Near-optimal Nonlinear Regression Trees

Dimitris Bertsimas[a], Jack Dunn[b], Yuchen Wang[c]

[a] *Sloan School of Management and Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139*

[b] *Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139*

[c] *Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139*

Abstract

We propose Near-optimal Nonlinear Regression Trees with hyperplane splits (NNRTs) that use a polynomial prediction function in the leaf nodes, which we solve by stochastic gradient methods. On synthetic data, we show experimentally that the algorithm converges to the global optimal. We compare NNRTs, ORT-LH, Multivariate Adaptive Regression Splines (MARS), Random Forests (RF) and XGBoost on 40 real-world datasets and show that overall NNRTs have a performance edge over all other methods.

Keywords: Decision Trees, Regression, Nonlinear Optimization

1. Introduction

Classification and Regression Trees (CART) introduced by [1] use a greedy approach to build the trees. Starting from the root node, CART chooses a variable which best splits the data and recurses on the two resulting children nodes. Although CART provides a fast way to find the decision tree, the top-down greedy algorithm only leads to a locally optimal solution that, while very interpretable, has weaker predictive power than RF and XGBoost. [2], [3] and [4] present Optimal Classification Trees (OCT) that utilize mixed-integer optimization (MIO) and local heuristic methods to find near-optimal trees for axis-paralleled splits (OCT), but also splits with

hyperplanes (OCT-H). In this way, while maintaining interpretability, OCT have comparable predictive accuracy with RF, and OCT-H have comparable predictive accuracy with XGBoost.

For regression, CART build the tree using a similar greedy method but predicts the mean outcome in each leaf. However, unlike classification problems, a constant outcome limits the prediction power of CART. Researchers consider using other functions like linear functions to replace the constant outcome. The most straightforward approach is training a regression tree with constant predictions first and then run a linear regression in the leaves which can avoid the cost of repeatedly fitting models during training ([5]). The main weakness of this method is that it builds the decision tree and decides the prediction function separately. The tree developed in this way is usually larger than we need because we only use a constant prediction when building the tree instead of using a linear function. [6], [7] and [8] propose alternative ways. They utilize hypothesis testing to choose the most significant variables in the model and use them on the split. This method speeds up the tree-building process but it limits the prediction power, because using the most significant variable to split is not equivalent to make the optimal split when the tree becomes deeper. [9] use stepwise regression to choose two variables for each split. While it maintains interpretability, this method can not be generalized to make a hyperplane split using more variables.

[2], [3] and [4] apply the Optimal Trees methodology for classification problems to the regression problem, yielding a procedure for generating Optimal Regression Trees with constant predictions in each leaf (ORT) and Optimal Regression Trees with linear predictions in the leaves (ORT-L). In a list of 60 problems from the UCI database ORT and ORT-L give average improvements in out-of-sample R^2 of 1.4% and 21.2% over CART. Instead of using parallel splits, they also illustrate how to form Optimal Regression Trees with hyperplanes and linear predictions in the leaves (ORT-LH) that provide average improvements in out-of-sample R^2 of 23.4% over CART.

Multivariate adaptive regression splines (MARS) is a popular machine learning approach proposed by [10], which is widely used for prediction in regression problems. It can be viewed as a generalization of stepwise linear regression of a modification of the CART method to improve the performance in regression problems ([11]). Instead of predicting a constant or linear function in the leaf node, MARS predicts a polynomial function using the value on each split.

In this paper, we combine ideas from MARS and ORT-LH to propose a new method we call Near-optimal Nonlinear Regression Trees with Hyperplanes (NNRTs), where we develop a tree with hyperplanes, but the prediction in the leaves uses a polynomial function instead of a linear one. We formulate the process of building such a tree directly as a nonlinear unconstrained optimization problem with a differentiable objective function and use gradient methods to solve the underlying optimization problem we propose. Specifically, our contributions in this paper include:

1. We present a novel formulation of the regression tree problem and present a gradient method to solve it.
2. We demonstrate that NNRTs increase out-of-sample R^2 by 0.043 (a 6.1% improvement) on average compared to MARS in 40 datasets from UCI machine learning Repository ([12]). Furthermore, NNRTs increase out-of-sample R^2 by 0.013 (a 1.8% improvement) on average over ORT-LH in the same 40 datasets.
3. We further show that NNRTs yield 0.017 (a 2.3% improvement) and 0.013 (a 1.8% improvement) in the out-of-sample R^2 compared to random forests and gradient-boosted trees in the same 40 datasets, respectively.

The paper is structured as follows. In Section 2, we describe how to formulate NNRTs as a nonlinear optimization problem. In Section 3, we present our algorithm to solve the optimization problem. In Section 4, we show experimentally that the algorithm converges to the global optimal. We also illustrate the regularization term avoids overfitting caused by using polynomial prediction functions. In Section 5, we present computational results on 40 datasets from UCI machine learning Repository and conclude in Section 6.

2. Near-optimal Nonlinear Regression Trees with Hyperplanes

We are given data $(\mathbf{x}_i, y_i), i \in [n] = \{1, \ldots, n\}$ with $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$. By applying the transformation

$$\hat{x}_{ij} := \frac{x_{ij} - x_{j,\min}}{x_{j,\max} - x_{j,min}},\ i \in [n],\ j \in [m],$$

where $x_{j,\max} = \max_{i\in[n]} x_{ij}$ and $x_{j,\min} = \min_{i\in[n]} x_{ij}$, we can assume that $\mathbf{x}_i \in [0,1]^m$.

A decision tree T is comprised of two parts, the set of nodes $N = \{1, 2, \ldots, t\}$ and the set of directed arcs $A = \{(i, j) : i, j \in N\}$. Arc $(i, j) \in A$ represents the directed arc from parent node i to child node j. If j is the left (right) child of i, we define (i, j) as a left (right) arc. We let $\mathcal{L}_p$ represent the set of nodes from the root of the tree to node p and we have $\mathcal{L}_p = \mathcal{L}_p^{<} \cup \mathcal{L}_p^{\geq}$, where $\mathcal{L}_p^{<}$ is the set of ancestors of p whose left branch has been followed on the path from root node p and $\mathcal{L}_p^{\geq}$ is the set of right-branch ancestors.

In an NNRT T of depth D, there are two groups of parameters:

- Vector $\mathbf{a}_\ell$ and scalar b_ℓ associated with each node ℓ in the tree with the interpretation that for a given sample point $\mathbf{x}$, if $\mathbf{a}_\ell^T \mathbf{x} < b_\ell$, then sample point $\mathbf{x}$ traverses the left arc, otherwise it traverses the right arc.
- The polynomial prediction function $g_p(\mathbf{x})$ for each leaf node p in T. We let $\mathcal{L}_p = (\ell_1, \ldots, \ell_D)$ $(\ell_D = p)$. $g_p(\mathbf{x})$ is of the form

$$g_p(\mathbf{x}) = f_{p,D+1} + \sum_{i=1}^{D} f_{p,i} \prod_{\ell=\ell_1}^{\ell_i} |\mathbf{a}_\ell^T \mathbf{x} - b_\ell|. \tag{1}$$

Given an input $\mathbf{x}$, the overall prediction is given by:

$$g(\mathbf{x}) = \sum_{p \in L} g_p(\mathbf{x}) \prod_{\ell \in \mathcal{L}_p^{<}} \mathbb{1}\left\{ {\mathbf{a}_\ell}^T \mathbf{x} < b_\ell \right\} \prod_{\ell \in \mathcal{L}_p^{\geq}} \mathbb{1}\left\{ {\mathbf{a}_\ell}^T \mathbf{x} \geq b_\ell \right\}, \tag{2}$$

where L is the set of all leaf nodes.

We call splits ${\mathbf{a}_\ell}^T \mathbf{x} < b_\ell$ univariate when they involve only one variable, otherwise, we call them multivariate. Based on the type of

- Split: Univariate, multivariate
- Prediction function: Constant, linear or polynomial function
- Method of calculation: Greedy, optimal

we summarize the decision tree methods for regression in Table 1.

In this paper, we propose NNRTs with multivariate splits and polynomial prediction functions. Figure 1 shows an NNRT of depth two.

Method Name	Split	Outcome	Solution	Reference
CART	Univariate	Constant	Greedy	[1]
M5	Univariate	Linear	Greedy	[5]
MARS	Univariate	Polynomial	Greedy	[10]
ORT	Univariate	Constant	Optimal	[3]
ORT-L	Univariate	Linear	Optimal	[3]
ORT-LH	Multivariate	Linear	Optimal	[3]
NNRT	Multivariate	Polynomial	Near-optimal	This paper

Table 1: List of Decision Tree Methods for Regression.

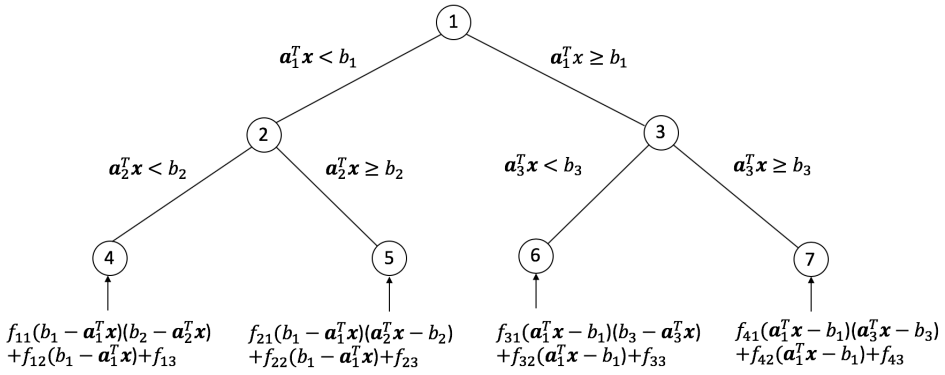

Figure 1: An NNRT of depth two.

2.1. Formulating NNRTs as a nonlinear optimization problem

In this section, we formulate NNRTs as a nonlinear optimization problem. Since

$$\begin{aligned} |\mathbf{a}_\ell^T \mathbf{x} - b_\ell| \mathbb{1}\left\{ {\mathbf{a}_\ell}^T \mathbf{x} < b_\ell \right\} &= \max(b_\ell - {\mathbf{a}_\ell}^T \mathbf{x}, 0), \\ |\mathbf{a}_\ell^T \mathbf{x} - b_\ell| \mathbb{1}\left\{ {\mathbf{a}_\ell}^T \mathbf{x} \geq b_\ell \right\} &= \max({\mathbf{a}_\ell}^T \mathbf{x} - b_\ell, 0), \end{aligned} \tag{3}$$

we substitute $g_p(\mathbf{x})$ to Eq (2) and obtain

$$\begin{aligned} g(\mathbf{x}) = \sum_{p \in L} \Bigg(\sum_{i=1}^{D} f_{p,i} \prod_{\ell \in \mathcal{L}_{\ell_i}^{<}} \max(b_\ell - {\mathbf{a}_\ell}^T \mathbf{x}, 0) \\ \prod_{\ell \in \mathcal{L}_{\ell_i}^{\geq}} \max({\mathbf{a}_\ell}^T \mathbf{x} - b_\ell, 0) + f_{p,D+1} \Bigg). \end{aligned} \tag{4}$$

Given data $(\mathbf{x}_i, y_i),\ i \in [n]$ we propose to determine the NNRT by solving the optimization problem:

$$\underset{\mathbf{a},\mathbf{b},\mathbf{f}}{\text{minimize}} \quad L(\mathbf{a}, \mathbf{b}, \mathbf{f}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - g(\mathbf{x}_i))^2 + \lambda \left(\|\mathbf{a}\|_2^2 + \|\mathbf{b}\|_2^2 + \|\mathbf{f}\|_2^2 \right). \tag{5}$$

where $\mathbf{a}$ is a concatenation of vectors $\mathbf{a}_\ell$, $\mathbf{b}$ is a concatenation of scalar $\mathbf{b}_\ell$, $\mathbf{f}$ is a concatenation of all coefficients f. Note that we have added a quadratic regularization term motivated by robustness considerations (see [13]) and we have used a quadratic loss. Note further that we use two hyper-parameters: the depth of the tree D and the complexity parameter λ.

3. Methods to find NNRTs

In this section, we present gradient descent Algorithm 1 to solve problem (5). Before discussing the details of Algorithm 1, we note that if the values of $\mathbf{a}_\ell$ and b_ℓ for all ℓ are fixed, we can find the optimal $\mathbf{f}$ using ridge regression. This is because when the structure of the tree is fixed, we know the leaf node p_i each $\mathbf{x}_i$ belongs to. According to Eq. (1), the outcome for $\mathbf{x}_i$ is given by:

$g_{p_i}(\mathbf{x}_i) = f_{p_i,D+1} + \sum_{j=1}^{D} f_{p_i,j} \prod_{\ell \in \mathcal{L}_{\ell_j}} |\mathbf{a}_\ell^T \mathbf{x}_i - b_\ell|$. So the optimization problem (5) can be viewed as a ridge regression:

$$\underset{\mathbf{f}}{\text{minimize}} \quad L(\mathbf{f}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f_{p_i,D+1} - \sum_{j=1}^{D} f_{p_i,j} \prod_{\ell \in \mathcal{L}_{\ell_j}} |\mathbf{a}_\ell^T \mathbf{x}_i - b_\ell|)^2 + \lambda \|\mathbf{f}\|_2^2 \tag{6}$$

To start Algorithm 1, we need to find the initial value $\mathbf{a}^{[1]}$, $\mathbf{b}^{[1]}$ and $\mathbf{f}^{[1]}$. We randomly generate M times $\mathbf{a}$, $\mathbf{b}$ and calculate the corresponding $\mathbf{f}$ by solving problem (6). Then we find one group of parameters $\mathbf{a}, \mathbf{b}, \mathbf{f}$ with smallest objective value $L(\mathbf{a}, \mathbf{b}, \mathbf{f})$ as the starting point.

Considering Problem (5) is a non-convex optimization problem with a differentiable objective function, the most widely used method is gradient descent such as stochastic Gradient Descent (SGD) ([14], [15]) and its variant Adaptive Moment Estimation (Adam) ([16]). Based on experimental results, we choose Adam to update the parameter $\mathbf{a}^{[t]}$ and $\mathbf{b}^{[t]}$. After that, we update the parameter $\mathbf{f}^{[t]}$ by solving the problem (6). We have found experimentally that this works better than updating $\mathbf{a}^{[t]}$, $\mathbf{b}^{[t]}$ and $\mathbf{f}^{[t]}$ at the same time.

There are two parameters we need to choose for the above algorithm, the depth of the tree D and the complexity parameter λ. A complete process for tuning the parameters using a validation set is described in Algorithm 2.

4. Performance on Synthetic Datasets

In this section, we investigate experimentally (a) whether the algorithm converges to the ground truth we generate using synthetic datasets; (b) whether the regularization term avoids overfitting.

To investigate question (a), we generate various nonlinear regression trees with hyperplane splits as ground truths based on different values of (D, n, m):

- Depth of tree D: 2, 3, 4
- Size of the data n: 10,000, 100,000

Algorithm 1 NNRT

Initialize $\mathbf{a}^{[\mathbf{1}]}$, $\mathbf{b}^{[\mathbf{1}]}$ and $\mathbf{f}^{[\mathbf{1}]}$.
while $|L(\mathbf{a}^{[\mathbf{t-1}]}, \mathbf{b}^{[\mathbf{t-1}]}, \mathbf{f}^{[\mathbf{t-1}]}) - L(\mathbf{a}^{[\mathbf{t}]}, \mathbf{b}^{[\mathbf{t}]}, \mathbf{f}^{[\mathbf{t}]})| \geq \epsilon$ **do**
    $t \leftarrow t + 1$
    $g_{t,1} = \nabla_{\mathbf{a}^{[\mathbf{t}]}} L(\mathbf{a}^{[\mathbf{t}]}, \mathbf{b}^{[\mathbf{t}]}, \mathbf{f}^{[t]}), g_{t,2} = \nabla_{\mathbf{b}^{[\mathbf{t}]}} L(\mathbf{a}^{[\mathbf{t}]}, \mathbf{b}^{[\mathbf{t}]}, \mathbf{f}^{[t]})$ (Get gradients)
    $m_{t,1} = \beta_1 m_{t-1,1} + (1-\beta_1) g_{t,1}, m_{t,2} = \beta_1 m_{t-1,2} + (1-\beta_1) g_{t,2}$ (Update biased first moment)
    $\hat{m}_{t,1} = m_{t,1}/(1-\beta_1^t), \hat{m}_{t,2} = m_{t,2}/(1-\beta_1^t)$ (Get bias-corrected first moment)
    $v_{t,1} = \beta_2 v_{t-1,1} + (1-\beta_2) g_{t,1}^2, v_{t,2} = \beta_2 v_{t-1,2} + (1-\beta_2) g_{t,2}^2$ (Update biased second moment)
    $\hat{v}_{t,1} = v_{t,1}/(1-\beta_2^t), \hat{v}_{t,2} = v_{t,2}/(1-\beta_2^t)$ (Get bias-corrected second moment)
    $\mathbf{a}^{[\mathbf{t}]} = \mathbf{a}^{[\mathbf{t-1}]} - \alpha \hat{m}_{t,1}/(\sqrt{\hat{v}_{t,1}} + \epsilon)$ (Update the parameter $\mathbf{a}$)
    $\mathbf{b}^{[\mathbf{t}]} = \mathbf{b}^{[\mathbf{t-1}]} - \alpha \hat{m}_{t,2}/(\sqrt{\hat{v}_{t,2}} + \epsilon)$ (Update the parameter b)
    Use ridge regression to calculate the value of $\mathbf{f}^{[\mathbf{t}]}$ based on the value of $\mathbf{a}^{[\mathbf{t}]}$ and $\mathbf{b}^{[\mathbf{t}]}$.
end while

Algorithm 2 Tuning the parameters for NNRTs

Set the maximum depth of NNRT D_{max} and possible complexity parameters $\{\lambda_1, \ldots, \lambda_j\}$.
for $D = 1, \ldots, D_{max}$ **do**
    for $\lambda = \lambda_1, \ldots, \lambda_j$ **do**
        Find NNRT with depth D using complexity parameter λ.
        Add this NNRT to the solution pool.
    end for
end for
Identify the solution with largest R^2 on the validation set.

D	n	m	In-sample R^2	Out-of-sample R^2	Time (seconds)
2	10000	10	0.998 (0.995,1)	0.998 (0.996,1)	24.6 (16.5,32.7)
	10000	50	0.999 (0.999,0.999)	0.999 (0.999,0.999)	24.2 (22.6,25.8)
	10000	250	0.999 (0.997,1)	0.999 (0.997,1)	47 (35.3,58.7)
	100000	10	0.999 (0.998,1)	0.999 (0.998,1)	210.5 (132.7,288.3)
	100000	50	0.998 (0.995,1)	0.998 (0.995,1)	131.7 (106.3,157.1)
	100000	250	0.999 (0.997,1)	0.999 (0.997,1)	296.3 (240.0,352.6)
3	10000	10	0.998 (0.997,0.999)	0.998 (0.997,0.999)	27.4 (23.4,31.4)
	10000	50	0.999 (0.997,1)	0.999 (0.997,1)	28.9 (22.2,35.6)
	10000	250	0.999 (0.997,1)	0.999 (0.997,1)	88.9 (59.1,118.7)
	100000	10	0.997 (0.994,1)	0.997 (0.993,1)	156.6 (109.3,203.9)
	100000	50	0.999 (0.998,1)	0.999 (0.998,1)	270.7 (219.2,322.2)
	100000	250	0.998 (0.995,1)	0.998 (0.995,1)	385.7 (289.9,481.5)
4	10000	10	0.998 (0.997,0.999)	0.998 (0.997,0.999)	30 (24.7,35.3)
	10000	50	0.999 (0.999,0.999)	0.999 (0.999,0.999)	37.2 (33.3,41.1)
	10000	250	0.998 (0.995,1)	0.998 (0.995,1)	111.9 (54.7,169.1)
	100000	10	0.998 (0.997,0.999)	0.998 (0.997,0.999)	281.5 (147.6,415.4)
	100000	50	0.999 (0.998,1)	0.999 (0.998,1)	417.2 (335.5,498.9)
	100000	250	0.996 (0.992,1)	0.996 (0.992,1)	910.4 (486.8,1334)

Table 2: The training, test set R^2 and the time for different experiments to illustrate the convergence of the algorithm.

- Number of variables m: 10, 50, 250

and we report three measures of tree quality in all experiments:

- In-sample R^2 : Training set accuracy.
- Out-of-sample R^2 : Test set accuracy.
- Time (seconds) : Time to solve the problem.

In each experiment, we generate five random nonlinear trees with hyperplane as ground truths with corresponding training and test set pairs. We show the means and confidence interval of these measures. We set the complexity parameter λ to 0 and $\epsilon = 10^{-7}$. We used one random start in this experiment. Table 2 shows the result of this experiment. These experimental results suggest that NNRTs recovered the ground truth and converged to an optimal solution.

D	n	m	In-sample R^2	Out-of-sample R^2	Time (seconds)
	10000	10	0.999 (0.998,1)	0.998 (0.996,1)	20.4 (13.21,27.59)
	10000	50	0.999 (0.998,1)	0.998 (0.996,1)	24.3 (15.18,33.42)
	10000	250	0.995 (0.992,0.998)	0.994 (0.991,0.997)	28.9 (15.75,42.05)
2	100000	10	0.999 (0.998,1)	0.997 (0.994,1)	100.5 (83.32,117.68)
	100000	50	0.999 (0.998,1)	0.997 (0.994,1)	154.3 (135.89,172.71)
	100000	250	0.997 (0.994,1)	0.998 (0.996,1)	203.1 (165.76,240.44)
	10000	10	0.999 (0.998,1)	0.997 (0.994,1)	25.4 (21.19,29.61)
	10000	50	0.998 (0.996,1)	0.997 (0.994,1)	27.8 (21.66,33.94)
	10000	250	0.993 (0.990,0.996)	0.993 (0.990,0.996)	35.6 (28.76,42.44)
3	100000	10	0.999 (0.998,1)	0.998 (0.996,1)	121.1 (103.39,138.81)
	100000	50	0.998 (0.996,1)	0.997 (0.994,1)	147.6 (127.79,167.41)
	100000	250	0.996 (0.994,0.998)	0.994 (0.991,0.997)	249 (213.06,284.94)
	10000	10	0.998 (0.996,1)	0.998 (0.996,1)	32.1 (26.49,37.71)
	10000	50	0.995 (0.992,0.998)	0.994 (0.991,0.997)	33.5 (27.36,39.64)
4	10000	250	0.990 (0.985,0.995)	0.989 (0.985,0.993)	46.5 (38.26,54.74)
	100000	10	0.999 (0.998,1)	0.998 (0.996,1)	200.1 (146.63,253.57)
	100000	50	0.997 (0.994,1)	0.998 (0.996,1)	352.2 (263.84,440.56)
	100000	250	0.994 (0.990,0.998)	0.994 (0.990,0.998)	782.7 (426.65,1138.75)

Table 3: The training, test set R^2 and the time for different experiments to illustrate the use of regularization term.

We next investigate question (b), we generate various linear regression trees with hyperplane splits. In the leaf node, the ground truth is a linear function instead of a polynomial function with degree D. To be specific, we let the coefficients f in the leaf node to be 0 if the degree of the corresponding function degree is larger than 1. The target of this experiment is to test whether the regularization term avoids overfitting caused by using degree D polynomial prediction functions. Table 3 shows the result of this experiment. These experimental results suggest that the regularization term of NNRTs avoids overfitting.

Summarizing, we found that NNRTs matched the ground truth trees and the regularization term of NNRTs avoids overfitting.

5. Performance Comparison on Real-World Datasets

In this section, we report the out-of-sample accuracy of NNRTs, MARS, Random Forests (RF), XGBoost and ORT-LH on 40 datasets obtained from UCI Machine Learning Repository ([12]). Our objective is to assess the

relative strength of NNRTs.

We partitioned each dataset into the training (50%), validation (25%), and testing sets (25%). The parameter we need to tune is the regularized parameter λ. We train the NNRT on the training set using different λ's and then we used the validation set R^2 to choose the tree based on which λ we want to use. After that, we kept the condition on each split unchanged and updated the coefficients f on each leaf using the combined training and validation sets.

We trained NNRTs of depths 1 to 12 on all datasets and compared them to solutions from MARS trimmed to the same depth. We use 50 random starts. To make a fair comparison, we also tuned the penalty parameter for MARS to receive the best solution. In order to minimize the influence of splitting the training, validation and test sets, we conducted the experiments five times with a different splitting each time. The final out-of-sample R^2 reported is the average of the five runs. The confidence interval is calculated by the mean and the standard deviations.

NNRTs were implemented in the `Julia` programming language ([17]). For testing the performance of MARS, we used the `earth` package in the `R` language ([18]). For random forests, we use the random forest package in `R` ([19]). For gradient-boosted trees, we use the XGBoost library ([20]) with $\eta = 0.1$. For ORT-LH, we used the OptimalTrees package in Julia programming language from ([4]) with auto tuning complexity parameter.

All problems were solved in parallel using MIT Engaging Cluster (Intel Xeon 2.1 GHz) with 4 CPU cores and 16GB memory.

We first compare all methods in all datasets and then present pairwise comparisons of NNRTs with MARS, Random Forest, XGBoost and ORT-LH.

Table 4 depicts the out of sample performance among all methods with maximum depth twelve. We determined the best depth for MARS, RF and XGBoost and best depth and complexity parameter for ORT-LH and NNRTs using the validation set. For each dataset, we rank the performance of all methods with one indicating best performance and five worst performance. Overall, NNRTs had average rank of 2.3, followed by XGBoost of 2.8, ORT-LH 3.0, RF 3.1, and MARS 3.8. From both average R^2 and average rank, we observe NNRTs have stronger performance than other methods. We notice XGBoost's rank is slightly better than ORT-LH because we include larger

Dataset			Mean out-of-sample R^2				
File	n	m	MARS	Random Forest	XGBoost	ORT-LH	NNRTs
abalone	4176	7	0.559 (0.542,0.576;2)	0.488 (0.519,0.553;5)	0.536 (0.528,0.544;4)	0.549 (0.541,0.557;3)	**0.564 (0.556,0.572;1)**
ailerons	7153	40	0.829 (0.821,0.837;4)	0.837 (0.83,0.844;2)	0.823 (0.815,0.831;5)	0.836 (0.832,0.84;3)	**0.846 (0.842,0.85;1)**
airfoil-self-noise	1502	5	0.802 (0.79,0.813;5)	0.91 (0.897,0.923;2)	**0.941 (0.935,0.947;1)**	0.896 (0.893,0.899;4)	0.903 (0.896,0.910;3)
automobile	391	8	0.849 (0.814,0.884;2)	0.847 (0.843,0.851;3)	**0.849 (0.79,0.908;1)**	0.838 (0.814,0.862;4)	0.804 (0.786,0.822;5)
autompg	158	51	0.844 (0.812,0.876;2)	0.732 (0.7,0.764;4)	0.736 (0.694,0.778;3)	0.631 (0.612,0.65;5)	**0.847 (0.828,0.866;1)**
Bike_sharing	17379	12	0.935 (0.931,0.939;4)	0.93 (0.928,0.933;5)	**0.951 (0.949,0.953;1)**	0.939 (0.936,0.942;3)	0.949 (0.947,0.951;2)
BlogFeedBack	52397	80	0.417 (0.416,0.418;4)	0.406 (0.405,0.407;5)	**0.563 (0.562,0.564;1)**	0.431 (0.431,0.431;3)	0.459 (0.459,0.459;2)
cart-artificial	40767	10	0.945 (0.945,0.945;5)	0.945 (0.945,0.945;4)	**0.948 (0.948,0.948;1)**	**0.948 (0.948,0.948;1)**	**0.948 (0.948,0.948;1)**
CBM	11934	16	0.93 (0.926,0.934;4)	**0.942 (0.937,0.946;1)**	0.934 (0.931,0.937;3)	0.927 (0.924,0.93;5)	0.935 (0.933,0.937;2)
Combined_cycle_power_plant	9568	4	0.949 (0.945,0.953;5)	0.956 (0.951,0.96;3)	0.966 (0.964,0.968;2)	0.955 (0.951,0.959;4)	**0.973 (0.971,0.975;1)**
communities-and-crime	122	122	0.301 (0.19,0.412;5)	0.702 (0.591,0.813;2)	0.679 (0.568,0.79;4)	**0.749 (0.693,0.805;1)**	0.684 (0.629,0.74;3)
computer-hardware	208	36	0.961 (0.953,0.969;3)	0.929 (0.916,0.942;5)	0.929 (0.927,0.931;4)	0.973 (0.969,0.977;2)	**0.986 (0.982,0.99;1)**
Concrete_data	1030	8	0.853 (0.847,0.859;5)	0.884 (0.879,0.889;4)	**0.929 (0.923,0.935;1)**	0.919 (0.916,0.922;3)	0.923 (0.92,0.926;2)
concrete-slump-test-compressive	102	7	0.947 (0.905,0.989;2)	0.69 (0.615,0.765;5)	0.792 (0.764,0.82;4)	0.873 (0.844,0.902;3)	**0.953 (0.936,0.97;1)**
concrete-slump-test-flow	102	7	0.364 (0.252,0.476;5)	0.437 (0.331,0.543;3)	0.396 (0.268,0.524;4)	0.476 (0.412,0.54;2)	**0.524 (0.469,0.579;1)**
concrete-slump-test-slump	102	7	0.265 (0.165,0.365;5)	0.359 (0.231,0.487;2)	0.272 (0.172,0.372;4)	0.277 (0.216,0.338;3)	**0.439 (0.401,0.477;1)**
cpu-act	8191	21	0.975 (0.974,0.976;5)	0.982 (0.982,0.982;2)	0.98 (0.976,0.984;4)	**0.984 (0.984,0.984;1)**	0.981 (0.98,0.982;3)
cpu-small	8191	12	0.966 (0.965,0.967;5)	**0.975 (0.975,0.976;2)**	0.975 (0.971,0.979;1)	0.974 (0.972,0.976;3)	0.971 (0.97,0.972;4)
elevators	8751	18	0.895 (0.886,0.904;3)	0.811 (0.802,0.82;5)	0.833 (0.824,0.842;4)	0.912 (0.908,0.916;2)	**0.918 (0.914,0.922;1)**
Facebook Comment Volume	50993	53	0.478 (0.477,0.479;2)	0.435 (0.434,0.436;4)	**0.528 (0.527,0.529;1)**	0.426 (0.426,0.426;5)	0.437 (0.436,0.438;3)
friedman-artificial	40767	10	0.902 (0.902,0.902;5)	0.932 (0.932,0.932;4)	0.952 (0.952,0.952;3)	**0.956 (0.956,0.956;1)**	**0.956 (0.956,0.956;1)**
geographic-origin	1059	68	0.609 (0.553,0.665;2)	0.608 (0.544,0.672;3)	**0.627 (0.572,0.682;1)**	0.53 (0.501,0.559;5)	0.562 (0.534,0.59;4)
housing	505	13	0.507 (0.476,0.538;5)	0.852 (0.82,0.884;2)	**0.863 (0.833,0.893;1)**	0.804 (0.789,0.82;3)	0.791 (0.775,0.807;4)
hybrid-price	152	3	0.62 (0.511,0.729;3)	0.615 (0.545,0.685;4)	0.577 (0.468,0.686;5)	**0.648 (0.608,0.688;1)**	0.642 (0.589,0.695;2)
kin8nm	8191	8	0.75 (0.74,0.76;3)	0.674 (0.665,0.683;4)	0.671 (0.659,0.683;5)	**0.851 (0.846,0.856;1)**	0.809 (0.804,0.814;2)
lpga-2008	156	6	**0.897 (0.865,0.929;1)**	0.77 (0.751,0.789;5)	0.775 (0.732,0.818;4)	0.851 (0.831,0.871;3)	0.863 (0.85,0.876;2)
lpga-2009	145	11	0.84 (0.826,0.854;5)	0.893 (0.88,0.906;2)	0.885 (0.865,0.905;4)	**0.904 (0.895,0.913;1)**	0.889 (0.881,0.897;3)
Obesity_levels	2111	16	0.842 (0.831,0.853;5)	0.867 (0.853,0.882;2)	**0.882 (0.874,0.89;1)**	0.85 (0.844,0.856;4)	0.863 (0.858,0.868;3)
parkinsons-telemonitoring-motor	5874	16	0.3 (0.29,0.31;2)	**0.341 (0.332,0.35;1)**	0.264 (0.251,0.277;4)	0.281 (0.275,0.287;3)	0.245 (0.24,0.25;5)
parkinsons-telemonitoring-total	5874	16	0.289 (0.271,0.307;3)	**0.355 (0.341,0.369;1)**	0.27 (0.251,0.289;5)	0.316 (0.307,0.325;2)	0.282 (0.276,0.288;4)
QSAR_aquatic_toxicity	546	8	0.401 (0.311,0.491;5)	0.539 (0.273,0.457;4)	0.541 (0.464,0.618;3)	0.569 (0.524,0.614;2)	**0.585 (0.543,0.627;1)**
Real_estate	414	6	0.597 (0.534,0.66;5)	0.626 (0.532,0.67;4)	0.638 (0.575,0.701;2)	0.636 (0.603,0.669;3)	**0.642 (0.609,0.675;1)**
Residential-Building-Data-Set-1	372	107	0.971 (0.965,0.977;4)	0.957 (0.95,0.964;5)	0.973 (0.968,0.978;3)	0.979 (0.976,0.982;2)	**0.985 (0.982,0.988;1)**
Residential-Building-Data-Set-2	372	107	0.973 (0.967,0.979;4)	0.943 (0.937,0.949;5)	0.981 (0.974,0.988;2)	0.978 (0.975,0.981;3)	**0.984 (0.981,0.987;1)**
Slice_localization_data	53500	385	0.896 (0.895,0.897;5)	0.983 (0.982,0.984;2)	0.966 (0.965,0.967;3)	0.932 (0.932,0.932;4)	**0.986 (0.986,0.986;1)**
TomsHardware	28178	96	0.953 (0.95,0.956;5)	0.998 (0.998,0.998;2)	**0.999 (0.999,0.999;1)**	0.957 (0.953,0.961;4)	0.973 (0.971,0.975;3)
vote-for-clinton	2703	9	0.255 (0.239,0.271;5)	**0.416 (0.401,0.431;1)**	0.389 (0.372,0.406;3)	0.395 (0.387,0.403;2)	0.353 (0.341,0.365;4)
wine-quality-red	1598	11	0.423 (0.4,0.446;2)	**0.435 (0.416,0.453;1)**	0.389 (0.365,0.413;3)	0.304 (0.293,0.315;5)	0.345 (0.335,0.355;4)
wine-quality-white	4897	11	0.3 (0.292,0.308;5)	**0.459 (0.451,0.467;1)**	0.422 (0.412,0.432;2)	0.349 (0.344,0.354;3)	0.318 (0.314,0.322;4)
yacht-hydrodynamics	307	6	0.994 (0.991,0.997;2)	0.994 (0.99,0.998;3)	**0.996 (0.994,0.998;1)**	0.991 (0.99,0.992;4)	**0.996 (0.995,0.997;1)**
Average			0.71 (0.706,0.714;3.8)	0.736 (0.732,0.74;3.1)	0.74 (0.736,0.744;3.0)	0.74 (0.736,0.744;2.8)	**0.753 (0.750,0.756;2.3)**

Table 4: Out of sample performance among all methods with maximum depth twelve. The number in parenthesis indicates the confidence interval and the rank of each method with one indicating best performance and five worst performance.

datasets in this paper compared to the datasets in [4]. Table 5 shows the number of wins of pairwise performance comparison of the methods in Table 4. We observe the number of wins of NNRTs is higher than all other methods.

Table 6 depicts the average out-of-sample R^2 across all 40 datasets for each method and tree depth. Maximum depth equals k means we train models from depth 1 to k separately and use validation set performance to decide the depth to use. NNRTs at depth three outperform all other methods at depth twelve. Furthermore, NNRTs average performance increased slightly (from average R^2 of 0.749 to 0.753) as we increased the maximum depth from four to twelve. We observed similar insensitivity to depth for XGBoost, ORT-LH and MARS. RF continued to improve with increasing depth. The fact that we only need depth four for NNRTs for most datasets is important

	MARS	RF	XGBoost	ORT-LH	NNRTs
MARS	-	15	12	11	6
RF	24-1	-	17	20	16
XGBoost	28	23	-	19	15
ORT-LH	29	20	21	-	12
NNRTs	34	24	23-2	28	-

Table 5: Number of wins of pairwise performance comparison of the methods in Table 4. 24-1 for example means that RF has stronger performance than MARS in 24/40 problems, 1/40 tie and MARS has stronger performance than RF in 15/40 problems.

Max.Depth	MARS	RandomForest	XGBoost	ORT-LH	NNRTs
1	0.641 (0.615,0.667)	0.532 (0.506,0.558)	0.662 (0.636,0.688)	0.688 (0.662,0.714)	0.665 (0.639,0.691)
2	0.68 (0.654,0.706)	0.591 (0.573,0.609)	0.692 (0.679,0.705)	0.714 (0.696,0.732)	0.703 (0.685,0.721)
3	0.698 (0.688,0.708)	0.65 (0.632,0.668)	0.732 (0.728,0.742)	0.725 (0.714,0.736)	0.74 (0.731,0.749)
4	0.704 (0.696,0.712)	0.671 (0.653,0.689)	0.734 (0.724,0.744)	0.735 (0.729,0.741)	0.749 (0.745,0.753)
5	0.71 (0.706,0.714)	0.686 (0.677,0.695)	0.738 (0.729,0.747)	0.738 (0.733,0.743)	0.753 (0.75,0.756)
6	0.71 (0.706,0.714)	0.698 (0.689,0.707)	0.739 (0.735,0.743)	0.74 (0.736,0.744)	0.753 (0.75,0.756)
7	0.71 (0.706,0.714)	0.71 (0.701,0.719)	0.739 (0.735,0.743)	0.74 (0.736,0.744)	0.753 (0.75,0.756)
8	0.71 (0.706,0.714)	0.721 (0.714,0.728)	0.74 (0.736,0.744)	0.74 (0.736,0.744)	0.753 (0.75,0.756)
9	0.71 (0.706,0.714)	0.727 (0.721,0.733)	0.74 (0.736,0.744)	0.74 (0.736,0.744)	0.753 (0.75,0.756)
10	0.71 (0.706,0.714)	0.732 (0.728,0.736)	0.74 (0.736,0.744)	0.74 (0.736,0.744)	0.753 (0.75,0.756)
11	0.71 (0.706,0.714)	0.735 (0.731,0.739)	0.74 (0.736,0.744)	0.74 (0.736,0.744)	0.753 (0.75,0.756)
12	0.71 (0.706,0.714)	0.736 (0.732,0.74)	0.74 (0.736,0.744)	0.74 (0.736,0.744)	0.753 (0.75,0.756)

Table 6: Mean out-of-sample R^2 for each method across all 40 datasets based on the maximum tree depth.

as the interpretability of trees is stronger with lower depth. From another perspective, the number of parameters a model used is important to the interpretability of a model. Because XGBoost and RF are ensemble models, the model makes predictions using a combination of hundreds of trees. Correspondingly, the parameters of XGBoost and RF are significantly more than a single-tree model like NNRTs, which is an advantage of NNRTs.

In Table 7, we calculate the number of datasets for which each method performed better, broken down by the maximum of the depth of the trees. At all depths, NNRTs are stronger than MARS in most datasets. The percentage of wins for NNRTs vs MARS ranges from 68%-83%. Furthermore, NNRTs had a higher R^2 of 0.03-0.05 ($5\% - 7\%$) at all depths.

Although NNRTs have a performance edge, it still has its own limitations. Comparing to CART or MARS which use univariate splits, NNRTs use multivariate splits which are harder to interpret. For larger datasets, NNRTs took much larger time to train than XGboost.

Max.depth	MARS wins	NNRTs wins	R^2 improvement
2	13	27	0.029(4.52%)
3	8	32	0.050(7.16%)
4	9	31	0.047(6.67%)
5	6	34	0.043(6.10%)

Table 7: Comparison of NNRTs and MARS from depth two to five showing the number of datasets for which each method had the highest out-of-sample R^2 and the mean improvement of R^2 when using NNRTs.

6. Conclusion

In this paper, we presented a novel formulation of the regression trees problem as a nonlinear optimization problem that motivates our new regression method, Near-optimal Nonlinear Regression Trees with Hyperplanes (NNRTs).

We illustrate how to use gradient descent method to solve the nonlinear optimization problem we propose. Experiments with synthetic data illustrate the algorithm recovers the ground truth and the regularization term avoids overfitting.

Results from 40 real-world datasets suggest that overall NNRTs have a performance edge over all other methods. The closest methods are XGBoost and ORT-LH. NNRTs have a considerable edge over MARS (increase out-of-sample R^2 of 6.1%).

References

[1] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[2] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, no. 7, pp. 1039–1082, 2017.

[3] D. Bertsimas and J. Dunn, *Machine learning under a modern optimization lens*. Dynamic Ideas, 2019.

[4] J. Dunn, *Optimal Trees for Prediction and Prescription*. PhD dissertation, Massachusetts Institute of Technology, 2018.

[5] J. R. Quinlan *et al.*, "Learning with continuous classes," in *5th Australian Joint Conference on Artificial Intelligence*, vol. 92, pp. 343–348, Singapore, 1992.

[6] P. Chaudhuri, W.-D. Lo, W.-Y. Loh, and C.-C. Yang, "Generalized regression trees," *Statistica Sinica*, pp. 641–666, 1995.

[7] T. Hothorn, K. Hornik, and A. Zeileis, "Unbiased recursive partitioning: A conditional inference framework," *Journal of Computational and Graphical Statistics*, vol. 15, no. 3, pp. 651–674, 2006.

[8] W.-Y. Loh, "Regression tress with unbiased variable selection and interaction detection," *Statistica Sinica*, pp. 361–386, 2002.

[9] H. Kim, W.-Y. Loh, Y.-S. Shih, and P. Chaudhuri, "Visualizable and interpretable regression models with good prediction power," *IIE Transactions*, vol. 39, no. 6, pp. 565–579, 2007.

[10] J. H. Friedman, "Multivariate adaptive regression splines," *The Annals of Statistics*, pp. 1–67, 1991.

[11] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[12] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[13] D. Bertsimas and M. S. Copenhaver, "Characterization of the equivalence of robustification and regularization in linear and matrix regression," *European Journal of Operational Research*, vol. 270, no. 3, pp. 931–942, 2018.

[14] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Advances in Neural Information Processing Systems*, pp. 161–168, 2008.

[15] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2015.

[17] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[18] M. S. Milborrow, "Package 'earth'," *R Software package*, 2019.

[19] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[20] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data mining*, pp. 785–794, ACM, 2016.

[21] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. Jan, pp. 1–30, 2006.

[22] S. Garcia and F. Herrera, "An extension on"statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons," *Journal of Machine Learning Research*, vol. 9, no. Dec, pp. 2677–2694, 2008.

[23] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, pp. 65–70, 1979.