

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 $f(x_1, \dots, x_d)$ 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 (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-Nielsen (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 (Giroso 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 $f(x_1, \dots, x_d)$, 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.

<p>Let the current model M be zero everywhere;</p> <p>Repeat until the current model M does not appreciably change:</p> <ul style="list-style-type: none"> 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 α 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.

<p>Set y_0 equal to the mean of the target variable y;</p> <p>For $j = 1, \dots, d$ initialize the function $h_j(x_j)$;</p> <p>Repeat until the functions $h_j(x_j)$ do not appreciably change:</p> <p style="padding-left: 2em;">For $j = 1, \dots, d$ do the following:</p> <p style="padding-left: 4em;">Use the smoother to construct a model $H_j(x_j)$ that predicts the following target value using only input feature x_j:</p> $\text{new target} = y - y_0 - \sum_{k \neq j} h_k(x_k)$ <p style="padding-left: 4em;">Update $h_j(x_j) \leftarrow H_j(x_j)$;</p>
--

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

<p>Set y_0 equal to the mean of the target variable y;</p> <p>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;</p> <p>Calculate linear regression coefficients λ_j such that $\sum_j \lambda_j H_j(x_j)$ is a best predictor of $(y - y_0)$;</p> <p>For $j = 1, \dots, d$ set $h_j(x_j) = \lambda_j H_j(x_j)$;</p>
--

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 $\{(x_1, \dots, x_d, y)\}$. As a first attempt, we might try constructing a generalized additive model of the form

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

where

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

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

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

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

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

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

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

Equations 4a and 4b define the 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

