>0.655</b>	0.559	0.566	0.429
CoIL	<b>0.431</b>	0.382	0.311	0.373
KDDCup98 B	<b>0.217</b>	0.216	0.160	0.164
KDDCup98 D	<b>0.157</b>	0.140	0.102	0.000
A	0.536	0.468	<b>0.541</b>	0.162
D	0.536	<b>0.543</b>	0.447	0.409
M	<b>0.690</b>	0.682	0.638	0.380
R	<b>0.508</b>	0.481	0.491	0.435

## 5 Conclusions

The experimental results presented above 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. Transform regression uses the LRT algorithm to construct a series of global functions that are then linearly combined. Although this use of LRT is very constrained, in many cases it enables better models to be constructed than with the pure local fitting of LRT.

Transform regression is also computationally efficient. Only two passes over the data are required to construct each boosting stage: one to build linear regression trees for all input features to a boosting stage; another 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 is between one to two times the computation needed to build the first level of a conventional linear regression tree when the LRT algorithm is applied outside the transform regression framework.

Transform regression, however, is still a greedy hill-climbing algorithm. As such, it can get caught in local minima and at saddle points. In particular, in order to model cross-product interactions, at least one of the input features that appears in an interaction must first be introduced by one gradient boosting stage in order to enable subsequent boosting stages to model the interaction. For example, in the case of Equation 5, if symmetric sampling is used to generate synthetic data as done for Figures 1-3, then at least one of the  $x$  or  $y$  terms must appear in the target function in order for the cross-product interaction to be discovered. On the other hand, if the  $x$  and  $y$  terms are dropped but asymmetric sampling is used (e.g., if only one quadrant is sampled) then the asymmetry would itself cause nonlinear transformations of  $x$  and/or  $y$  to be constructed by the first gradient boosting stage and subsequent boosting stages would then be able to model the cross-product interaction.

In order to avoid local minima and saddle points entirely, additional research is needed to further improve the transform regression algorithm. One obvious research direction is to exploit the mathematical implications of Kolmogorov's superposition theorem, and not simply the form of his equation. Several authors [11-16] have been investigating the computational aspects of directly applying Kolmogorov's theorem. Given the strength of the results obtain above using the form of the superposition equation alone, research aimed at creating a combined approach could potentially be quite fruitful.

## References

- [1] 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).
- [2] Lorentz, G.G. (1962) Metric entropy, widths, and superposition of functions. *American Mathematical Monthly*, **69**:469-485.
- [3] Sprecher, D.A. (1965) On the structure of continuous functions of several variables. *Transactions American Mathematical Society*, **115**(3):340-355.
- [4] Hecht-Nielsen, R. (1987) Kolmogorov's mapping neural network existence theorem. *Proc. IEEE International Conference on Neural Networks, Vol. 3*, 11-14.
- [5] Girosi, F. & Poggio, T. (1989) Representation properties of networks: Kolmogorov's theorem is irrelevant. *Neural Computation* **1**(4):465-469.
- [6] Friedman, J.H. (2001) Greedy function approximation: a gradient boosting machine. *Annals of Statistics* **29**(5):1189-1232.

- [7] Friedman, J.H. (2002) Stochastic gradient boosting. *Computational Statistics & Data Analysis* **38**(4):367-378.
- [8] Hastie, T. & Tibshirani, R. (1990) *Generalized Additive Models*. New York: Chapman and Hall.
- [9] 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](http://www.siam.org).
- [10] Hand, D.J. (1997) *Construction and Assessment of Classification Rules*. New York: John Wiley and Sons.
- [11] Kůrková, V. (1991) Kolmogorov's theorem is relevant. *Neural Computation* **3**(4):617-622.
- [12] Kůrková, V. (1992) Kolmogorov's theorem and multilayer neural networks. *Neural Networks* **5**(3):501-506.
- [13] Sprecher, D.A. (1996) A numerical implementation of Kolmogorov's superpositions. *Neural Networks* **9**(5):765-772.
- [14] Sprecher, D.A. (1997) A numerical implementation of Kolmogorov's superpositions II. *Neural Networks* **10**(3):447-457.
- [15] Roman, N., Arnošt, Š & Jitka, D. (2000) Towards feasible learning algorithm based on Kolmogorov theorem. *Proc. International Conference on Artificial Intelligence, Vol. II*, pp. 915-920. CSREA Press.
- [16] Sprecher, D.A. (2002) Space-filling curves and Kolmogorov superposition-based neural networks. *Neural Networks* **15**(1):57-67.