Pattern Recognition 86 (2019) 1–13

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Nonlinear regression via incremental decision trees

N. Denizcan Vanli[a], Muhammed O. Sayin[b], Mohammadreza Mohaghegh N.[c,*], Huseyin Ozkan[d], Suleyman S. Kozat[c]

[a] Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[b] Coordinated Science Laboratory, University of Illinois at Urbana-Champaign (UIUC), Urbana, IL 61801, USA
[c] Department of Electrical and Electronics Engineering, Bilkent University, Ankara 06800, Turkey
[d] Faculty of Engineering and Natural Sciences, Sabancı University, Istanbul 34956, Turkey

ARTICLE INFO

Article history:
Received 4 December 2017
Revised 7 July 2018
Accepted 27 August 2018
Available online 28 August 2018

Keywords:
Online regression
Sequential learning
Nonlinear models
Incremental decision trees

ABSTRACT

We study sequential nonlinear regression and introduce an online algorithm that elegantly mitigates, via an adaptively incremental hierarchical structure, convergence and undertraining issues of conventional nonlinear regression methods. Particularly, we present a piecewise linear (or nonlinear) regression algorithm that partitions the regressor space and learns a linear model at each region to combine. Unlike the conventional approaches, our algorithm effectively learns the optimal regressor space partition with the desired complexity in a completely sequential and data driven manner. Our algorithm sequentially and asymptotically achieves the performance of the optimal twice differentiable regression function for any data sequence without any statistical assumptions. The introduced algorithm can be efficiently implemented with a computational complexity that is only logarithmic in the length of data. In our experiments, we demonstrate significant gains for the well-known benchmark real data sets when compared to the state-of-the-art techniques.

1. Introduction

Nonlinear regression has been extensively studied in machine learning and signal processing literature due to its applicability in an extremely wide range of real life scenarios ranging from prediction in time series [1,2] to face recognition [3,4] and object tracking [5]. Numerous methods have been proposed based on various approaches such as neural networks [6–8], Volterra filters [9], and B-splines [10]. However, we observe that existing methods suffer from convergence and scalability issues in addition to their limited generalization across applications and data domains [11–14]. To address these issues, we study online nonlinear regression based on a sequence of multi-dimensional regressors and a corresponding sequence of desired outputs. At each time in our framework, after we produce our estimate, the desired output is revealed and then we aim to improve our model based on the induced error. Our goal is to sequentially learn the nonlinear and possibly time varying model that minimizes the accumulated square error across the observations in the target class of all twice differentiable regression functions.

We consider hierarchical models of gradually increasing complexity to develop a novel regression model that infers the required complexity of any regression problem regardless of the data and its domain. Our approach is to recursively partition the regressor space into subsequent regions and fit a linear model in each partition region. Then, we obtain a piecewise-linear (thus, nonlinear) regression model by combining such local linear models of finite complexity. During partitioning, we specifically avoid creating undertrained regions until a sufficient amount of data is observed. In this sense, the nonlinear modeling power is incrementally increased to the required degree by sequentially inferring the right (in terms of the granularity and shape) partitioning structure directly from the data. As a result, we avoid any unnecessary complexity or nonlinearity (while staying capable of achieving arbitrarily high modeling power if required) to mitigate overfitting issues by operating at the true complexity.

We prove that our hierarchical piecewise linear regression algorithm asymptotically achieves the performance of the optimal twice differentiable regression function. We obtain this strong performance guarantee in a truly sequential manner without any statistical assumptions and parameter tuning. Hence, our algorithm is universally well-behaved in terms of its convergence and consis-

* Corresponding author.
E-mail addresses: denizcan@mit.edu (N.D. Vanli), sayin2@illinois.edu (M.O. Sayin), mohammadreza@ee.bilkent.edu.tr (M. Mohaghegh N.), hozkan@sabanciuniv.edu (H. Ozkan), kozat@ee.bilkent.edu.tr (S.S. Kozat).

https://doi.org/10.1016/j.patcog.2018.08.014
0031-3203/

tency. Since most of the existing nonlinear regression algorithms such as neural networks and Volterra filters can be accurately represented by twice differentiable functions [15,16], our algorithm readily -and at least- performs asymptotically as well as those. Additionally, the computational complexity of our technique is only logarithmic in the length of data under mild regularity conditions.

The overfitting handling capability and thus the generalization of piecewise-linear models can be illustrated in the case of classification. Linear classifiers are desirable due to their good generalization since they have finite $d+1$ (d is the data dimension) VC dimension [17], but they might not necessarily address the true complexity of the problem in hand which might be severely nonlinear. Kernelization through radial basis function -as an example- achieves great nonlinear modeling power, but it has infinite VC dimension leaving it susceptible to overfitting. On the other hand, the VC dimension of piecewise linear classifiers (with r regions) is still finite, bounded by $2\frac{(r-1)^2+2}{2}\log(e\frac{(r-1)^2+2}{2})(d+1) < \infty$ [18], which provides a decent model to fight against overfitting at the desired (possibly arbitrarily large) complexity. In accordance, our idea is to use a combination of finite complexity linear regression models to solve more complex regression problems. By gradually increasing the number of combined models, one can match the true complexity of the problem with a decent piecewise linear approximation. Consequently, our technique directly addresses and mitigates overfitting at the right operating point in the trade-off between the generalization and modeling power. Additionally, we learn the optimal partitioning in a completely data driven manner while specifically avoiding undertrained regions in the phase of region splitting for enhanced generalization. We also exploit a carefully designed weighting to favor simpler models initially, and then to dynamically and gradually switch to more complex models as the data overwhelm. This not only addresses the cold-start problem as an additional merit, but also manages the piecewise linear models with special regards to overfitting. Thus, our hierarchical class of hypotheses is grown in a completely data driven manner until the desired level of complexity, and then we provide the optimal regression model from that class with strong mathematical performance guarantees.

For reducing the sensitivity of our approach to noise, we use regularized least squares as the linear model that our combination is based on. One can use any linear model that might be less sensitive to noise and outliers for this purpose. Our emphasis in this paper is on the combination, rather than the individual linear models. Secondly, one should not create new regions based on noise; and in our method, a new region is never created until a sufficiently number of instances is observed in its parent region. This level of sufficiency can be studied more deeply to reduce the effect of noise. In this paper, we opt to provide it as a framework and continue with an appropriate setting that leads to an impressive performance in our experiments.

2. Related work

Piecewise linear regression using tree structures has been studied in the computational learning [14,19–21] and signal processing [13,22] literature due to its attractive convergence and consistency features. Remarkably, the tree based partitioning methods in [14,16,22] consider a large class of hierarchical models and achieve the performance of the optimal one defined by the best pruning of the tree. However, these methods only yield a satisfactory performance when the right partitioning of the regressor space is already known initially beforehand, which cannot be satisfied in practice. In another example, Vanli and Kozat [11] proposes an algorithm that achieves the performance of the optimal combination of such piecewise models, rather than the optimal single one. However, it considers a partitioning tree with a pre-fixed depth and its computational complexity is exponentially greater compared to the ones in [14,16,22]. All these algorithms can only provide a limited modeling power since their tree structure is fixed. Furthermore, they can only learn the locally optimal region boundaries due to their highly nonconvex optimization. Unlike these methods, our technique incrementally increases its nonlinear modeling power according to the observed data and directly achieves the performance of the best twice differentiable regression function that globally minimizes the accumulated regression error. In contrast to the relevant studies of the literature in which the undertrained (i.e., unnecessary) partitions are kept in the overall structure, our method eliminates the unnecessarily finer partitions without any loss in asymptotical performance.

The Classification and Regression Trees (CART) [23] recursively partitions the observation space based on the data attributes using a certain splitting criterion at each node, such as the squared error for regression or Gini index for classification, and runs a predictor at leaf nodes. Pruning can also be incorporated once the tree is learned [24]. Utgoff's perceptron tree [25] is a decision tree of a hybrid representation consisting of decision nodes with attribute tests and leaf nodes with perceptrons. Splitting is information theoretical and continues until linear separability. Extensions largely investigate various univariate/multivariate splitting criteria, stopping criteria and pruning methods [24]. Model trees combine a conventional decision tree with linear regression functions at the leaves [26]. M5 (and M5') of Quinlan [26] is a model tree in which a splitting criterion is used to minimize the intra-subset variation down each branch. Cubist [27] is a rule-based model that is an extension of Quinlan's M5 model tree. Online regression trees are discussed in [27]. Especially, a recent model tree called Fast Incremental Model Tree (FIMT) is studied and compared to the previous incremental trees. The FIMT, FIRT-DD and FIMT-DD algorithms by Ikonomovska et al. [28–30] are representatives of Hoeffding-based learning algorithms in the domain of regression analysis. The FIMT-DD algorithm uses a probabilistic sampling strategy for learning in non-stationary environments [27]. FIMT is an online algorithm to learn linear model trees from stationary streams. The FIRT-DD is an extended version of FIMT equipped with change detection abilities to learn from time-varying data streams. FIRT-DD does not use linear models in leaves but FIMT-DD has both linear models in leaves and change detection. In these tree based algorithms, the major effort to optimize the model is given to the optimization of the splitting criterion at each node [24]. On the contrary, we opt to mainly consider the model optimization directly in the class of twice differentiable regression functions while using a straightforward splitting criterion at each node. We emphasize that our approach of this direct optimization covers the solutions or their approximations resulting from splitting criteria optimization. Moreover, any splitting criterion can also be straightforwardly incorporated into our framework.

In nonlinear techniques such as B-splines and Volterra series [9,10], the nonlinearity is introduced by the basis functions to create polynomial estimators. The performance of this approach is satisfactory when the data generation is in accordance with the employed basis function. However, the underlying model that generates the data is usually unknown in the real life applications. On the other hand, our algorithm achieves the performance of any such regressor provided that its basis functions are twice differentiable. In this sense, unlike the conventional methods whose performances are highly dependent on the basis functions, our method can well approximate these basis functions via piecewise models and therefore effectively addresses the well-known basis/kernel selection problem. Namely, the difference between the performance of our algorithm and the best such regressor vanishes asymptotically in a strong individual sequence manner without any statistical assumptions.

We first provide the problem description in Section 3 and then introduce our incremental decision tree in Section 4. We present our performance guarantees in Section 5 which are explained in detail in Section 6. Section 7 presents the experimental results and then we conclude in Section 8.

3. Problem description

We study sequential nonlinear regression to estimate an unknown desired sequence $\{d[t]\}_{t\geq 1}$ by using a sequence of regressor vectors $\{\boldsymbol{x}[t]\}_{t\geq 1}$, where the desired sequence and the regressor vectors are real valued and bounded, i.e., $d[t]\in\mathbb{R}$, $\boldsymbol{x}[t]\triangleq[x_1[t],\ldots,x_p[t]]^T\in\mathbb{R}^p$ for an arbitrary integer p and $|d[t]|\leq A<\infty$, $|x_i[t]|\leq A<\infty$ for all t and $i=1,\ldots,p$.

We point out that in this work, the regressors and the responses are both assumed to be coming from a compact space of known bounds in its dimensions, i.e., $|x_i[t]|\leq A$, $|d[t]|\leq A$ and A is known. We consider that this does not hinder online/sequential processing as it can be readily known (as in the case of images or digitized/quantized signals) or conservatively and accurately estimated (by observing a small portion at the beginning of the data stream) in most of the practical cases. We call the regressors "sequential" if they only use the past information $d[1],\ldots,d[t-1]$ and the observed regressor vectors[1] $\boldsymbol{x}[1],\ldots,\boldsymbol{x}[t]$ in order to estimate the desired data at time t, i.e., $d[t]$.

In this framework, a piecewise linear model is constructed by dividing the regressor space into disjoint regions with a linear model in each region. As an example, suppose that the regressor space is parsed into K disjoint regions $\mathcal{R}_1,\ldots,\mathcal{R}_K$ such that $\bigcup_{k=1}^{K}\mathcal{R}_k=[-A,A]^p$. Given such a model, at each time t, the sequential linear[2] regressor predicts $d[t]$ as $\hat{d}[t]=\boldsymbol{v}_k^T[t]\boldsymbol{x}[t]$ when $\boldsymbol{x}[t]\in\mathcal{R}_k$, where $\boldsymbol{v}_k[t]\in\mathbb{R}^p$ for all $k=1,\ldots,K$. These linear models assigned to each region can be trained independently using different adaptive methods such as the gradient descent or the recursive least squares (RLS) algorithms.

However, by directly partitioning the regressor space in advance before the processing starts and optimizing only the internal parameters of the piecewise linear model, i.e., $\boldsymbol{v}_k[t]$, one significantly limits the performance of the overall regressor since we do not have any prior knowledge on the underlying desired signal. Therefore, instead of committing to a single piecewise linear model with a fixed and given partition, one can use a decision tree to partition the regressor space and aim to achieve the performance of the best partition over the whole doubly exponential number of different models represented by this tree [31].

As an example, we partition the one dimensional regressor space $[-A,A]$ using a depth-2 tree in Fig. 1a, where the regions $\mathcal{R}_1,\ldots,\mathcal{R}_4$ correspond to disjoint intervals on the real line and the internal nodes are constructed using union of these regions. In the generic case of a depth-d full decision tree, there exist 2^d leaf nodes and 2^d-1 internal nodes. Each node of the tree represents a portion of the regressor space such that the union of the regions represented by the leaf nodes is equal to the entire regressor space $[-A,A]^p$. Moreover, the region corresponding to each internal node is constructed by the union of the regions of its children. In this way, we obtain $2^{d+1}-1$ different nodes (regions) on the depth-d decision tree (on the regressor space) and approximately 1.5^{2^d} different piecewise models that can be represented by certain collections of the regions at the nodes of the decision tree [31]. For example, there are 7 different nodes on the depth-2 tree in Fig. 1a; and as shown in Fig. 1b, a depth-2 tree defines 5 different piecewise partitions or models, where each of these models is constructed using certain unions of the nodes of the full depth decision tree.

We emphasize that given a decision tree of depth-d, the nonlinear modeling power of this tree is fixed and finite since there are only $2^{d+1}-1$ different regions (one for each node) and approximately 1.5^{2^d} different piecewise models (i.e. partitions) defined on this tree. To avoid such a limitation, we recursively increment the depth of the decision tree as the length of data increases. We call such a tree the "incremental decision tree" since the depth of the decision tree is incremented (and potentially goes to infinity) as the data length n increases. Hence, we can achieve the modeling power of an infinite depth tree.

Using this incremental structure, we construct our sequential regression algorithm whose estimate at time t is $\hat{d}_s[t]$. When applied to any sequence of data and regressor vectors, our algorithm yields the regret performance

$$\sum_{t=1}^{n}\left(d[t]-\hat{d}_s[t]\right)^2-\inf_{f\in\mathcal{F}}\sum_{t=1}^{n}\left(d[t]-\hat{d}_f[t]\right)^2\leq o(n) \tag{1}$$

over any n without the knowledge of n, where $\mathcal{F}$ represents the class of all twice differentiable functions whose parameters are set in hindsight, i.e., after observing the entire data before processing starts, and $\hat{d}_f[t]$ represents the estimate of the twice differentiable function $f\in\mathcal{F}$ at time t. The relative accumulated error in (1) represents the performance difference of the introduced algorithm and the optimal batch twice differentiable regressor. Hence, an upper bound of $o(n)$ in (1) implies that the algorithm $\hat{d}_s[t]$ sequentially and asymptotically converges to the performance of the regressor $\hat{d}_f[t]$ for any $f\in\mathcal{F}$.

4. Nonlinear regression via incremental decision trees

In this section, we present our incremental decision tree structure and use it for piecewise linear regression. For clarity, we first introduce the notation to effectively describe our incremental decision tree structure. We next introduce an iterative regressor space partitioning rule and construct an incremental decision tree using the resulting partitions. We then assign separate linear regressors to each node on this incremental decision tree and then introduce a sequential algorithm that achieves the performance of the best piecewise model on this incremental decision tree in Section 6.

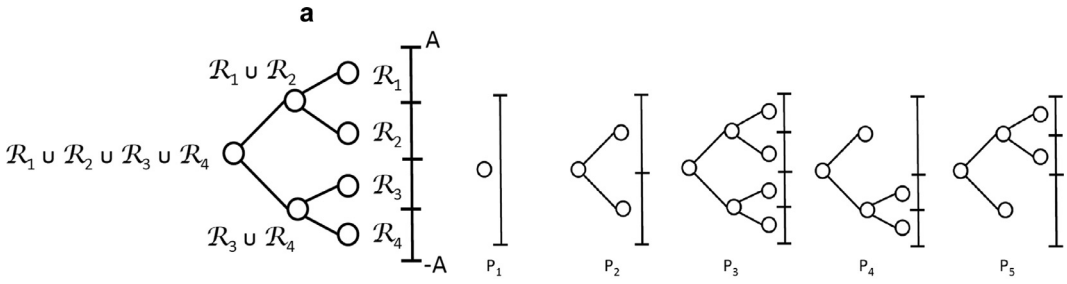
4.1. Notation

We introduce a labeling for the nodes of the tree as in [32]. The root node is labeled with an empty binary string λ; and assuming that a node has a label κ, where $\kappa=\nu_1\ldots\nu_l$ is a binary string of length l formed from letters $\nu_1,\ldots,\nu_l$, we label its upper and lower children as $\kappa 1$ and $\kappa 0$, respectively. Here, we emphasize that a string can only take its letters from the binary alphabet, i.e., $\nu\in\{0,1\}$, where 0 refers to the lower child, and 1 refers to the upper child of a node. According to this notation, we say that a string $\kappa'=\nu'_1\ldots\nu'_{l'}$ is a prefix to string $\kappa=\nu_1\ldots\nu_l$ if $l'\leq l$ and $\nu'_i=\nu_i$ for all $i=1,\ldots,l'$, where the empty string λ is a prefix to all strings. We let $l(\kappa)$ represent the length of the string κ and $\mathcal{J}(\kappa)$ represent the set of all prefixes to the string κ, i.e., $\mathcal{J}(\kappa)\triangleq\{\kappa_0,\ldots,\kappa_l\}$, where $l(\kappa)=l$ is the length of the string κ, κ_i is the string with length $l(\kappa_i)=i$, and $\kappa_0=\lambda$ is the empty string such that the first i letters of the string κ forms the string κ_i for all $i=0,\ldots,l$.

We let $\mathcal{L}_t$ and $\mathcal{N}_t$ represent the set of all leaf nodes and the set of all nodes on the incremental decision tree at time t, respectively. For each leaf node on the incremental decision tree at each time t, i.e., $\forall\kappa\in\mathcal{L}_t$, we assign a specific index $\alpha_\kappa\in\{0,\ldots,M-1\}$

[1] All vectors are column vectors and denoted by boldface lower case letters. Matrices are denoted by boldface upper case letters. For a vector $\boldsymbol{x}$, $\boldsymbol{x}^T$ is the ordinary transpose. We denote $d_a^b\triangleq\{d[t]\}_{t=a}^b$. Also, the $p\times p$ identity matrix is shown as $\boldsymbol{I}_p$.

[2] Note that affine models can also be represented as linear models by appending a 1 to $\boldsymbol{x}[t]$, where the dimension of the regressor space increases by one.



(a) The partitioning of a one dimensional regressor space, i.e., [−A;A], using a depth-2 full decision tree, where each node represents a portion of the regressor space.

(b) All piecewise linear models that can be obtained using a depth-2 decision tree with one dimensional regressor space. These models are based on the partitioning shown in Fig. 1a.

Fig. 1. The partitioning of the regressor space by using a decision tree.

representing the number of regressor vectors that has fallen into $\mathcal{R}_\kappa$. The parameter M controls the rate of the growth regarding our tree as well as the set $\mathcal{M}_n$ of all hierarchical prediction models defined on our incremental decision tree at time n. The depth of the tree increases as M decreases, in which case each node of the tree is trained -but- using less instances. Hence, decreasing M increases the variance of the piecewise models but also increases the modeling power of our method. However, the resulting rate of tree growth due to $M = 2$ along with the weighting over the set $\mathcal{M}_n$ elegantly achieves the quickest possible rate of inclusion of new powerful models into $\mathcal{M}_n$ and this is in line with the learning rate from data becoming available (cf. our regret analysis). We use $M = 2$ throughout the paper.

4.2. Incremental decision trees

Before the processing starts, i.e., at time $t = 0$, we begin with a single node, i.e., the root node λ, having index $\alpha_\lambda = 0$. Then, we recursively construct the decision tree according to the following principle. For every time instant $t > 0$, we find the leaf node of the tree $\kappa \in \mathcal{L}_t$ such that $\boldsymbol{x}[t] \in \mathcal{R}_\kappa$. For this node, if we have $\alpha_\kappa = 0$, we do not modify the tree but only increment this index by 1. On the other hand, if $\alpha_\kappa = 1$, then we generate two children nodes $\kappa 0, \kappa 1$ for this node by dividing the region $\mathcal{R}_\kappa$ into two disjoint regions $\mathcal{R}_{\kappa 0}, \mathcal{R}_{\kappa 1}$, using the plane $x_i = c$, where $i - 1 \equiv l(\kappa) \pmod p$ and c is the midpoint of the region $\mathcal{R}_\kappa$ along the ith dimension. For node $\kappa\nu$ with $\boldsymbol{x}[t] \in \mathcal{R}_{\kappa\nu}$ (i.e., the child node containing the current regressor vector), we set $\alpha_{\kappa\nu} = 1$ and the index of the other child is set to 0. We emphasize that this simple splitting criterion yields our desired performance, as shown in the proof of Theorem 2. Using this splitting, each dimension of the regions corresponding to the nodes with the same depth on the tree has the same radius, which can be calculated and used to prove the desired performance bounds. The accumulated regressor vectors $\mathcal{T}(\kappa)$ for region of node κ (i.e. $\mathcal{T}(\kappa) = \{t_i : \boldsymbol{x}[t_i] \in \mathcal{R}(\kappa)\}$) and the data in node κ are transferred to its children to train a linear regressor in these child nodes.

As an example, we consider one dimensional regressor space $[-A, A]$ and present a sample evolution of the tree in Fig. 2a. At time $t = 2$, we have a depth-1 tree of two nodes 0 and 1 with corresponding regions $\mathcal{R}_0 = [-A, 0]$, $\mathcal{R}_1 = [0, A]$, and $\alpha_0 = 1$, $\alpha_1 = 0$. At time $t = 3$, we observe a regressor vector $\boldsymbol{x}[3] \in \mathcal{R}_0$ and divide this region into two disjoint regions using $x_1 = -A/2$ line. We then find that $\boldsymbol{x}[3] \in \mathcal{R}_{01}$ and set $\alpha_{01} = 1$, whereas $\alpha_{00} = 0$.

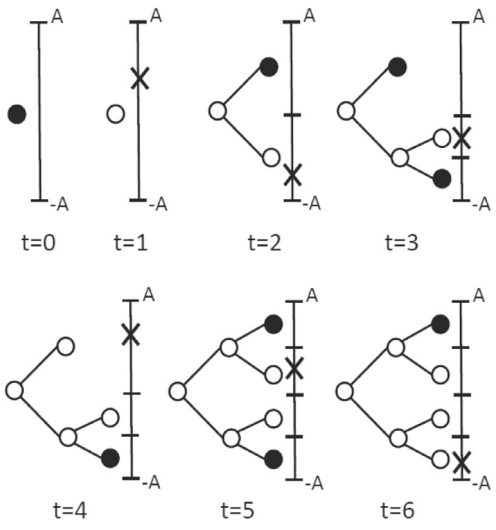
As another example, we depict a tree of depth 3 for 2-dimensional regressor vectors over $[-A, A]^2$ in Fig. 2b. In order to split the root node in this example, we use $x_1 = 0$ as the separating hyperplane, since the length of the code describing the root node (i.e., the depth of the node in the tree) equals 0 that yields $i = 1$ as the index of the splitting dimension. Similarly, we use $x_2 = 0$ as the separating hyperplane for the nodes with depth 1, since we obtain $i = 2$ for these nodes and $x_2 \in [-A, A]$ for both of these nodes, i.e., $c = 0$ is the midpoint along the second dimension in both of these nodes. For the depth 3 nodes, we obtain $i = [2 \mod 2] + 1 = 1$, therefore, we do the splitting along x_1. For example, in Fig. 2b, for the highest node with depth 2, i.e., $\kappa = 11$ (with the coding scheme stated in the paper), we have $x_1 \in [0, A]$ and $c = A/2$ is the midpoint along x_1. Thus, we use $x_1 = A/2$ as the separating hyperplane to generate the nodes with codes 111 and 110 from the node 11.

We assign an independent linear regressor to each node on the incremental decision tree. Each linear regressor is trained using only the information contained in its corresponding node. Hence, we can obtain different piecewise models by using a certain collection of this node regressors according to the hierarchical structure. Using this incremental hierarchical structure with linear regressors at each region, the incremental decision tree can represent up to 1.5^n different piecewise linear models after observing a data of length n. For example, at time $t = 6$ in Fig. 2a, we have 5 different piecewise linear models (see Fig. 1b), whereas at time $t = 4$, we have 3 different piecewise linear models. Each of these piecewise linear models can be used to perform the estimation task. We introduce the following universal piecewise linear regressor for the piecewise model m. Assuming that $\boldsymbol{x}[t] \in \mathcal{R}_\kappa$, we let

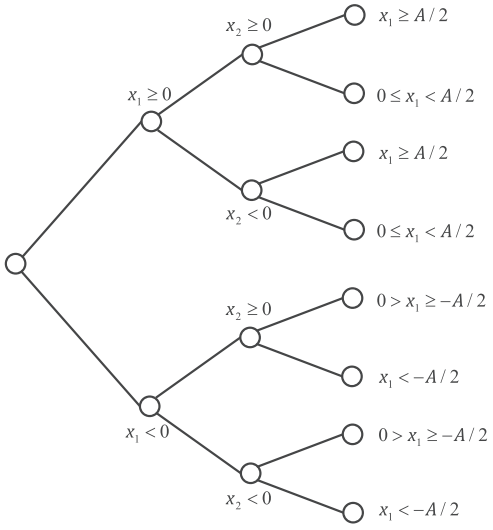
$$\hat{d}^{(m)}[t] = \boldsymbol{v}_\kappa^T[t]\boldsymbol{x}[t], \tag{2}$$

where $\boldsymbol{v}_\kappa[t] = (\boldsymbol{R}_\kappa[t] + \delta\boldsymbol{I})^{-1}\boldsymbol{p}_\kappa[t]$ with $\boldsymbol{I}$ representing the appropriate sized identity matrix, $\boldsymbol{R}_\kappa[t] \triangleq \sum_{t' \le t\,:\,\boldsymbol{x}[t'] \in \mathcal{R}_\kappa} \boldsymbol{x}[t']\boldsymbol{x}^T[t']$, and $\boldsymbol{p}_\kappa[t] \triangleq \sum_{t' < t\,:\,\boldsymbol{x}[t'] \in \mathcal{R}_\kappa} d[t']\boldsymbol{x}[t']$. In addition, δ is a regularization parameter used to avoid taking inverse of a singular matrix, hence it is usually set to be very small. Therefore, we initialize the matrix $\boldsymbol{R}_\kappa$ for every node (as soon as a node is added to the tree) by $\boldsymbol{R}_\kappa[0] = \delta\boldsymbol{I}$, update it by $\boldsymbol{R}_\kappa[t] = \boldsymbol{R}_\kappa[t-1] + \boldsymbol{x}[t]\boldsymbol{x}^T[t]$, and reformulate $\boldsymbol{v}_\kappa[t]$ as $\boldsymbol{v}_\kappa[t] = \boldsymbol{R}_\kappa^{-1}[t]\boldsymbol{p}_\kappa[t]$. For instance, one can set $\delta = 0.01$ in practice.

However, we use a mixture of experts approach to combine the outputs of all piecewise linear models instead of relying on a single one. To this end, one can assign a performance dependent weight to each piecewise linear model defined on the incremental decision tree and combine their weighted outputs to obtain the final estimate [33]. In a conventional setting, such a mixture of expert approach is guaranteed to asymptotically achieve the performance of the best piecewise linear model defined on the tree [34]. However, in our incremental decision tree framework, as t increases (i.e., as we observe new data), the total number of different piecewise linear models can increase exponentially with t.



(a) A sample evolution of the incremental decision tree with 1-D regressor space. The "×" indicates the regressor at that specific time. Light (dark) nodes are of index of 1 (0).



(b) The depth-3 tree constructed for partitioning two dimensional regressors (p = 2).

Fig. 2. Two partitioning examples in 1-D and 2-D scenarios.

Thus, we have a highly dynamic optimization framework. For example, at time $t = 4$ in Fig. 2a, we have 3 different piecewise linear models, hence calculate the final output of our algorithm as $\hat{d}[t] = w_1[t]\hat{d}^{(1)}[t] + w_2[t]\hat{d}^{(2)}[t] + w_3[t]\hat{d}^{(3)}[t]$, where $\hat{d}^{(i)}[t]$ represents the output of the ith piecewise linear model and $w_i[t]$ represents its weight. However, at time $t = 6$, we have 5 different piecewise linear models, i.e., $\hat{d}[t] = \sum_{i=1}^{5} w_i[t]\hat{d}^{(i)}[t]$, therefore the number of experts increases. Hence, not only such a combination approach requires the processing of the entire observed data at each time t (i.e., it results in a brute-force batch-to-online conversion), but also it cannot be practically implemented even for a considerably short data sequences such as $n = 100$.

To elegantly solve this problem, we assign a weight to each node on the incremental decision tree instead of using a conventional mixture of experts approach. In this way, we illustrate a method to calculate the original highly dynamic combination weights in an efficient manner without requiring the processing of the entire data for each new sample and with a significantly reduced computational complexity. The main structure of the proposed algorithm is provided in Algorithm 1. In this algorithm,

Algorithm 1: Incremental Decision Tree (IDT).

1: Find the leaf node containing $\vec{x}[t]$, denote it by κ.
2: **if** $\alpha_\kappa = 1$ **then**
3: incrementTree(κ) using the Algorithm 3
4: Find the new leaf node containing $\vec{x}[t]$ on the incremented tree, denote it by κ.
5: **end if**
6: $\alpha_\kappa = 1$.
7: $\mathcal{T}_{\kappa_i} = \mathcal{T}_{\kappa_i} \cup \{t\}$, $\forall \kappa_i \in \mathcal{J}(\kappa)$.
8: predict($\vec{x}[t], \mathcal{J}(\kappa)$) using the Algorithm 2
9: update($d[t], \vec{x}[t], \mathcal{J}(\kappa)$) using the Algorithm 4

when a regressor vector $\boldsymbol{x}[t]$ is received at time t, we find the leaf node κ containing this sample. Clearly, due to the structure of the tree, all the ancestors of the κ also contain this sample. Hence, in line 8 of the Algorithm 1, we use the estimations of all nodes in $\mathcal{J}(\kappa)$ to produce the final output $\hat{d}[t]$ (as will be discussed in Section 6 and Algorithm 2). Furthermore, using the function

Algorithm 2: predict($\boldsymbol{x}[t], \mathcal{J}(\kappa)$).

1: **for all** $\kappa_i \in \mathcal{J}(\kappa)$ **do**
2: Use (16) to find π_{κ_i}.
3: $\mu_{\kappa_i} = \pi_{\kappa_i} E_{\kappa_i} / P_\lambda$
4: $\hat{d}_{\kappa_i} = \vec{w}_{\kappa_i}^T \vec{x}[t]$
5: **end for**
6: $\hat{d} = \sum_{\kappa_i \in \mathcal{J}(\kappa)} \mu_{\kappa_i} \hat{d}_{\kappa_i}$

"incrementTree(κ)" (which will be explained later in Algorithm 3),

Algorithm 3: incrementTree(κ).

1: Fix the regularization parameter δ at a very small positive constant
2: Initialize $\vec{R}_{\kappa 0} = \delta \vec{I}_p$, $\vec{R}_{\kappa 1} = \delta \vec{I}_p$ and $E_{\kappa\nu} = 1$.
3: **for all** $z \in \mathcal{T}_\kappa$ **do**
4: **if** $\vec{x}[z] \in \mathcal{R}_{\kappa 0}$ **then**
5: $\nu = 0$
6: **else**
7: $\nu = 1$
8: **end if**
9: $\mathcal{T}_{\kappa\nu} = \mathcal{T}_{\kappa\nu} \cup \{z\}$
10: $E_{\kappa\nu} = E_{\kappa\nu} \exp(-(d[z] - \vec{w}_{\kappa\nu}^T \vec{x}[z])^2 / 2a)$
11: $P_{\kappa\nu} = E_{\kappa\nu}$
12: $\vec{R}_{\kappa\nu} = \vec{R}_{\kappa\nu} + \vec{x}[z]\vec{x}^T[z]$
13: $\vec{w}_{\kappa\nu} = \vec{w}_{\kappa\nu} + \vec{R}_{\kappa\nu}^{-1}(\vec{x}[z](d[z] - \vec{w}_{\kappa\nu}^T \vec{x}[z]))$
14: **end for**
15: **for all** $\kappa_i \in \mathcal{J}(\kappa)$ **do**
16: $P_{\kappa_i} = (P_{\kappa_i 0} P_{\kappa_i 1} + E_{\kappa_i})/2$
17: **end for**

we pass the accumulated information (e.g., the linear estimator) in

the node κ to its children, when this node receives enough amount of data to be split. Note that $\mathcal{T}(\kappa_i)$ indicates the set of all time indexes t_i such that $\boldsymbol{x}[t_i] \in \mathcal{R}(\kappa_i)$. In addition, we also update the linear regressors of all nodes containing $\boldsymbol{x}[t]$ (i.e., all nodes in $\mathcal{J}(\kappa)$) using the Algorithm 4 , which will be discussed later. Before describing our algorithm in detail, we first provide the theoretical guarantees of our algorithm in the following section.

Algorithm 4: update$(d[t], \boldsymbol{x}[t], \mathcal{J}(\kappa))$.

for all $\kappa_i \in \mathcal{J}(\kappa)$ **do**
  $E_{\kappa_i} = E_{\kappa_i} \exp(-(d[t] - \hat{d}_{\kappa_i})^2/(2a))$
  $P_{\kappa_i} = \begin{cases} E_{\kappa_i} & \text{, if } \kappa_i = \kappa \\ (P_{\kappa_i 0} P_{\kappa_i 1} + E_{\kappa_i})/2 & \text{, o.w.} \end{cases}$
  $\vec{R}_{\kappa_i} = \vec{R}_{\kappa_i} + \vec{x}[t]\vec{x}^T[t]$
  $\vec{w}_{\kappa_i} = \vec{w}_{\kappa_i} + \vec{R}_{\kappa_i}^{-1}(\vec{x}[t](d[t] - \hat{d}_{\kappa_i}))$
end for

5. Main results

We introduce the main results in this section. Particularly, we first show that the introduced sequential piecewise linear regression algorithm asymptotically achieves the performance of the best piecewise linear model defined on the incremental decision tree (with possibly infinite depth) with the optimal regression parameters at each region that minimizes the accumulated loss. We then use this result to prove that the introduced algorithm asymptotically achieves the performance of any twice differentiable regression function. We provide the algorithmic details and the construction of the algorithm in Section 6.

Theorem 1. *Let $\{d[t]\}_{t\geq 1}$ and $\{\boldsymbol{x}[t]\}_{t\geq 1}$ be arbitrary, bounded, and real-valued sequences of data and regressor vectors, respectively, i.e., $\boldsymbol{x}[t] \in [-A, A]^p, \forall t$. Then, Algorithm 1, whose prediction at time t is $\hat{d}[t]$, yields*

$$\sum_{t=1}^{n} \left(d[t] - \hat{d}[t]\right)^2 - \inf_{m\in\mathcal{M}_n} \left[\inf_{\boldsymbol{v}^{(m)}\in\mathbb{R}^{pK_m}} \left\{ \sum_{t=1}^{n} \left(d[t] - \hat{d}^{(m)}_{\text{batch}}[t]\right)^2 + \delta \left|\left|\boldsymbol{v}^{(m)}\right|\right|^2 \right\} \right] \leq O\left(p \log^2(n)\right),$$

for any n with computational complexity upper bounded by $O(t)$ at each time instance t, where $\mathcal{M}_n$ represents the set of all hierarchical models with at most $O(\log(n))$ leaves on the incremental decision tree at time n, $\hat{d}^{(m)}_{\text{batch}}[t]$ is the prediction of the mth model in the set $\mathcal{M}_n$ whose parameter vectors at each node are chosen non-causally (which needs the knowledge of the final decision tree in advance of the processing), K_m is the number of partitions in the mth model, i.e., $K_m \leq O(\log(n))$, $\forall m \in \mathcal{M}_n$, and $\boldsymbol{v}^{(m)}$ is the vector constructed by concatenating the parameter vectors at each node on the mth model.

This theorem indicates that the introduced algorithm can asymptotically and sequentially achieve the performance of any piecewise model in the set $\mathcal{M}_n$, i.e., the piecewise models having at most $O(\log(n))$ leaves defined on the tree. In particular, over any unknown length of data n, the performance of the piecewise models with $O(\log(n))$ leaves can be sequentially achieved by the introduced algorithm with a regret upper bounded by $O(p\log^2(n))$. In this sense, we do not compare the performance of the introduced algorithm with a class of regressors that is fixed over any length of data n. Instead, the regret of the introduced algorithm is defined with respect to a set of piecewise linear regressors whose number of partitions are upper bounded by $O(\log(n))$, i.e., the competition class grows as n increases. In the conventional tree based regression methods, the depth of the tree is set before processing starts and the performance of the regressor is highly sensitive with respect to the unknown length of data. For example, if the depth of the tree is large whereas there are not enough data samples, then the piecewise model will be undertrained and yield an unsatisfactory performance. Similarly, if the depth of the tree is small whereas huge number of data samples are available, then trees (and regressors) with higher depths (and finer regions) can be better trained. As shown in Theorem 1, the introduced algorithm elegantly and intrinsically makes such decisions and performs asymptotically as well as any piecewise regressor in the competition class that grows exponentially in n. Such a significant performance is achieved with computational complexity upper bounded by $O(n)$, i.e., only linear in the length of data, whereas the number of different piecewise models defined on the incremental decision tree can be in the order of 1.5^n [31]. Moreover, under certain regularity conditions, the computational complexity of the algorithm is $O(\log(n))$ as will be discussed in Remark 1. This theorem is an intermediate step to show that the introduced algorithm yields the desired performance guarantee in (1), and will be used to prove the next theorem.

Using Theorem 1, we introduce another theorem presenting the main result of the paper, where we define the performance of the introduced algorithm with respect to the class of twice differentiable functions as in (1).

Theorem 2. *Let $\{d[t]\}_{t\geq 1}$ and $\{\boldsymbol{x}[t]\}_{t\geq 1}$ be arbitrary, bounded, and real-valued sequences of data and regressor vectors, respectively. Let $\mathcal{F}$ be the class of all twice differentiable functions such that $\forall f \in \mathcal{F}$, $\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} \leq D < \infty$, $i, j = 1, \ldots, p$ and $\hat{d}_f[t] = f(\boldsymbol{x}[t])$. Then, Algorithm 1, whose prediction at time t is $\hat{d}[t]$, yields*

$$\sum_{t=1}^{n} \left(d[t] - \hat{d}[t]\right)^2 - \inf_{f\in\mathcal{F}} \sum_{t=1}^{n} \left(d[t] - \hat{d}_f[t]\right)^2 \leq o(p^2 n),$$

for any n with computational complexity upper bounded by $O(t)$ at each time t.

This theorem presents the nonlinear modeling power of the introduced algorithm. Specifically, it states that the introduced algorithm can asymptotically achieve the performance of the optimal twice differentiable function that is selected after observing the entire data in hindsight.

6. Construction of the algorithm

In this section, we first introduce several lemmas before proving the theorems. In particular, we first introduce a weighting procedure over the incremental decision tree at time n (i.e., the final decision tree) and construct a regressor using this weighting. The resulting regressor is non-causal since the final decision tree needs to be known in advance of the processing. We then derive a regret upper bound on the performance of this non-causal regression algorithm. We next introduce a weighting procedure, whose values at time t are calculated using the incremental decision tree at time t. Using this new weights, we introduce a causal regression algorithm and show that it achieves the same performance as the aforementioned non-causal regressor. Following this procedure, we construct our algorithm and prove our results.

Let $\hat{d}_\kappa[t]$ denote the prediction of node κ at time t, where this predictor can be chosen arbitrarily. According to these prediction values, we assign a performance dependent weight to each node

on the incremental decision tree at time n as follows

$$P_\kappa(n) \triangleq \begin{cases} \exp\left\{-\frac{1}{2a}\sum_{\substack{t\le n\\ \boldsymbol{x}[t]\in\mathcal{R}_\kappa}}(d[t]-\delta_\kappa[t])^2\right\}, & \text{if } \kappa\in\mathcal{L}_n \\ \frac{1}{2}P_{\kappa 0}(n)P_{\kappa 1}(n)+\frac{1}{2}\exp & \\ \quad\left\{-\frac{1}{2a}\sum_{\substack{t\le n\\ \boldsymbol{x}[t]\in\mathcal{R}_\kappa}}(d[t]-\delta_\kappa[t])^2\right\}, & \text{otherwise,} \end{cases} \tag{3}$$

where we set

$$\delta_\kappa[t] \triangleq \begin{cases} \hat{d}_{\kappa_t}[t], & \text{if } \kappa\notin\mathcal{N}_t, \\ \hat{d}_\kappa[t], & \text{otherwise,} \end{cases} \tag{4}$$

with $\kappa_t \in \mathcal{L}_t \cap \mathcal{J}(\kappa)$ representing the closest ancestor of κ that is available on the incremental tree at time t. Also, a is a positive constant related to the learning rate of the algorithm and we set it to $a \ge 4A^2$ as explained in Lemma 4. In our algorithm, $1/a$ can be considered as the step size, hence, a smaller value for a results in a faster algorithm. However, as pointed out in Lemma 4, there is a minimum value for a to guarantee the convergence of the algorithm. In (4), for any node that is on the final decision tree but not on the incremental decision tree at time t, we set its prediction to be equal to the prediction of its closest prefix that is on the incremental decision tree at time t. In this sense, $\delta_\kappa[t]$ can be considered as a pseudo-predictor of the original predictor $\hat{d}_\kappa[t]$.

We use the weights in (3) to obtain performance guarantees for the models defined on the incremental decision tree. To this end, we introduce the following lemmas. All of the proofs are provided in the supplementary material.

Lemma 1. *The weight of the root node λ (according to (3)) can be obtained as*

$$P_\lambda(n) = \sum_{m\in\mathcal{M}_n} 2^{-B_m}\exp\left\{-\frac{1}{2a}\sum_{t=1}^{n}\left(d[t]-\delta^{(m)}[t]\right)^2\right\}, \tag{5}$$

where $\delta^{(m)}[t]=\delta_\kappa[t]$ for $\kappa\in\mathcal{L}(m)$ such that $\boldsymbol{x}[t]\in\mathcal{R}_\kappa$, B_m represents the number of bits required to represent the model m on the binary tree using a universal code (e.g., [35]), $\mathcal{L}(m)$ represents the set of all disjoint regions (i.e., nodes) in the mth model, and $\mathcal{M}_n$ represents the set of all hierarchical models defined on the incremental decision tree at time n.

We next introduce the following lemma, by which we relate the performance of the original regressors to the weighting function in (3).

Lemma 2. *According to the definitions in (3) and (4), we have*

$$-2a\ln\left(P_\lambda(n)\right) \le \min_{m\in\mathcal{M}_n}\left\{\sum_{t=1}^{n}\left(d[t]-\hat{d}^{(m)}[t]\right)^2\right\} + \left(2a\ln(2)+4A^2\right)O(\log(n)). \tag{6}$$

Hence, we obtain a weighting assignment achieving the performance of the optimal piecewise linear model. We present the following lemma to introduce a low complexity sequential algorithm.

Lemma 3. *Assume that $\boldsymbol{x}[t]\in\mathcal{R}_\kappa$ for some $\kappa\in\mathcal{L}_n$. Then, we can write*

$$P_\lambda(t-1) = \sum_{\kappa_i\in\mathcal{J}(\kappa)}\pi_{\kappa_i}[t-1]\exp\left\{-\frac{1}{2a}\sum_{\substack{t'<t\\ \boldsymbol{x}[t']\in\mathcal{R}_{\kappa_i}}}\left(d[t']-\delta_{\kappa_i}[t']\right)^2\right\}, \tag{7}$$

where $\kappa_i\in\mathcal{J}(\kappa)$ is the string formed from the first i letters of $\kappa=\nu_1\dots\nu_l$ and

$$\pi_{\kappa_i}[t] \triangleq \begin{cases} \frac{1}{2}, & \text{if } i=0 \\ \frac{1}{2}P_{\kappa_{i-1}\nu_i^c}(t-1)\,\pi_{\kappa_{i-1}}[t], & \text{if } 1\le i\le l-1. \\ P_{\kappa_{i-1}\nu_i^c}(t-1)\,\pi_{\kappa_{i-1}}[t], & \text{if } i=l \end{cases} \tag{8}$$

We use this lemma to construct a sequential algorithm achieving the regret bound in Lemma 2. To this end, we define the following predictor

$$\hat{d}[t] \triangleq \sum_{\kappa_i\in\mathcal{J}(\kappa)}\mu_{\kappa_i}[t-1]\,\delta_{\kappa_i}[t], \text{ where} \tag{9}$$

$$\mu_{\kappa_i}[t-1] \triangleq \frac{\pi_{\kappa_i}[t-1]\exp\left\{-\frac{1}{2a}\sum_{\substack{t'<t\\ \boldsymbol{x}[t']\in\mathcal{R}_{\kappa_i}}}\left(d[t']-\delta_{\kappa_i}[t']\right)^2\right\}}{P_\lambda(t-1)}. \tag{10}$$

The exponentially lifted losses $\exp\{-\frac{1}{2a}\sum_{t'<t\boldsymbol{x}[t']:\in\mathcal{R}_{\kappa_i}}(d[t']-\delta_{\kappa_i}[t'])^2\}$ in node κ_i accumulated until time $t-1$ in (10) is being referred to as E_{κ_i} in Algorithm 2, where the time index is dropped for simplicity. Note that the sum of E_{κ_i}'s after weighting with π_{κ_i}'s over nodes from κ to λ yields P_λ, the total weighted performance of all hierarchical models in $\mathcal{M}_n$ (cf. Lemma 1). Therefore, normalization of E_{κ_i} (that is weighted by π_{κ_i}) by P_λ gives the node weight μ_{κ_i}, which we exploit in constructing our algorithm. Also, the calculation of E_{κ_i} accepts recursive updates, i.e., update with $\boldsymbol{x}[t']\in\mathcal{R}_{\kappa_i}$: $E_{\kappa_i}=E_{\kappa_i}\exp(-(d[t']-\hat{d}_{\kappa_i})^2/(2a))$, where $E_{\kappa_i}=1$ is set initially (Algorithm 3 and Algorithm 4). In the next lemma, we relate the performance of this predictor in (9) to the weight of the root node. In this way, we relate the performance of the sequential predictor in (9) to the performance of the best piecewise model defined on the incremental decision tree using Lemma 2.

Lemma 4. *For any $a\ge 4A^2$, the sequential predictor in (9) achieves*

$$\sum_{t=1}^{n}\left(d[t]-\hat{d}[t]\right)^2 \le -2a\ln(P_\lambda(n)). \tag{11}$$

Although in Lemma 4 we presented a performance guarantee to the sequential predictor in (9), this predictor still needs to know the final decision tree in advance since we assumed $\kappa\in\mathcal{L}_n$. In particular, the summation in (9) is over the final decision tree at time n, whereas we only have access to the nodes on the incremental decision tree at time t. To remove this assumption, we use the definition of the predictors $\delta_{\kappa_i}[t]$ given in (4) and introduce the following weighting

$$\widetilde{P}_\kappa(t) \triangleq \begin{cases} \exp\left\{-\frac{1}{2a}\sum_{\substack{t'\le t\\ \boldsymbol{x}[t']\in\mathcal{R}_\kappa}}\left(d[t']-\delta_\kappa[t']\right)^2\right\}, & \text{if } \kappa\in\mathcal{L}_t \\ \frac{1}{2}\widetilde{P}_{\kappa 0}(t)\widetilde{P}_{\kappa 1}(t)+\frac{1}{2}\exp & \\ \quad\left\{-\frac{1}{2a}\sum_{\substack{t'\le t\\ \boldsymbol{x}[t']\in\mathcal{R}_\kappa}}\left(d[t']-\delta_\kappa[t']\right)^2\right\}, & \text{otherwise} \end{cases}, \tag{12}$$

$\forall\kappa\in\mathcal{N}_t$. Note that this weighting is over the incremental decision tree that is available at time t. Using this new weighting over the incremental decision tree, our aim is to design a sequential algorithm that achieves the performance of the predictor in (9) without the knowledge of the final incremental decision tree at time n. To this end, we first introduce the following lemma.

Lemma 5. *For all nodes on the final incremental decision tree at time n (but not at an intermediate time t), i.e., $\forall\kappa\in\mathcal{L}_t\cup(\mathcal{N}_n-\mathcal{N}_t)$, we have*

$$P_\kappa(t) = \exp\left\{-\frac{1}{2a}\sum_{\substack{t'\le t\\ \boldsymbol{x}[t']\in\mathcal{R}_\kappa}}\left(d[t']-\delta_\kappa[t']\right)^2\right\}. \tag{13}$$

We next introduce the following corollary illustrating that the weights $\widetilde{P}_\kappa(t)$ are the same as the weights $P_\kappa(t)$ over the incremental decision tree at time t.

Corollary 1. *The weights in (3) and (12) satisfy* $P_\kappa(t) = \widetilde{P}_\kappa(t)$, $\forall \kappa \in \mathcal{N}_t$.

This corollary directly follows from the definitions in (3) and (12) as well as Lemma 5, hence its proof is omitted.

Using this new weighting over the incremental decision tree at time t, our aim is to introduce a sequential algorithm over this incremental decision tree at time t. To this end, (9) can be written as

$$\hat{d}[t] = \sum_{\kappa_i \in \mathcal{J}(\kappa_r)} \widetilde{\mu}_{\kappa_i}[t-1]\, \hat{d}_{\kappa_i}[t], \tag{14}$$

where $\kappa_r \in \mathcal{J}(\kappa) \cap \mathcal{L}_t$ is the leaf node (with depth r) on the incremental decision tree at time t containing the current regressor vector, i.e., $\boldsymbol{x}[t] \in \mathcal{R}_{\kappa_r}$, and

$$\widetilde{\mu}_{\kappa_i}[t] \triangleq \begin{cases} \mu_{\kappa_i}[t], & \text{if } i < r \\ \sum_{j=r}^{l} \mu_{\kappa_j}[t], & \text{if } i = r \end{cases}. \tag{15}$$

Here, we emphasize that the summation in (14) is over the incremental decision tree at time t, whereas $\widetilde{\mu}_{\kappa_i}$'s are still defined using the parameters over the incremental decision tree at time n. In order to construct $\widetilde{\mu}_{\kappa_i}$'s with the parameters over the incremental decision tree at time t, we introduce the following lemma.

Lemma 6. *Letting*

$$\widetilde{\pi}_{\kappa_i}[t] \triangleq \begin{cases} \frac{1}{2}, & \text{if } i = 0 \\ \frac{1}{2}\widetilde{P}_{\kappa_{i-1}v_i^c}(t-1)\, \widetilde{\pi}_{\kappa_{i-1}}[t], & \text{if } 1 \le i \le r-1 \\ \widetilde{P}_{\kappa_{i-1}v_i^c}(t-1)\, \widetilde{\pi}_{\kappa_{i-1}}[t], & \text{if } i = r \end{cases}, \tag{16}$$

$\forall i \le r$, *we obtain* $:\widetilde{\mu}_{\kappa_i}[t]$

$$= \frac{\widetilde{\pi}_{\kappa_i}[t-1] \exp\left(-\frac{1}{2a}\sum_{t'<t,\ \boldsymbol{x}[t'] \in \mathcal{R}_{\kappa_i}} \left(d[t'] - \delta_{\kappa_i}[t']\right)^2\right)}{\widetilde{P}_\lambda(t-1)}. \tag{17}$$

This lemma illustrates that we can obtain both $\widetilde{\mu}_{\kappa_i}[t-1]$ and $\hat{d}_{\kappa_i}[t]$, $\forall i \le r$ using the incremental decision tree at time t to construct the predictor in (14). Thus, our algorithm does not require any knowledge on the final incremental decision tree at time n and a description of this prediction is provided in Algorithm 2, where $\boldsymbol{w}_{\kappa_i}$ denotes the linear regressor at the node κ_i. Observe that in line 6 of Algorithm 2, the final output $\hat{d}$ is computed by a linear combination of the node estimates of all nodes in $\mathcal{J}(\kappa)$. A regret bound on the performance of the universal piecewise linear regressor in (2) is given in the following lemma.

Lemma 7. *For any* $m \in \mathcal{M}_n$ *having* $K_m = |\mathcal{L}(m)|$ *disjoint regions, the piecewise linear regressor in (2) achieves the following performance guarantee*

$$\sum_{t=1}^{n} \left(d[t] - \hat{d}^{(m)}[t]\right)^2 - \min_{\boldsymbol{v}^{(m)} \in \mathbb{R}^{pK_m}} \left\{ \sum_{t=1}^{n} \left(d[t] - \hat{d}^{(m)}_{\text{batch}}[t]\right)^2 + \delta \left|\left|\boldsymbol{v}^{(m)}\right|\right|^2 \right\} \le A^2 K_m p \ln(n/K_m) + O(1), \tag{18}$$

where $\hat{d}^{(m)}_{\text{batch}}[t] = \boldsymbol{v}_\kappa^T \boldsymbol{x}[t]$ *such that* $\kappa \in \mathcal{L}(m)$ *with* $\boldsymbol{x}[t] \in \mathcal{R}_\kappa$, *and* $\boldsymbol{v}^{(m)}$ *is the vector of concatenating the parameter vectors at each node on the mth model (i.e., letting* $\mathcal{L}(m) = \{\kappa^{(1)}, \ldots, \kappa^{(K_m)}\}$, *we have* $\boldsymbol{v}^{(m)} = [\boldsymbol{v}^T_{\kappa^{(1)}}, \ldots, \boldsymbol{v}^T_{\kappa^{(K_m)}}]^T$*).*

We emphasize that in each region of a piecewise model, different learning algorithms (not necessarily the above universal piecewise linear regressor), e.g., different linear regressors or nonlinear ones, from the broad literature can be used. Although the main contribution of this paper is the hierarchical organization and efficient management of these piecewise models, we also discuss the implementation of the universal piecewise linear model of Singer et al. [36] into our framework for completeness in Algorithms 3 and 4. When a new sample falls into the region $\mathcal{R}_\kappa$, where κ is a leaf node and $\alpha_\kappa = 1$, we split the node using the Algorithm 3, which distributes the set of accumulated regressor vectors $\mathcal{T}_\kappa$ among its children and trains a different linear regressor in each of these children nodes. However, we do not update $\mathcal{T}_\kappa$ in Algorithm 3, instead, we update it in line 7 of Algorithm 1. Moreover, Algorithm 4 updates the linear regression parameters of all nodes in $\mathcal{J}(\kappa)$, i.e., all nodes containing the current sample that contribute to the current estimation.

We use the discussed lemmas to prove Theorem 1. Then, we prove Theorem 2 using Theorem 1. Proofs of theorems and lemmas are provided in the supplementary material.

Remark 1. Algorithm 1 achieves the performance of the best piecewise linear model having $O(\log(n))$ partitions with a regret of $O(p\log^2(n))$. In the most generic case of an arbitrary piecewise model m having $O(K_m)$ partitions, the introduced algorithm still achieves a regret of $O(pK_m \log(n/K_m))$. This indicates that for models having $O(n)$ partitions, the introduced algorithm achieves a regret of $O(pn)$, hence, the performance of the piecewise model cannot be asymptotically achieved. However, we emphasize that no other algorithm can achieve a smaller regret than $O(pn)$ as shown by Kozat et al. [22], i.e., the introduced algorithm is optimal in a strong minimax sense. Intuitively, this lower bound can be justified by considering the case in which the regressor vector at time t falls into the tth region of the piecewise model.

Remark 2. Consider that the regressor vectors are i.i.d. with a continuous pdf f over $[-A, A]^p$. If $\sup_{\boldsymbol{x} \in [-A,A]^p} f(\boldsymbol{x}) / \inf_{\boldsymbol{x} \in [-A,A]^p} f(\boldsymbol{x}) = O(1)$, then the average computational complexity of the algorithm is $O(\log n)$. To justify this statement, we can quantize the given pdf f over intervals of length ϵ, where $\epsilon > 0$ is arbitrary. Since the data is uniformly distributed in every ϵ interval with respect to this quantized pdf, then given that n_1 data points have fallen into the first ϵ interval, our algorithm will create a depth-$\log(n_1)$ complete subtree as $n_1 \to \infty$ over this ϵ interval. Therefore, the running time of the algorithm will be $\log(n_1)$ in average over this interval. To generalize this behavior, let f_i be the value of the quantized pdf for the ith ϵ interval. Then, we have $\sum_{i=1}^{2A/\epsilon} f_i = 1/\epsilon$ since the area under the pdf curve should be 1. Therefore, given that we observe n data points in total, each subtree growing in these ϵ intervals will contain $O(n/\epsilon)$ data points since $f_i/f_j = O(1)$ for any pair of i and j according to our assumption. Therefore, each of these subtrees will grow in the order of $O(\log(n)/\epsilon)$, which will result in a computational complexity of $O(\log(n))$ in average. Since the quantized pdf can arbitrarily approximate the original pdf for any continuous distribution, the statement follows.

Remark 3. As mentioned in Remark 1, no algorithm can converge to the performance of the piecewise linear models having $O(n)$ disjoint regions. Therefore, we can limit the maximum depth of the tree by $O(\log(t))$ at each time t to achieve a low complexity implementation. With this limitation and according to the update rule of the tree, we can observe that while dividing a region into two disjoint regions, we may be forced to perform $O(t)$ computations due to the accumulated regressor vectors (since their number can be as large as t). However, since a regressor vector is processed by at most $O(\log(t))$ nodes for any t, the average computational complexity of the update rule of the tree remains to be upper bounded by $O(\log(n))$. Furthermore, the performance of this low complexity implementation will be asymptotically the same as the exact implementation provided that the regressor vectors are evenly dis-

tributed in the regressor space, i.e., they are not gathered around a considerably small neighborhood.

Corollary 2. *Let* $\{d[t]\}_{t\geq 1}$ *and* $\{\boldsymbol{x}[t]\}_{t\geq 1}$ *be arbitrary, bounded, and real-valued sequences of data and regressor vectors, respectively. Let* $\mathcal{F}$ *be the class of all twice differentiable functions such that* $\forall f \in \mathcal{F}$, $\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} \leq D < \infty$, $i, j = 1, \ldots, p$ *and* $\hat{d}_f[t] = f(\boldsymbol{x}[t])$. *Consider an arbitrary partitioning rule satisfying* $|x_i - y_i| \leq \frac{2A}{r(d)}$, $\forall i \in \{1, \ldots, p\}$ *and* $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{R}_\kappa$, *where* κ *is any depth-*d *node on the incremental decision tree and* $r(d)$ *is an appropriate deterministic real-valued function with domain of integers. Then, Algorithm 1 with this partitioning rule, whose prediction at time* t *is* $\hat{d}[t]$, *yields*

$$\sum_{t=1}^{n} \left(d[t] - \hat{d}[t]\right)^2 - \inf_{f\in\mathcal{F}} \sum_{t=1}^{n} \left(d[t] - \hat{d}_f[t]\right)^2 \leq O\left(\frac{p^2 n}{r^2(\log(n))}\right),$$

for any n *with computational complexity upper bounded by* $O(t)$ *at each time* t.

The proof of Corollary 2 is provided in the supplementary material.

The inequality $|x_i - y_i| \leq \frac{2A}{r(d)}$ in Corollary 2 illustrates that the diameter of the regions of the nodes with depth-d is upper bounded by $\frac{2A}{r(d)}$. Such a function $r(d)$ always exists for any partitioning since we can always pick $r(d) = 1$, $\forall d$.

Remark 4. Ideally, relatively simpler models in $\mathcal{M}_n$ should get larger weights initially; and the weights of the relatively more complex models should gradually increase as more data become available. This ideal adaptive weighting over the introduced dynamically growing regression tree models in $\mathcal{M}_n$ can be achieved by ensuring the two diminishing rates of the following regrets to be similar: (1) regret between the overall technique and the best sequential model predictors (Lemma 2) and (2) regret between the best sequential predictor and the corresponding batch predictor (Lemma 7). We emphasize that these two desirable properties are elegantly achieved by our weighting as the upper bounds in Lemmas 2 and 7 are asymptotically similar, i.e., $\frac{1}{n}(O(\log(n) - p\log^2(n))) \to 0$ (cf. the proof of Theorem 1). In comparison, other schemes such as the uniform weighting failing to achieve this would not be suitable.

7. Experiments

In this section, we investigate the performance of the introduced algorithm in comparison to various methods including " M5' " and "CUBIST" [26,27] under several benchmark scenarios. Throughout the experiments, we denote the introduced incremental decision tree technique in Algorithm 1 by "IDT", the context tree weighting algorithm of Kozat et al. [22] by "CTW", the linear regressor by "LR", the Volterra series regressor by "VSR" [9], the sliding window multivariate adaptive regression splines of Friedman [37,38] by "MARS", and the Fourier nonlinear regressor of Carini and Sicuranza [39] by "FNR". The combination weights of the LR, VSR, and FNR are updated using the recursive least squares (RLS) algorithm [33]. Unless otherwise stated, the CTW algorithm has depth 2, the VSR, FNR, and MARS algorithms are second order, and the MARS algorithm uses 21 knots with a window length of 500 that shifts in every 200 samples. Also, we use $\delta = 0.1$ as the regularization parameter in our algorithm. When we have access to unlabeled data instances, we set $a = 4A^2$ with A being an upper bound on the feature vector attributes. For data generated by random models with infinite support, we set $a = 4A^2$, where $A = 4\delta$ is an upper bound on the features with high probability, e.g., more than 99% when each feature is generated by $N(0, \delta^2)$. When we do not have access to feature vectors, a can be set robustly to guarantee that $\sqrt{a/4}$ is an upper bound for features.

Table 1
Comparison of complexities of the proposed algorithms with the corresponding update rules. In the table, p represents the dimensionality of the regressor space, d represents the depth of the trees in the respective algorithms, and r represents the order of the corresponding filters and algorithms. For the MARS algorithm (particularly, the fast MARS algorithm, cf. [38]), b represents the number of basis functions and w represents the window length.

Algorithm	Computational complexity	Space complexity
IDT	$O(p^2\log(n))$	$O(p^2 n)$
CTW	$O(p^2 d)$	$O(p^2 2^d)$
LR	$O(p^2)$	$O(p^2)$
VSR	$O(p^{2r})$	$O(p^{2r})$
MARS	$O(rbw^3)$	$O(rbw)$
FNR	$O((pr)^{2r})$	$O((pr)^{2r})$

In Table 1, we provide the computational and space complexity of the proposed algorithms. We emphasize that although the computational complexity to create and run the incremental decision tree is $O(\log(n))$, the overall computational complexity of the algorithm is $O(p^2\log(n))$ due to the universal linear regressors at each region. Particularly, since the universal linear regressor at each region has a computational complexity of $O(p^2)$, the overall computational complexity of $O(p^2\log(n))$ follows. However, this universal linear regressor can be straightforwardly replaced with any linear (or nonlinear) regressor in the literature. For example, if we use the LMS algorithm to update the parameters of the linear regressor instead of using the universal algorithm for this update, the computational complexity of the overall structure becomes $O(p\log(n))$. Hence, although the computational complexity of the original IDT algorithm is $O(\log(n))$, this computational complexity may increase according to the computational complexity of the node regressors.

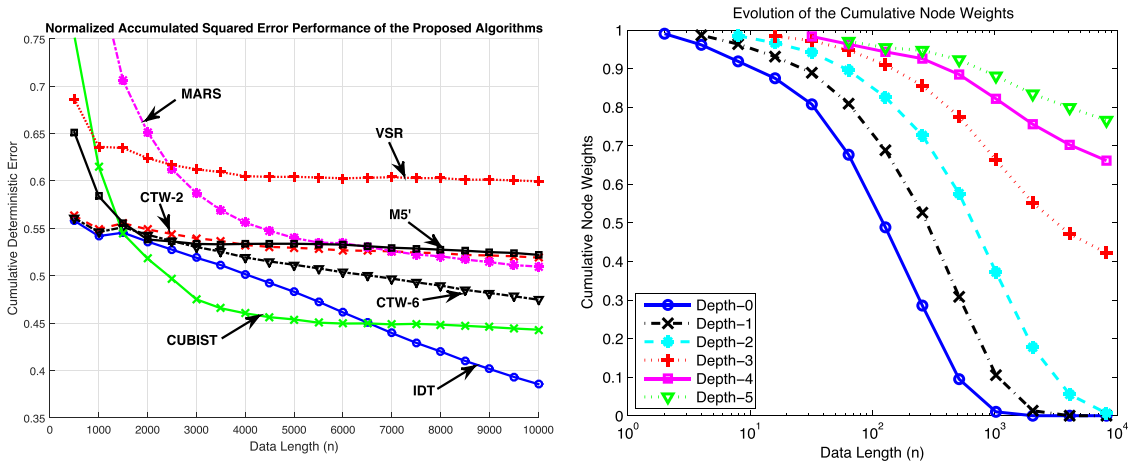
In this section, we first illustrate the performances of the proposed algorithms for a synthetic piecewise linear model that do not match the modeling structure of any of the above algorithms. We then consider the prediction of well-known data sequences such as Mackey–Glass sequence and Chua's circuit [13]. Finally, we consider the prediction of real life examples that can be found in various benchmark data set repositories (e.g., [40,41]).

Synthetic data: In this part, we consider the scenario where the desired data is generated by the following piecewise linear model

$$d[t] = \begin{cases} x_1[t] + x_2[t] + n[t], & \text{if } ||\boldsymbol{x}[t]||^2 \in [0, 0.1] \cup [0.5, 1] \\ -x_1[t] - x_2[t] + n[t], & \text{otherwise} \end{cases}, \tag{19}$$

and $\boldsymbol{x}[t] = [x_1[t], x_2[t]]^T$ are sample functions of a jointly Gaussian process of mean $[0, 0]^T$ and covariance matrix $\boldsymbol{I}$, and $n[t]$ is a sample function from a zero mean white Gaussian process with variance 0.1. Note that the piecewise model in (19) has circular regions, which cannot be represented by hyperplanes or twice differentiable functions. Hence, the underlying relationship between the desired data and the regressor vectors cannot be exactly modeled using any of the proposed algorithms.

In Fig. 3a, we present the normalized accumulated squared errors of the proposed algorithms averaged over 10 trials. For each of these trials, a different sample function is realized from the defined joint Gaussian process for the regressors $\boldsymbol{x}[t]$ and another sample function is realized from the defined noise process $n[t]$. This yields a set of 10 data sequences of length 10^4, the normalized accumulated squared error performance is obtained for each sequence and then we report the average performance over the generated 10 data sequences. For this experiment, "CTW-2" and "CTW-6" show the performances of the CTW algorithm with depths 2 and 6, respectively. Since the performances of the LR and FNR algorithms are incomparable with the rest of the algorithms, they are



(a) Normalized accumulated squared errors for the piecewise linear model in (19) averaged over 10 trials.

(b) Normalized cumulative node weights for the piecewise linear model (19) averaged over 10 trials.

Fig. 3. Synthetic data simulation results.

not included in the figure for this experiment. Fig. 3a illustrates that even for a highly nonlinear system (19), our algorithm significantly outperforms the other algorithms. The normalized accumulated error of the introduced algorithm approaches to the variance of the noise signal as n increases, unlike the rest of the algorithms, whose performances converge to the performance of their optimal batch variants as n increases. This observation can be seen in Fig. 3a, where the normalized cumulative error of the IDT algorithm steadily decreases since the IDT algorithm creates finer regions as the observed length of data increases. Hence, even for a highly nonlinear model such as the circular piecewise linear model in (19), which cannot be represented via hyperplanes, the IDT algorithm can well approximate this highly nonlinear relationship by incrementally introducing finer partitions as the observed length of data increases.

Furthermore, even though the depth of the introduced algorithm is comparable with the CTW-6 algorithm over short data sequences, the performance of our algorithm is superior to the CTW-6 algorithm. This result follows since the IDT algorithm intrinsically eliminates the extremely finer models at the early processing stages and introduces them when they are needed, unlike the CTW-6 algorithm. This procedure can be observed in Fig. 3b, where the IDT algorithm introduces finer regions (i.e., nodes with higher depths) to the hierarchical model as the coarser regions becomes unsatisfactory. Since the universal algorithms such as CTW distribute a "budget" into numerous experts, as the number of experts increases, the performance of such algorithms deteriorate. On the other hand, the introduced algorithm intrinsically limits the number of experts according to the unknown length of data at each iteration, hence we avoid such possible performance degradations as observed in Fig. 3b.

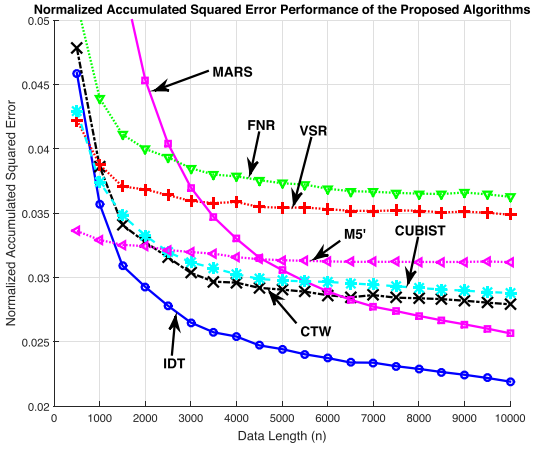
Benchmark Sequences: In this part, we consider the prediction of the Mackey-Glass and Chua's circuit sequences. The Mackey-Glass sequence is defined by the differential equation $\frac{dx[t]}{dt} = \frac{\beta x[t-\tau]}{1+(x[t-\tau])^n} - \gamma x[t]$, where we set $\beta = 2$, $\gamma = 1$, $\tau = 2$, and $n = 10$ with the initial condition $x[t] = 0.5$ for $t < 0$. We generate the time series using the fourth order Runge–Kutta method. The Chua's circuit is generated according to the differential equations $\frac{dx}{dt} = \alpha(y - x - f(x))$, $\frac{dy}{dt} = x - y + z$, $\frac{dz}{dt} = -\beta y$, where we drop the time index for simplicity, and $f(x) = m_1 x + 0.5(m_0 - m_1)(|x+1| - |x-1|)$, $\alpha = 15.6$, $\beta = 28$, $m_0 = -1.143$, $m_1 = -0.714$ initially with $[x, y, z] = [0.7, 0, 0]$ for $t < 0$.

Fig. 4 shows the normalized accumulated squared error performances of the proposed algorithms for the Mackey-Glass and Chua's circuit sequences. Due to the chaotic nature of the signals, we observe non-uniform curves in Fig. 4a and b. In the figures, the algorithms with incomparable (i.e., unsatisfactory) performance are omitted. Fig. 4 presents that the IDT algorithm achieves an average of 20% relative gain in the performance with respect to the other algorithm and can accurately predict these well-known data sequences.

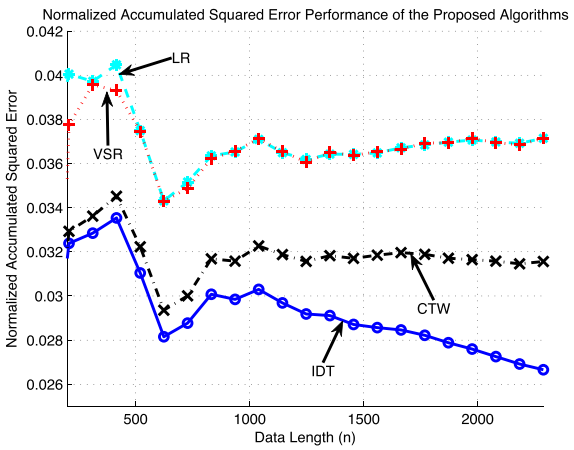
Real data: In this part, we evaluate the performance of the proposed algorithms for several well-known real data sets in machine learning literature.

We first compare the proposed algorithms using "kinematics" and "pumadyn" data sets that are taken from Rasmussen et al. [41]. The kinematics data set involves a realistic simulation of the forward dynamics of an 8 link all-revolute robot arm and the task is to predict the distance of the end-effector from a target. Among its variants, we used the one having 9 attributes and being nonlinear as well as medium noisy. The pumadyn data set involves a realistic simulation of the dynamics of a Puma 560 robot arm. The task in these datasets is to predict the angular acceleration of one of the robot arm's links. Among its variants, we used the one having 9 attributes and being nonlinear as well as medium noisy.

Fig. 5 shows the normalized accumulated squared error performances of the proposed algorithms for the kinematics and pumadyn data sets. In the experiments, all dimensions of the regressor vector and desired data are normalized between $[-1, 1]$. Although the VSR algorithm provides the best performance in Fig. 5a and the MARS algorithm achieves the minimum accumulated error in Fig. 5b, the performances of these algorithms in the reciprocal experiments are highly unsatisfactory. This result implies that the data in the first experiment can be well approximated by Volterra series, whereas the model that generates the data in the second experiment is more in line with B-splines. Hence, the performances of these algorithms are extremely sensitive to the underlying structure that generates the data. On the other hand, the IDT algorithm nearly achieves the performance of the best algorithm in both experiments and presents a desirable performance under different scenarios. This result implies that the introduced algorithm can be used in various frameworks without any significant performance degradations owing to its guaranteed performance upper bounds without any statistical or structural assumptions.

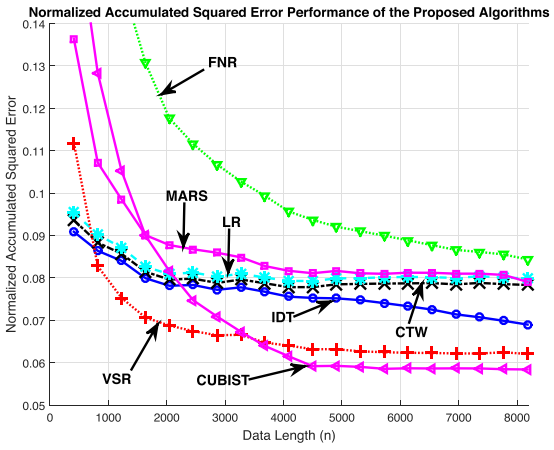


(a) Normalized accumulated squared error performances for the Mackey-Glass sequence.

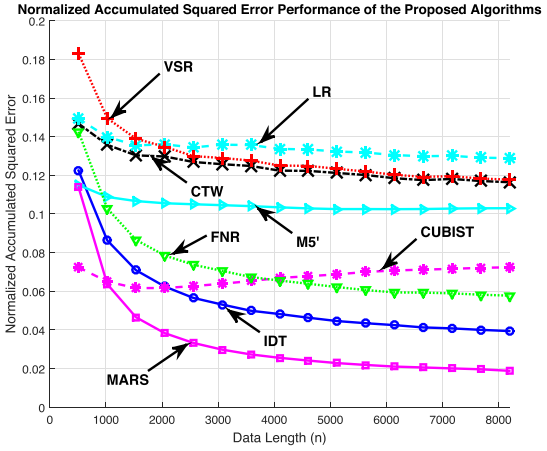


(b) Normalized accumulated squared error performances for the Chua's circuit sequence.

Fig. 4. Results for benchmark sequences.



(a) Normalized accumulated squared errors for the \kinematics" data set.



(b) Normalized accumulated squared errors for the \pumadyn" data set.

Fig. 5. Real data simulation results.

Table 2
Squared errors of the proposed algorithms for various benchmark data sets, where each dimension of the data sets are scaled between $[-1, 1]$.

	IDT	FNR	MARS	CTW	LR	VSR	M5'	Cubist
Dolphin [42]	0.0221	0.0290	0.0293	0.0235	0.0253	0.0223	**0.0004**	0.0008
LMPAVW [43]	0.0314	0.0984	0.0934	0.1115	0.1153	0.1122	0.052	**0.0309**
CCPP [40]	0.0129	0.0138	0.0178	0.0131	0.0147	0.0129	0.0004	**0.0001**
Protein Tertiary [40]	0.2195	0.2224	0.2468	0.2379	0.2458	0.2257	0.1524	**0.1153**
Lorenz Attractor [44]	**0.0336**	0.0389	0.0396	0.0346	0.0404	0.0354	0.0756	0.0401
Hwang-SIF [41]	**0.0190**	0.0952	0.0530	0.0919	0.3036	0.1617	0.1313	0.1195
Hwang-HF [41]	0.1331	0.1991	0.1867	0.2388	0.3225	0.3243	0.0727	**0.0661**
Hwang-AF [41]	**0.1705**	0.2578	0.2412	0.2688	0.3340	0.3007	0.2251	0.1822
Hwang-CIF [41]	0.1699	0.2464	0.2325	0.2512	0.3334	0.2800	0.1306	**0.1220**

In Table 2, we present the performance of the proposed algorithms for various benchmark data sets that are widely used in machine learning literature.

The proposed IDT algorithm significantly outperforms the competitor algorithms (cf. Table 2). In the first four rows of the table, the squared errors of the proposed algorithms can be found for the corresponding data sets that are collected from real life applications. In the "Dolphin", "CCPP", and "Protein Tertiary" data sets, the linear regressor can achieve a comparable performance with respect to the competitor nonlinear learning algorithms. This indicates that the underlying data sequences are highly noisy so that we cannot sufficiently train nonlinear models. On the other hand, for the "LMPAVW" data set, the performance of the linear regressor is significantly outperformed by the nonlinear regressors (e.g., the "FNR" and "MARS" algorithms). Thus, this data may be accurately modeled by some nonlinear function. As a consequence, the proposed IDT algorithm efficiently approximates this nonlinear function and presents a performance gain around $300-400\%$ with respect to the competitor algorithms. These results indicate that the proposed IDT algorithm can be used in real life applications, es-

pecially in ones that include high levels of nonlinearity. This conclusion can also be observed from the results of our experiments with the "Lorenz Attractor", which is used to model atmospheric convection.

Cubist builds on M5 by smoothing the prediction of the leaves with its predecessors. Since this smoothing does not significantly improve the performance for our synthetic data, Cubist does not yield a remarkable performance improvement in that case. On the other hand, real-world data usually come from a sample distribution and usually the target predictor is smooth. That is the reason why modeling power of both of our tree-based regressor and the suitable smoothing operator yield a decent performance in real data.

In the last four rows of Table 2, we present the performance of the algorithms for benchmark data sets that are synthetically generated using various test functions. We note that the competitor algorithms assume fixed basis functions and perform regression over the space spanned by these functions. Consequently, the performances of the competitor algorithms are highly dependent on the portion of the data energy that lies in this space. On the other hand, the proposed IDT algorithm can well approximate these unknown test functions using piecewise models, and hence achieves a significantly better squared error performance than the competitor algorithms.

8. Concluding remarks

We proposed an incremental decision tree method to solve online nonlinear regression problems with bounded but otherwise arbitrary feature vectors. The incremental decision tree is used to partition the feature vector space, where each partition is represented by a leaf node of the tree. The number of nodes in the tree is incremented as our model observes new data samples and these increments are made using a universal rule. In particular, the structure of the partitioning of the feature space, i.e., the splitting rule at the tree nodes, is fixed a priori of the data processing, but a new partition from our partitioning rule is introduced into the model only when it contains at least one feature vector. Hence, our model can be viewed as an infinite dimensional decision tree that is pruned according to the data. We assigned an independent regressor to each node of the tree and combined their outputs using the exponential weights algorithm. We proved that the proposed method asymptotically achieves the performance of the best pruning of $O(\log(n))$-depth defined on the tree and the performance of the optimal twice differentiable regressor with bounded second derivative. This performance is achieved with only computational complexity that is logarithmic in the data length n, i.e., $O(\log(n))$ (under regularity conditions). We demonstrated the superior performance of the introduced algorithm over a series of benchmark applications in the regression literature.

An interesting problem which is left open is to understand the convergence properties of the algorithm with data-driven partitioning rules. It is numerically observed in [11] that data-driven space partitioning methods usually outperform universal partitioning rules. However, implementation of such methods into incremental decision trees seems to be challenging, since the number of data instances that has fallen into each partition changes as the partitioning of the space varies. Hence, the theoretical tractability of such a method is extremely challenging and remains an open problem that provides a future research direction. A limitation of the proposed method is that its computational complexity grows with n. Ideally, one would want to use an infinite depth decision tree with constant number of floating point operations. This paper introduces an efficient method for "backward pruning, i.e., an efficient pruning of the leaf nodes of the infinite depth decision tree into a finite depth tree. On the other hand, one can forward prune the root node by deleting it and passing the data it has to its children, whereby defining independent decision trees with distinct roots. Note that similar ideas have been exploited in [27]. This approach would decrease the computational complexity of the algorithm and potentially provides a more efficient method. Yet, the theoretical analysis of this method seems nontrivial even with the tools introduced in this paper. We consider this as future work. Numerical results suggest that the proposed method works particularly well when the ratio between number of data points and the dimension of the feature space is large, whereas its performance degrades as this ratio gets smaller, which is consistent with the theoretical findings.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.patcog.2018.08.014.

References

[1] K. Lau, Q. Wu, Local prediction of non-linear time series using support vector regression, Pattern Recognit. 41 (2008) 1539–1547.
[2] J. Read, L. Martino, J. Hollmén, Multi-label methods for prediction with sequential data, Pattern Recognit. 63 (2017) 45–55.
[3] I. Naseem, R. Togneri, M. Bennamoun, Robust regression for face recognition, Pattern Recognit. 45 (2012) 104–118.
[4] Q. Liu, J. Yang, J. Deng, K. Zhang, Robust facial landmark tracking via cascade regression, Pattern Recognit. 66 (2017) 53–62.
[5] S.-I. Jang, K. Choi, K.-A. Toh, A.B.J. Teoh, J. Kim, Object tracking based on an online learning network with total error rate minimization, Pattern Recognit. 48 (2015) 126–139.
[6] J. Du, Y. Xu, Hierarchical deep neural network for multivariate regression, Pattern Recognit. 63 (2017) 149–157.
[7] A. Krzyzak, T. Linder, C. Lugosi, Nonparametric estimation and classification using radial basis function nets and empirical risk minimization, IEEE Trans. Neural Netw. 7 (1996) 475–487.
[8] A. Krzyzak, T. Linder, Radial basis function networks and complexity regularization in function learning, IEEE Trans. Neural Netw. 9 (1998) 247–256.
[9] M. Schetzen, The Volterra and Wiener Theories of Nonlinear Systems, Krieger Publishing Co., Inc., Melbourne, FL, USA, 2006.
[10] M. Scarpiniti, D. Comminiello, R. Parisi, A. Uncini, Nonlinear spline adaptive filtering, Signal Process. 93 (2013) 772–783.
[11] N.D. Vanli, S.S. Kozat, A comprehensive approach to universal piecewise nonlinear regression based on trees, IEEE Trans. Signal Process. 62 (2014) 5471–5486.
[12] N.D. Vanli, S.S. Kozat, A unified approach to universal prediction: generalized upper and lower bounds, IEEE Trans. Neural Netw. Learn. Syst. 26 (2015) 646–651.
[13] O.J.J. Michel, A.O. Hero, A.E. Badel, Tree-structured nonlinear signal modeling and prediction, IEEE Trans. Signal Process. 47 (1999) 3027–3041.
[14] D.P. Helmbold, R.E. Schapire, Predicting nearly as well as the best pruning of a decision tree, Mach. Learn. 27 (1997) 51–68.
[15] E. Takimoto, A. Maruoka, V. Vovk, Predicting nearly as well as the best pruning of a decision tree through dynamic programming scheme, Theor. Comput. Sci. 261 (2001) 179–209. Eighth International Workshop on Algorithmic Learning Theory.
[16] E. Takimoto, M.K. Warmuth, Predicting nearly as well as the best pruning of a planar decision graph, Theor. Comput. Sci. 288 (2002) 217–235. Algorithmic Learning Theory.
[17] O. Bousquet, S. Boucheron, G. Lugosi, Introduction to Statistical Learning Theory, in: Advanced lectures on machine learning, Springer, 2004, pp. 169–207.
[18] J. Wang, V. Saligrama, Locally-linear learning machines (l3m), in: Asian Conference on Machine Learning, 2013, pp. 451–466.
[19] A.B. Lee, B. Nadler, L. Wasserman, Treeletsan adaptive multi-scale basis for sparse unordered data, Ann. Appl. Stat. 2 (2008) 435–471.
[20] J. Gama, Functional trees, Mach. Learn. 55 (2004) 219–250.
[21] J.a. Gama, R. Rocha, P. Medas, Accurate decision trees for mining high-speed data streams, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, in: KDD '03, ACM, New York, NY, USA, 2003, pp. 523–528.
[22] S.S. Kozat, A.C. Singer, G.C. Zeitler, Universal piecewise linear prediction via context trees, IEEE Trans. Signal Process. 55 (2007) 3730–3745.
[23] L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen, Classification and Regression Trees, Routledge, 2017.
[24] L. Rokach, O. Maimon, Top-down induction of decision trees classifiers-a survey, IEEE Trans. Syst. Man Cybern.Part C (Appl. Rev.) 35 (4) (2005) 476–487.
[25] P.E. Utgoff, Perceptron trees: a case study in hybrid concept representations, Conn. Sci. 1 (4) (1989) 377–391.
[26] R. Quinlan, Learning with continuous classes, in: Proceedings of the 5th Australian Joint Conference on Artificial Intelligence, 1992, pp. 343–348.

[27] E. Ikonomovska, Algorithms for learning regression trees and ensembles on evolving data streams, Jozef Stefan International Postgraduate School, Ljubljana, Slovenia, 2012 Doctoral Dissertation.
[28] E. Ikonomovska, J. Gama, S. Džeroski, Learning model trees from evolving data streams, Data Min. Knowl. Discovery 23 (2011) 128–168.
[29] E. Ikonomovska, J. Gama, R. Sebastião, D. Gjorgjevik, Regression Trees from Data Streams with Drift Detection, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 121–135.
[30] E. Ikonomovska, J. Gama, S. Džeroski, Learning model trees from evolving data streams, Data Min. Knowl. Discovery 23 (2011) 128–168.
[31] A.V. Aho, N.J. Sloane, Some doubly exponential sequences, Fibonacci Quart 11 (1973) 429–437.
[32] F.M.J. Willems, Y.M. Shtarkov, T.J. Tjalkens, The context-tree weighting method: basic properties, IEEE Trans. Inf. Theory 41 (1995) 653–664.
[33] A. Sayed, Fundamentals of Adaptive Filtering, Wiley - IEEE, Wiley, 2003.
[34] S.S. Kozat, A.T. Erdogan, A.C. Singer, A.H. Sayed, Steady-state mse performance analysis of mixture approaches to adaptive filtering, IEEE Trans. Signal Process. 58 (2010) 4050–4063.
[35] F.M.J. Willems, Coding for a binary independent piecewise-identically-distributed source, IEEE Trans. Inf. Theory 42 (1996) 2210–2217.
[36] A.C. Singer, S.S. Kozat, M. Feder, Universal linear least squares prediction: upper and lower bounds, IEEE Trans. Inf. Theory 48 (2002) 2354–2362.
[37] J.H. Friedman, Multivariate adaptive regression splines, Ann. Stat. 19 (1991) 1–67.
[38] J.H. Friedman, Fast Mars, Technical Report, Stanford University, 1993. URL http://www.milbo.users.sonic.net/earth/Friedman-FastMars.pdf.
[39] A. Carini, G.L. Sicuranza, Fourier nonlinear filters, Signal Process. 94 (2014) 183–194.
[40] C. Blake, C. Merz, Uci Repository of Machine Learning Databases, Information and Computer Science, University of California, Irvine, CA, 2015, p. 1998. URL http://www.archive.ics.uci.edu/ml.
[41] C.E. Rasmussen, R.M. Neal, G. Hinton, D. Camp, M. Revow, Z. Ghahramani, R. Kustra, R. Tibshirani, Delve data sets (2015). URL http://www.cs.toronto.edu/~delve/data/datasets.html
[42] G.K. Smyth, Australasian data and story library (ozdasl)(2015). URL http://www.statsci.org/data
[43] P. Vlachos, Statlib (2015). URL http://lib.stat.cmu.edu/datasets.
[44] E.N. Lorenz, Deterministic nonperiodic flow, J. Atmos. Sci. 20 (1963) 130–141.

N. Denizcan Vanli received the B.S. and M.S. degrees in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2013 and 2015, respectively. He is currently pursuing the Ph.D. degree in electrical engineering and computer science with the Massachusetts Institute of Technology, Cambridge, MA, USA. His current research interests include convex optimization, online learning, and distributed optimization.

Muhammed O. Sayin received the B.S. and M.S. degrees in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2013 and 2015, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Illinois at UrbanaâChampaign, Champaign, IL, USA. His current research interests include distributed networks, optimal control, and dynamic games.

Mohammadreza Mohaghegh N. received the B.S. degree in electrical engineering from Sharif University, Tehran, Iran in 2014. He is currently pursuing the M.S. degree in electrical and electronics engineering with Bilkent University, Ankara, Turkey. His current research interests include statistical signal processing, online learning, and optimization.

Huseyin Ozkan is currently a full-time faculty member of Electronics Engineering and with the Faculty of Engineering and Natural Sciences at Sabanc University. Dr. Ozkan received his B.Sc. degrees in Electrical Engineering and Mathematics from Bogazici University; and his M.Sc. and Ph.D. degrees in Electrical Engineering from Boston University and Bilkent University, respectively. Previously, he had been working as a postdoctoral research associate in Vision and Computational Neuroscience at Massachusetts Institute of Technology. His research interests are in machine learning, signal processing, computer vision and computational neuroscience.

Suleyman Serdar Kozat received the B.S. (Hons.) degree from Bilkent University, Ankara, Turkey, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Illinois at UrbanaâChampaign, Urbana, IL, USA. He joined the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, as a Research Staff Member and later became a Project Leader with the Pervasive Speech Technologies Group, where he focused on problems related to statistical signal processing and machine learning. He was a Research Associate with the Cryptography and Anti-Piracy Group, Microsoft Research, Redmond, WA, USA. He is currently an Associate Professor with the Electrical and Electronics Engineering Department, Bilkent University. He has co-authored over 120 papers in refereed high impact journals and conference proceedings and holds several patent inventions (currently used in several different Microsoft and IBM products, such as MSN and ViaVoice). He holds several patent inventions due to his research accomplishments with the IBM Thomas J. Watson Research Center and Microsoft Research. His current research interests include cyber security, anomaly detection, big data, data intelligence, adaptive filtering, and machine learning algorithms for signal processing. Dr. Kozat received many international and national awards. He is the Elected President of the IEEE Signal Processing Society, Turkey Chapter.