Contents lists available at ScienceDirect

Signal Processing

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

Highly efficient hierarchical online nonlinear regression using second order methods

Burak C. Civek[a,*], Ibrahim Delibalta[b], Suleyman S. Kozat[a]

[a] Department of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey
[b] Turk Telekom Communications Services Inc., Istanbul, Turkey

ARTICLE INFO

Article history:
Received 29 July 2016
Revised 21 January 2017
Accepted 25 January 2017
Available online 26 January 2017

Keywords:
Hierarchical tree
Nonlinear regression
Online learning
Piecewise linear regression
Newton method

ABSTRACT

We introduce highly efficient online nonlinear regression algorithms that are suitable for real life applications. We process the data in a truly online manner such that no storage is needed, i.e., the data is discarded after being used. For nonlinear modeling we use a hierarchical piecewise linear approach based on the notion of decision trees where the space of the regressor vectors is adaptively partitioned based on the performance. As the first time in the literature, we learn both the piecewise linear partitioning of the regressor space as well as the linear models in each region using highly effective second order methods, i.e., Newton–Raphson Methods. Hence, we avoid the well known over fitting issues by using piecewise linear models, however, since both the region boundaries as well as the linear models in each region are trained using the second order methods, we achieve substantial performance compared to the state of the art. We demonstrate our gains over the well known benchmark data sets and provide performance results in an individual sequence manner guaranteed to hold without any statistical assumptions. Hence, the introduced algorithms address computational complexity issues widely encountered in real life applications while providing superior guaranteed performance in a strong deterministic sense.

1. Introduction

Recent developments in information technologies, intelligent use of mobile devices and Internet have procured an extensive amount of data for the nonlinear modeling systems [1,2]. Today, many sources of information from shares on social networks to blogs, from intelligent device activities to large scale sensor networks are easily accessible [3]. Efficient and effective processing of this data can significantly improve the performance of many signal processing and machine learning algorithms [4–6]. In accordance with the aim of achieving more efficient algorithms, hierarchical approaches have been recently proposed for nonlinear modeling systems [7,8].

In this paper, we investigate the nonlinear regression problem that is one of the most important topics in the machine learning and signal processing literatures. This problem arises in several different applications such as signal modeling [9,10], financial market [11] and trend analyses [12], intrusion detection [13] and recommendation [14]. However, traditional regression techniques show less than adequate performance in real-life applications having big data since (1) data acquired from diverse sources are too large in size to be efficiently processed or stored by conventional signal processing and machine learning methods [15–18]; (2) the performance of the conventional methods is further impaired by the highly variable properties, structure and quality of data acquired at high speeds [15–17].

In this context, to accommodate these problems, we introduce online regression algorithms that process the data in an online manner, i.e., instantly, without any storage, and then discard the data after using and learning [18,19]. Hence our methods can constantly adapt to the changing statistics or quality of the data so that they can be robust and prone to variations and uncertainties [19–21]. From a unified point of view, in such problems, we sequentially observe a real valued sequence vector sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ and produce a decision (or an action) d_t at each time t based on the past $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t$. After the desired output d_t is revealed, we suffer a loss and our goal is to minimize the accumulated (and possibly weighted) loss as much as possible while using a limited amount of information from the past.

To this end, for nonlinear regression, we use a hierarchical piecewise linear model based on the notion of decision trees, where the space of the regressor vectors, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$, is adaptively partitioned and continuously optimized in order to enhance the

* Corresponding author.
E-mail addresses: civek@ee.bilkent.edu.tr (B.C. Civek), ibrahim.delibalta@turktelekom.com.tr (I. Delibalta), kozat@ee.bilkent.edu.tr (S.S. Kozat).

performance [10,22,23]. We note that the piecewise linear models are extensively used in the signal processing literature to mitigate the overtraining issues that arise because of using nonlinear models [10]. However their performance in real life applications are less than adequate since their successful application highly depends on the accurate selection of the piecewise regions that correctly model the underlying data [24]. Clearly, such a goal is impossible in an online setting since either the best partition is not known, i.e., the data arrives sequentially, or in real life applications the statistics of the data and the best selection of the regions change in time. To this end, as the first time in the literature, we learn both the piecewise linear partitioning of the regressor space as well as the linear models in each region using highly effective second order methods, i.e., Newton–Raphson Methods [25]. Hence, we avoid the well known over fitting issues by using piecewise linear models, moreover, since both the region boundaries as well as the linear models in each region are trained using the second order methods we achieve substantial performance compared to the state of the art [25]. We demonstrate our gains over the well known benchmark data sets extensively used in the machine learning literature. We also provide theoretical performance results in an individual sequence manner that are guaranteed to hold without any statistical assumptions [18]. In this sense, the introduced algorithms address computational complexity issues widely encountered in real life applications while providing superior guaranteed performance in a strong deterministic sense.

In adaptive signal processing literature, there exist methods which develop an approach based on weighted averaging of all possible models of a tree based partitioning instead of solely relying on a particular piecewise linear model [23,24]. These methods use the entire partitions of the regressor space and implement a full binary tree to form an online piecewise linear regressor. Such approaches are confirmed to lessen the bias variance trade off in a deterministic framework [23,24]. However, these methods do not update the corresponding partitioning of the regressor space based on the upcoming data. One such example is that the recursive dyadic partitioning, which partitions the regressor space using separation functions that are required to be parallel to the axes [26]. Moreover, these methods usually do not provide a theoretical justification for the weighting of the models, even if there exist inspirations from information theoretic deliberations [27]. For instance, there is an algorithmic concern on the definitions of both the exponentially weighted performance measure and the “universal weighting” coefficients [19,24,28,29] instead of a complete theoretical justifications (except the universal bounds). Specifically, these methods are constructed in such a way that there is a significant correlation between the weighting coefficients, algorithmic parameters and their performance, i.e., one should adjust these parameters to the specific application for successful process [24]. Besides these approaches, there exists an algorithm providing adaptive tree structure for the partitions, e.g., the Decision Adaptive Tree (DAT) [30]. The DAT produces the final estimate using the weighted average of the outcomes of all possible subtrees, which results in a computational complexity of $O(m4^d)$, where m is the data dimension and d represents the depth. However, this would affect the computational efficiency adversely for the cases involving highly nonlinear structures. In this work, we propose a different approach that avoids combining the prediction of each subtrees and offers a computational complexity of $O(m^2 2^d)$. Hence, we achieve an algorithm that is more efficient and effective for the cases involving higher nonlinearities, whereas the DAT is more feasible when the data dimension is quite high. Moreover, we illustrate in our experiments that our algorithm requires less number of data samples to capture the underlying data structure. Overall, the proposed methods are completely generic such that they are capable of incorporating all Recursive Dyadic, Random Projection (RP) and k-d trees in their framework, e.g., we initialize the partitioning process by using the RP trees and adaptively learn the complete structure of the tree based on the data progress to minimize the final error.

In Section 2, we first present the main framework for nonlinear regression and piecewise linear modeling. In Section 3, we propose three algorithms with regressor space partitioning and present guaranteed upper bounds on the performances. These algorithms adaptively learn the partitioning structure, region boundaries and region regressors to minimize the final regression error. We then demonstrate the performance of our algorithms through widely used benchmark data sets in Section 4. We then finalize our paper with concluding remarks.

2. Problem description

In this paper, all vectors are column vectors and represented by lower case boldface letters. For matrices, we use upper case boldface letters. The ℓ^2-norm of a vector $\boldsymbol{x}$ is given by $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^T\boldsymbol{x}}$ where $\boldsymbol{x}^T$ denotes the ordinary transpose. The identity matrix with $n \times n$ dimension is represented by $\boldsymbol{I}_n$.

We work in an online setting, where we estimate a data sequence $y_t \in \mathbb{R}$ at time $t \geq 1$ using the corresponding observed feature vector $\boldsymbol{x}_t \in \mathbb{R}^m$ and then discard $\boldsymbol{x}_t$ without any storage. Our goal is to sequentially estimate y_t using $\boldsymbol{x}_t$ as

$$\hat{y}_t = f_t(\boldsymbol{x}_t)$$

where $f_t(\cdot)$ is a function of past observations. In this work, we use nonlinear functions to model y_t, since in most real life applications, linear regressors are inadequate to successively model the intrinsic relation between the feature vector $\boldsymbol{x}_t$ and the desired data y_t [31]. Different from linear regressors, nonlinear functions are quite powerful and usually overfit in most real life cases [32]. To this end, we choose piecewise linear functions due to their capability of approximating most nonlinear models [33]. In order to construct a piecewise linear model, we partition the space of regressor vectors into K distinct m-dimensional regions S_k^m, where $\bigcup_{k=1}^K S_k^m = \mathbb{R}^m$ and $S_i^m \cap S_j^m = \emptyset$ when $i \neq j$. In each region, we use a linear regressor, i.e., $\hat{y}_{t,i} = \boldsymbol{w}_{t,i}^T \boldsymbol{x}_t + c_{t,i}$, where $\boldsymbol{w}_{t,\,i}$ is the linear regression vector, $c_{t,\,i}$ is the offset and $\hat{y}_{t,i}$ is the estimate corresponding to the ith region. We represent $\hat{y}_{t,i}$ in a more compact form as $\hat{y}_{t,i} = \boldsymbol{w}_{t,i}^T \boldsymbol{x}_t$, by including a bias term into each weight vector $\boldsymbol{w}_{t,\,i}$ and increasing the dimension of the space by 1, where the last entry of $\boldsymbol{x}_t$ is always set to 1.

To clarify the framework, in Fig. 1, we present a one dimensional regression problem, where we generate the data sequence using the nonlinear model

$$y_t = \exp(x_t \sin(4\pi x_t)) + \nu_t,$$

where x_t is a sample function from an i.i.d. standard uniform random process and ν_t has normal distribution with zero mean and 0.1 variance. Here, we demonstrate two different cases to emphasize the difficulties in piecewise linear modeling. For the case given in the upper plot, we partition the regression space into three regions and fit linear regressors to each partition. However, this construction does not approximate the given nonlinear model well enough since the underlying partition does not match exactly to the data. In order to better model the generated data, we use the second model as shown in the lower plot, where we have eight regions particularly selected according to the distribution of the data points. As the two cases signified in Fig. 1 imply, there are two major problems when using piecewise linear models. The first one is to determine the piecewise regions properly. Randomly selecting the partitions causes inadequately approximating models as indicated in the underfitting case on the top of Fig. 1 [22]. The second problem is to find out the linear model that best fits the data in

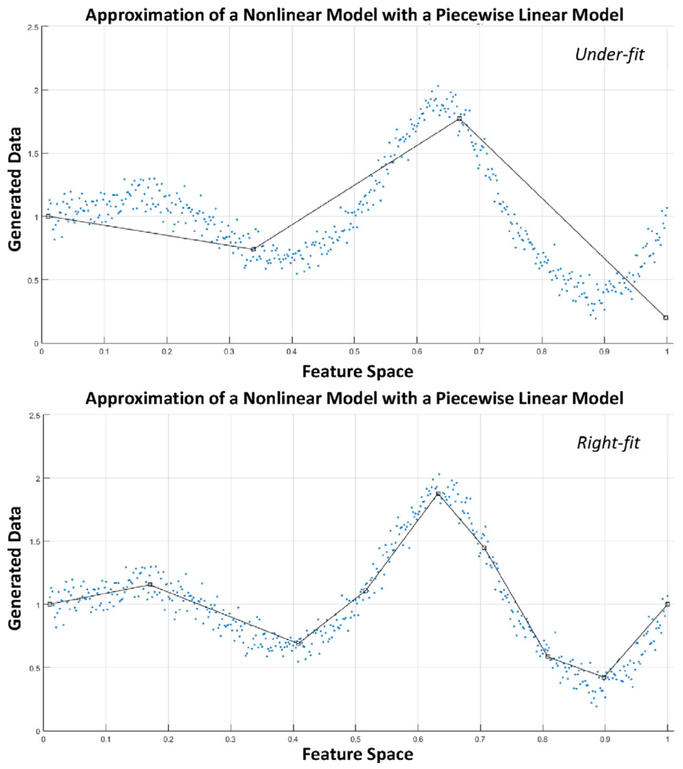

Fig. 1. In the upper plot, we represent an inadequate approximation of a piecewise linear model. In the lower plot, we represent a successive modeling with sufficiently partitioned regression space.

each distinct region in a sequential manner [24]. In this paper, we solve both of these problems using highly effective and completely adaptive second order piecewise linear regressors.

In order to have a measure on how well the determined piecewise linear model fits the data, we use instantaneous squared loss, i.e., $e_t^2 = (y_t - \hat{y}_t)^2$ as our cost function. Our goal is to specify the partitions and the corresponding linear regressors at each iteration such that the total regression error is minimized. Suppose $\boldsymbol{w}_n^*$ represents the optimal fixed weight for a particular region after n iteration, i.e.,

$$\boldsymbol{w}_n^* = \arg\min_{\boldsymbol{w}} \sum_{t=1}^{n} e_t^2(\boldsymbol{w}).$$

Hence, we would achieve the minimum possible regression error, if we have been considering $\boldsymbol{w}_n^*$ as the fixed linear regressor weight up to the current iteration, n. However, we do not process batch data sets, since the framework is online, and thus, cannot know the optimal weight beforehand [18]. This lack of information motivates us to implement an algorithm such that we achieve an error rate as close as the possible minimum after n iteration. At this point, we define the regret of an algorithm to measure how much the total error diverges from the possible minimum achieved by $\boldsymbol{w}_n^*$, i.e.,

$$\text{Regret}(A) = \sum_{t=1}^{n} e_t^2(\boldsymbol{w}_t) - \sum_{t=1}^{n} e_t^2(\boldsymbol{w}_n^*),$$

where A denotes the algorithm to adjust $\boldsymbol{w}_t$ at each iteration. Eventually, we consider the regret criterion to measure the modeling performance of the designated piecewise linear model and aim to attain a low regret [18].

In the following section, we propose three different algorithms to sufficiently model the intrinsic relation between the data sequence y_t and the linear regressor vectors. In each algorithm, we use piecewise linear models, where we partition the space of regressor vectors by using linear separation functions and assign a linear regressor to each partition. At this point, we also need to emphasize that we propose generic algorithms for nonlinear modeling. Even though we employ linear models in each partition, it is also possible to use, for example, spline modeling within the presented settings. This selection would cause additional update operations with minor changes for the higher order terms. Therefore, the proposed approaches can be implemented by using any other function that is differentiable without a significant difference in the algorithm, hence, they are universal in terms of the possible selection of functions. Overall, the presented algorithms ensure highly efficient and effective learning performance, since we perform second order update methods, e.g. Online Newton Step [34], for training of the region boundaries and the linear models.

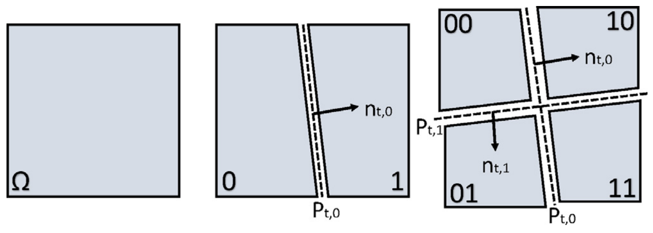

Fig. 2. Straight partitioning of the regression space.

3. Highly efficient tree based sequential piecewise linear predictors

In this section, we introduce three highly effective algorithms constructed by piecewise linear models. The presented algorithms provide efficient learning even for highly nonlinear data models. Moreover, continuous updating based on the upcoming data ensures our algorithms to achieve outstanding performance for online frameworks. Furthermore, we also provide a regret analysis for the introduced algorithms demonstrating strong guaranteed performance.

There exist two essential problems of piecewise linear modeling. The first significant issue is to determine how to partition the regressor space. We carry out the partitioning process using linear separation functions. We specify the separation functions as hyperplanes, which are $(m-1)$-dimensional subspaces of m-dimensional regression space and identified by their normal vectors as shown in Fig. 2. To get a highly versatile and data adaptive partitioning, we also train the region boundaries by updating corresponding normal vectors. We denote the separation functions as $p_{t,\,k}$ and the normal vectors as $\boldsymbol{n}_{t,\,k}$ where k is the region label as we demonstrate in Fig. 2. In order to adaptively train the region boundaries, we use differentiable functions as the separation functions instead of hard separation boundaries as seen in Fig. 3, i.e.,

$$p_{t,k} = \frac{1}{1 + e^{-\boldsymbol{x}_t^T \boldsymbol{n}_{t,k}}} \qquad (1)$$

where the offset $c_{t,\,k}$ is included in the norm vector $\boldsymbol{n}_{t,\,k}$ as a bias term. In Fig. 3, logistic regression functions for 1-dimensional case are shown for different parameters. Following the partitioning process, the second essential problem is to find out the linear models in each region. We assign a linear regressor specific to each distinct region and generate a corresponding estimate $\hat{y}_{t,r}$, given by

$$\hat{y}_{t,r} = \boldsymbol{w}_{t,r}^T \boldsymbol{x}_t \qquad (2)$$

where $\boldsymbol{w}_{t,r}$ is the regression vector particular to region r. In the following subsections, we present different methods to partition the regressor space to construct our algorithms.

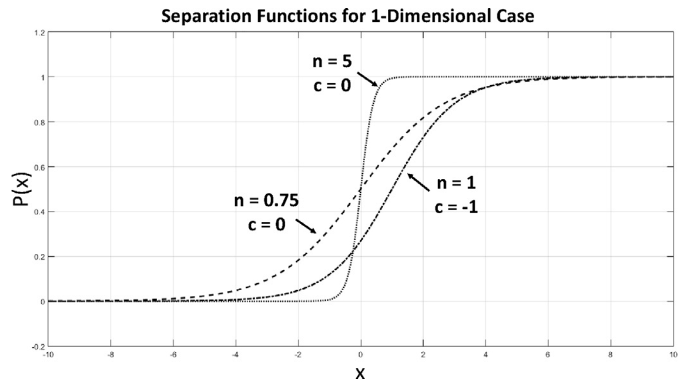

Fig. 3. Separation Functions for 1-Dimensional Case where $\{n = 5, c = 0\}$, $\{n = 0.75, c = 0\}$ and $\{n = 1, c = -1\}$. Parameter n specifies the sharpness, as c determines the position or the offset on the x-axis.

3.1. Partitioning methods

We introduce two different partitioning methods: *Type 1*, which is a straightforward partitioning and *Type 2*, which is an efficient tree structured partitioning.

3.1.1. Type 1 partitioning

In this method, we allow each hyperplane to divide the whole space into two subspaces as shown in Fig. 2. In order to clarify the technique, we work on the 2-dimensional space, i.e., the coordinate plane. Suppose, the observed feature vectors $\boldsymbol{x}_t = [x_{t,1}, x_{t,2}]^T$ come from a bounded set $\{\Omega\}$ such that $-A \leq x_{t,1}, x_{t,2} \leq A$ for some $A > 0$, as shown in Fig. 2. We define 1-dimensional hyperplanes, whose normal vector representation is given by $\boldsymbol{n}_{t,k} \in \mathbb{R}^2$ where k denotes the corresponding region identity. At first, we have the whole space as a single set $\{\Omega\}$. Then we use a single separation function, which is a line in this case, to partition this space into subspaces $\{0\}$ and $\{1\}$ such that $\{0\} \cup \{1\} = \{\Omega\}$. When we add another hyperplane separating the set Ω, we get four distinct subspaces $\{00\}$, $\{01\}$, $\{10\}$ and $\{11\}$ where their union forms the initial regression space. The number of separated regions increases by $O(k^2)$. Note that if we use k different separation functions, then we can obtain up to $\frac{k^2+k+2}{2}$ distinct regions forming a complete space.

3.1.2. Type 2 partitioning

In the second method, we use the tree notion to partition the regression space, which is a more systematic way to determine the regions [10,22]. We illustrate this method in Fig. 4 for 2-dimensional case. First step is the same as previously mentioned approach, i.e., we partition the whole regression space into two distinct regions using one separation function. In the following steps, the partition technique is quite different. Since we have two distinct subspaces after the first step, we work on them separately, i.e., the partition process continues recursively in each subspace independent of the others. Therefore, adding one more hyperplane has an effect on just a single region, not on the whole space. The number of distinct regions in total increases by 1, when we apply one more separation function. Thus, in order to represent $p+1$ distinct regions, we specify p separation functions. For the tree case, we use another identifier called the depth, which determines how deep the partition is, e.g. depth of the model shown in Fig. 4 is 2. In particular, the number of different regions generated by the depth-d models are given by 2^d. Hence, the number of distinct regions increases in the order of $O(2^d)$. For the tree based partitioning, we use the finest model of a depth-d tree. The finest partition consists of the regions that are generated at the deepest level, e.g. regions $\{00\}$, $\{01\}$, $\{10\}$ and $\{11\}$ as shown in Fig. 4.

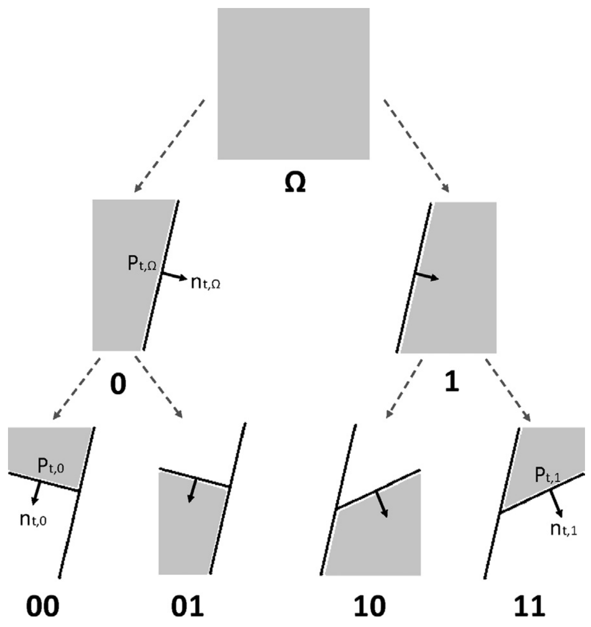

Fig. 4. Tree based partitioning of the regression space.

Both Type 1 and Type 2 partitioning have their own advantages, i.e., Type 2 partitioning achieves a better steady state error performance since the models generated by Type 1 partitioning are the subclasses of Type 2, however, Type 1 might perform better in the transient region since it uses less parameters.

3.2. Algorithm for Type 1 partitioning

In this part, we introduce our first algorithm, which is based on the *Type 1* partitioning. Following the model given in Fig. 2, say, we have two different separator functions, $p_{t,0}, p_{t,1} \in \mathbb{R}$, which are defined by $\boldsymbol{n}_{t,0}, \boldsymbol{n}_{t,1} \in \mathbb{R}^2$ respectively. For the region $\{00\}$, the corresponding estimate is given by

$$\hat{y}_{t,00} = \boldsymbol{w}_{t,00}^T \boldsymbol{x}_t,$$

where $\boldsymbol{w}_{t,00} \in \mathbb{R}^2$ is the regression vector of the region $\{00\}$. Since we have the estimates of all regions, the final estimate is given by

$$\begin{aligned}\hat{y}_t &= p_{t,0} p_{t,1} \hat{y}_{t,00} + p_{t,0}(1 - p_{t,1}) \hat{y}_{t,01} \\ &\quad + (1 - p_{t,0}) p_{t,1} \hat{y}_{t,10} + (1 - p_{t,0})(1 - p_{t,1}) \hat{y}_{t,11} \end{aligned} \tag{3}$$

when we observe the feature vector $\boldsymbol{x}_t$. This result can be easily extended to the cases where we have more than 2 separator functions.

We adaptively update the weights associated with each partition based on the overall performance. Boundaries of the regions are also updated to reach the best partitioning. We use the second order algorithms, e.g. Online Newton Step [34], to update both separator functions and region weights. To accomplish this, the weight vector assigned to the region $\{00\}$ is updated as

$$\begin{aligned}\boldsymbol{w}_{t+1,00} &= \boldsymbol{w}_{t,00} - \frac{1}{\beta} \boldsymbol{A}_t^{-1} \nabla e_t^2 \\ &= \boldsymbol{w}_{t,00} + \frac{2}{\beta} e_t p_{t,0} p_{t,1} \boldsymbol{A}_t^{-1} \boldsymbol{x}_t, \end{aligned} \tag{4}$$

where β is the step size, ∇ is the gradient operator w.r.t. $\boldsymbol{w}_{t,\,00}$ and $\boldsymbol{A}_t$ is an $m \times m$ matrix defined as

$$\boldsymbol{A}_t = \sum_{i=1}^{t} \nabla_i \nabla_i^T + \epsilon \boldsymbol{I}_m, \tag{5}$$

where $\nabla_t \triangleq \nabla e_t^2$ and $\epsilon > 0$ is used to ensure that $\boldsymbol{A}_t$ is positive definite, i.e., $\boldsymbol{A}_t > 0$, and invertible. Here, the matrix $\boldsymbol{A}_t$ is related to the Hessian of the error function, implying that the update rule uses the second order information [34].

Region boundaries are also updated in the same manner. For example, the direction vector specifying the separation function $p_{t,0}$ in Fig. 2, is updated as

$$\begin{aligned}\boldsymbol{n}_{t+1,0} &= \boldsymbol{n}_{t,0} - \frac{1}{\eta}\boldsymbol{A}_t^{-1}\nabla e_t^2 \\ &= \boldsymbol{n}_{t,0} + \frac{2}{\eta}e_t[p_{t,1}\hat{y}_{t,00} + (1-p_{t,1})\hat{y}_{t,01} \\ &\quad - p_{t,1}\hat{y}_{t,10} - (1-p_{t,1})\hat{y}_{t,11}]\boldsymbol{A}_t^{-1}\frac{\partial p_{t,0}}{\partial \boldsymbol{n}_{t,0}},\end{aligned} \tag{6}$$

where η is the step size to be determined, ∇ is the gradient operator w.r.t. $\boldsymbol{n}_{t,0}$ and $\boldsymbol{A}_t$ is given in (5). Partial derivative of the separation function $p_{t,0}$ w.r.t. $\boldsymbol{n}_{t,0}$ is given by

$$\frac{\partial p_{t,0}}{\partial \boldsymbol{n}_{t,0}} = \frac{\boldsymbol{x}_t e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,0}}}{(1+e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,0}})^2}. \tag{7}$$

All separation functions are updated in the same manner. In general, we derive the final estimate in a compact form as

$$\hat{y}_t = \sum_{r\in R}\hat{\psi}_{t,r}, \tag{8}$$

where $\hat{\psi}_{t,r}$ is the weighted estimate of region r and R represents the set of all region labels, e.g. $R = \{00, 01, 10, 11\}$ for the case given in Fig. 2. Weighted estimate of each region is determined by

$$\hat{\psi}_{t,r} = \hat{y}_{t,r}\prod_{i=1}^{K}\hat{p}_{t,P(i)}, \tag{9}$$

where K is the number of separation functions, P represents the set of all separation function labels and $P(i)$ is the ith element of set P, e.g. $P = \{0, 1\}, P(1) = 0$, and $\hat{p}_{t,P(i)}$ is defined as

$$\hat{p}_{t,P(i)} = \begin{cases} p_{t,P(i)}, & r(i) = 0 \\ 1 - p_{t,P(i)}, & r(i) = 1\end{cases}, \tag{10}$$

where $r(i)$ denotes the ith binary character of label r, e.g. $r = 10$ and $r(1) = 1$. We reformulate the update rules defined in (4) and (6) and present generic expressions for both regression weights and region boundaries. The derivations of the generic update rules are calculated after some basic algebra. Hence, the regression weights are updated as

$$\boldsymbol{w}_{t+1,r} = \boldsymbol{w}_{t,r} + \frac{2}{\beta}e_t\boldsymbol{A}_t^{-1}\boldsymbol{x}_t\prod_{i=1}^{K}\hat{p}_{t,P(i)} \tag{11}$$

and the region boundaries are updated as

$$\boldsymbol{n}_{t+1,k} = \boldsymbol{n}_{t,k} + \frac{2}{\eta}e_t\boldsymbol{A}_t^{-1}\left[\sum_{r\in R}\hat{y}_{t,r}(-1)^{r(i)}\prod_{\substack{j=1\\ j\neq i}}^{K}\hat{p}_{t,P(j)}\right]\frac{\boldsymbol{x}_t e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,k}}}{(1+e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,k}})^2}, \tag{12}$$

where we assign $k = P(i)$, i.e., separation function with label-k is the i^{th} entry of set P. Partial derivative of the logistic regression function $p_{t,k}$ w.r.t. $\boldsymbol{n}_{t,k}$ is also inserted in (12). In order to avoid taking the inverse of an $m \times m$ matrix, $\boldsymbol{A}_t$, at each iteration in (11) and (12), we generate a recursive formula using matrix inversion lemma for $\boldsymbol{A}_t^{-1}$ given as [4]

$$\boldsymbol{A}_t^{-1} = \boldsymbol{A}_{t-1}^{-1} - \frac{\boldsymbol{A}_{t-1}^{-1}\nabla_t\nabla_t^T\boldsymbol{A}_{t-1}^{-1}}{1+\nabla_t^T\boldsymbol{A}_{t-1}^{-1}\nabla_t}, \tag{13}$$

Algorithm 1 Straight partitioning.

```
1:  $A_0^{-1} = \frac{1}{\epsilon}I_m$
2:  for $t \leftarrow 1, n$ do
3:      $\hat{y}_t \leftarrow 0$
4:      for all $r \in R$ do
5:          $\hat{y}_{t,r} \leftarrow w_{t,r}^T x_t$
6:          $\hat{\psi}_{t,r} \leftarrow \hat{y}_{t,r}$
7:          $\nabla_{t,r} \leftarrow x_t$
8:          for $i \leftarrow 1, K$ do
9:              if $r(i) := 0$ then
10:                 $\hat{p}_{t,P(i)} \leftarrow p_{t,P(i)}$
11:             else
12:                 $\hat{p}_{t,P(i)} \leftarrow 1 - p_{t,P(i)}$
13:             end if
14:             $\hat{\psi}_{t,r} \leftarrow \hat{\psi}_{t,r}\hat{p}_{t,P(i)}$
15:             $\nabla_{t,r} \leftarrow \nabla_{t,r}\hat{p}_{t,P(i)}$
16:         end for
17:         for $i \leftarrow 1, K$ do
18:             $\alpha_{t,P(i)} \leftarrow (-1)^{r(i)}(\hat{\psi}_{t,r}/\hat{p}_{t,P(i)})$
19:         end for
20:         $\hat{y}_t \leftarrow \hat{y}_t + \hat{\psi}_{t,r}$
21:     end for
22:     $e_t \leftarrow y_t - \hat{y}_t$
23:     for all $r \in R$ do
24:         $\nabla_{t,r} \leftarrow -2e_t\nabla_{t,r}$
25:         $A_{t,r}^{-1} \leftarrow A_{t-1,r}^{-1} - \frac{A_{t-1,r}^{-1}\nabla_{t,r}\nabla_{t,r}^T A_{t-1,r}^{-1}}{1+\nabla_{t,r}^T A_{t-1,r}^{-1}\nabla_{t,r}}$
26:         $w_{t+1,r} \leftarrow w_{t,r} - \frac{1}{\beta}A_{t,r}^{-1}\nabla_{t,r}$
27:     end for
28:     for $i \leftarrow 1, K$ do
29:         $k \leftarrow P(i)$
30:         $\nabla_{t,k} \leftarrow -2e_t\alpha_{t,k}p_{t,k}(1-p_{t,k})x_t$
31:         $A_{t,k}^{-1} \leftarrow A_{t-1,k}^{-1} - \frac{A_{t-1,k}^{-1}\nabla_{t,k}\nabla_{t,k}^T A_{t-1,k}^{-1}}{1+\nabla_{t,k}^T A_{t-1,k}^{-1}\nabla_{t,k}}$
32:         $n_{t+1,k} \leftarrow n_{t,k} - \frac{1}{\eta}A_{t,k}^{-1}\nabla_{t,k}$
33:     end for
34: end for
```

where $\nabla_t \triangleq \nabla e_t^2$ w.r.t. the corresponding variable. The complete algorithm for *Type 1* partitioning is given in Algorithm 1 with all updates and initializations.

3.3. Algorithm for Type 2 partitioning

In this algorithm, we use another approach to estimate the desired data. The partition of the regressor space will be based on the finest model of a tree structure [10,23]. We follow the case given in Fig. 4. Here, we have three separation functions, $p_{t,\epsilon}$, $p_{t,0}$ and $p_{t,1}$, partitioning the whole space into four subspaces. The corresponding direction vectors are given by $\boldsymbol{n}_{t,\epsilon}$, $\boldsymbol{n}_{t,0}$ and $\boldsymbol{n}_{t,1}$ respectively. Using the individual estimates of all four regions, we find the final estimate by

$$\begin{aligned}\hat{y}_t &= p_{t,\epsilon}p_{t,0}\hat{y}_{t,00} + p_{t,\epsilon}(1-p_{t,0})\hat{y}_{t,01} \\ &\quad + (1-p_{t,\epsilon})p_{t,1}\hat{y}_{t,10} + (1-p_{t,\epsilon})(1-p_{t,1})\hat{y}_{t,11}\end{aligned} \tag{14}$$

which can be extended to depth-d models with $d > 2$.

Regressors of each region is updated similar to the first algorithm. We demonstrate a systematic way of labeling for partitions in Fig. 5. The final estimate of this algorithm is given by the following generic formula

$$\hat{y}_t = \sum_{j=1}^{2^d}\hat{\psi}_{t,R_d(j)} \tag{15}$$

where R_d is the set of all region labels with length d in the increasing order for, i.e., $R_1 = \{0, 1\}$ or $R_2 = \{00, 01, 10, 11\}$ and $R_d(j)$ represents the jth entry of set R_d. Weighted estimate of each region is found as

$$\hat{\psi}_{t,r} = \hat{y}_{t,r}\prod_{i=1}^{d}\hat{p}_{t,r_i} \tag{16}$$

where r_i denotes the first $i-1$ character of label r as a string, i.e., $r = \{0101\}, r_3 = \{01\}$ and $r_1 = \{\epsilon\}$, which is the empty string $\{\epsilon\}$.

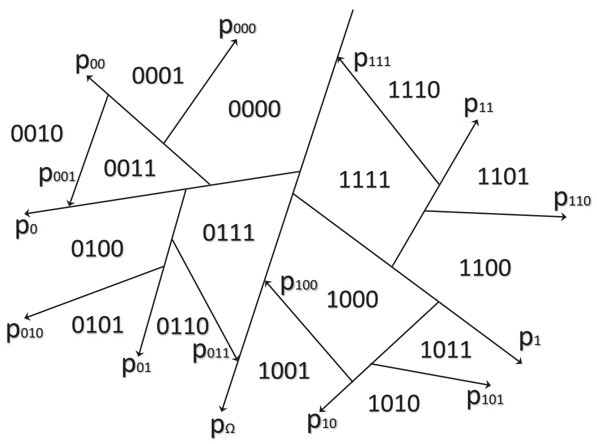

Fig. 5. Labeling example for the depth-4 case of the finest model

Here, $\hat{p}_{t,r_i}$ is defined as

$$\hat{p}_{t,r_i} = \begin{cases} p_{t,r_i}, & r(i) = 0 \\ 1 - p_{t,r_i}, & r(i) = 1 \end{cases}. \tag{17}$$

Update rules for the region weights and the boundaries are given as a generic form and the derivations of these updates are obtained after some basic algebra. Regressor vectors are updated as

$$\boldsymbol{w}_{t+1,r} = \boldsymbol{w}_{t,r} + \frac{2}{\beta} e_t \boldsymbol{A}_t \boldsymbol{x}_t \prod_{i=1}^{d} \hat{p}_{t,r_i} \tag{18}$$

and the separator function updates are given by

$$\boldsymbol{n}_{t+1,k} = \boldsymbol{n}_{t,k} + \frac{2}{\eta} e_t \boldsymbol{A}_t^{-1} \left[\sum_{j=1}^{2^{d-\ell(k)}} \hat{y}_{t,r} (-1)^{r(\ell(k)+1)} \prod_{\substack{i=1 \\ r_i \neq k}}^{d} \hat{p}_{t,r_i} \right] \frac{\partial p_{t,k}}{\partial \boldsymbol{n}_{t,k}} \tag{19}$$

where r is the label string generated by concatenating separation function id k and the label kept in jth entry of the set $R_{(d-\ell(k))}$, i.e., $r = [k; R_{(d-\ell(k))}(j)]$ and $\ell(k)$ represents the length of binary string k, e.g. $\ell(01) = 2$. The partial derivative of $p_{t,\,k}$ w.r.t. $\boldsymbol{n}_{t,\,k}$ is the same expression given in (14). The complete algorithm for *Type 2* partitioning is given in Algorithm 2 with all updates and initializations.

3.4. Algorithm for combining all possible models of tree

In this algorithm, we combine the estimates generated by all possible models of a tree based partition, instead of considering only the finest model. The main goal of this algorithm is to illustrate that using only the finest model of a depth-d tree provides a better performance. For example, we represent the possible models corresponding to a depth-2 tree in Fig. 6. We emphasize that the last partition is the finest model we use in the previous algorithm. Following the case in Fig. 6, we generate five distinct piecewise linear models and estimates of these models. The final estimate is then constructed by linearly combining the outputs of each piecewise linear model, represented by $\hat{\phi}_{t,\lambda}$, where λ represents the model identity. Hence, $\hat{y}_t$ is given by

$$\hat{y}_t = \boldsymbol{v}_t^T \hat{\boldsymbol{\phi}}_t \tag{20}$$

where $\hat{\boldsymbol{\phi}}_t = [\hat{\phi}_{t,1}, \hat{\phi}_{t,2}, \ldots, \hat{\phi}_{t,M}]^T$, $\boldsymbol{v}_t \in \mathbb{R}^M$ is the weight vector and M represents the number of possible distinct models generated by a depth-d tree, e.g. $M = 5$ for depth-2 case. In general, we have $M \approx (1.5)^{2^d}$. Model estimates, $\hat{\phi}_{t,\lambda}$, are calculated in the same way as in Section 3.3. Linear combination weights, $\boldsymbol{v}_t$, are also adaptively updated using the second order methods as performed in the previous sections.

Algorithm 2 Finest model partitioning.

```
1: $A_0^{-1} \leftarrow \frac{1}{\epsilon} \boldsymbol{I}_m$
2: for $t \leftarrow 1, n$ do
3:     $\hat{y}_t \leftarrow 0$
4:     for $j \leftarrow 1, 2^d$ do
5:         $r \leftarrow R_d(j)$
6:         $\hat{y}_{t,r} \leftarrow \boldsymbol{w}_{t,r}^T \boldsymbol{x}_t$
7:         $\hat{\psi}_{t,r} \leftarrow \hat{y}_{t,r}$
8:         $\gamma_{t,r} \leftarrow 1$
9:         for $i \leftarrow 1, d$ do
10:            if $r(i) \leftarrow 0$ then
11:                $\hat{p}_{t,r_i} \leftarrow p_{t,r_i}$
12:            else
13:                $\hat{p}_{t,r_i} \leftarrow 1 - p_{t,r_i}$
14:            end if
15:            $\hat{\psi}_{t,r} \leftarrow \hat{\psi}_{t,r} \hat{p}_{t,r_i}$
16:            $\gamma_{t,r} \leftarrow \gamma_{t,r} \hat{p}_{t,r_i}$
17:        end for
18:        $\hat{y}_t \leftarrow \hat{y}_t + \hat{\psi}_{t,r}$
19:    end for
20:    for $i \leftarrow 1, 2^d - 1$ do
21:        $k \leftarrow P(i)$
22:        for $j \leftarrow 1, 2^{d-\ell(k)}$ do
23:            $r \leftarrow concat[k : R_{d-\ell(k)}(j)]$
24:            $\alpha_{t,k} \leftarrow (-1)^{r(\ell(k)+1)} (\hat{\psi}_{t,r} / \hat{p}_{t,k})$
25:        end for
26:    end for
27:    $e_t \leftarrow y_t - \hat{y}_t$
28:    for $j \leftarrow 1, 2^d$ do
29:        $r \leftarrow R_d(j)$
30:        $\nabla_{t,r} \leftarrow -2 e_t \gamma_{t,r} \boldsymbol{x}_t$
31:        $\boldsymbol{A}_{t,r}^{-1} \leftarrow \boldsymbol{A}_{t-1,r}^{-1} - \frac{\boldsymbol{A}_{t-1}^{-1} \nabla_{t,r} \nabla_{t,r}^T \boldsymbol{A}_{t-1,r}^{-1}}{1 + \nabla_{t,r}^T \boldsymbol{A}_{t-1,r}^{-1} \nabla_{t,r}}$
32:        $\boldsymbol{w}_{t+1,r} \leftarrow \boldsymbol{w}_{t,r} - \frac{1}{\beta} \boldsymbol{A}_{t,r}^{-1} \nabla_{t,r}$
33:    end for
34:    for $i \leftarrow 1, 2^d - 1$ do
35:        $k \leftarrow P(i)$
36:        $\nabla_{t,k} \leftarrow -2 e_t \alpha_{t,k} p_{t,k} (1 - p_{t,k}) \boldsymbol{x}_t$
37:        $\boldsymbol{A}_{t,k}^{-1} \leftarrow \boldsymbol{A}_{t-1,k}^{-1} - \frac{\boldsymbol{A}_{t-1,k}^{-1} \nabla_{t,k} \nabla_{t,k}^T \boldsymbol{A}_{t-1,k}^{-1}}{1 + \nabla_{t,k}^T \boldsymbol{A}_{t-1,k}^{-1} \nabla_{t,k}}$
38:        $\boldsymbol{n}_{t+1,k} \leftarrow \boldsymbol{n}_{t,k} - \frac{1}{\eta} \boldsymbol{A}_{t,k}^{-1} \nabla_{t,k}$
39:    end for
40: end for
```

Table 1
Computational complexities.

Algorithms	FMP	SP	S-DAT	DFT	DAT
Complexity	$O(m^2 2^d)$	$O(m^2 k^2)$	$O(m^2 4^d)$	$O(md2^d)$	$O(m4^d)$
Algorithms	GKR	CTW	FNF	EMFNF	VF
Complexity	$O(m2^d)$	$O(md)$	$O(m^n n^n)$	$O(m^n)$	$O(m^n)$

3.5. Computational complexities

In this section, we determine the computational complexities of the proposed algorithms. In the algorithm for *Type 1* partitioning, the regressor space is partitioned into at most $\frac{k^2+k+2}{2}$ regions by using k distinct separator function. Thus, this algorithm requires $O(k^2)$ weight update at each iteration. In the algorithm for *Type 2* partitioning, the regressor space is partitioned into 2^d regions for the depth-d tree model. Hence, we perform $O(2^d)$ weight update at each iteration. The last algorithm combines all possible models of depth-d tree and calculates the final estimate in an efficient way requiring $O(4^d)$ weight updates [30]. Suppose that the regressor space is m-dimensional, i.e., $\boldsymbol{x}_t \in \mathbb{R}^m$. For each update, all three algorithms require $O(m^2)$ multiplication and addition resulting from a matrix-vector product, since we apply second order update methods. Therefore, the corresponding complexities are $O(m^2k^2)$, $O(m^2 2^d)$ and $O(m^2 4^d)$ for the Algorithm 1, the Algorithm 2 and the Algorithm 3 respectively. In Table 1, we represent the computational complexities of the existing algorithms. "FMP" and "SP" represents Finest Model Partitioning and Straight Partitioning algorithms respectively. "DFT" stands for Decision Fixed Tree and "DAT" represents Decision Adaptive Tree [30]. "S-DAT" denotes the Decision Adaptive Tree with second order update rules. "CTW" is used for Context Tree Weighting [24], "GKR" represents Gaussian-Kernel regressor [35], "VF" represents Volterra Filter [36], "FNF" and "EMFNF" stand for the Fourier and Even Mirror Fourier Nonlinear Filter [37] respectively.

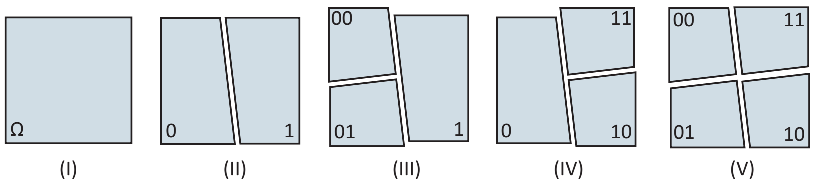

Fig. 6. All possible models for the depth-2 tree based partitioning.

3.6. Logarithmic regret bound

In this subsection, we provide regret results for the introduced algorithms. All three algorithms uses the second order update rule, Online Newton Step [34], and achieves a logarithmic regret when the normal vectors of the region boundaries are fixed and the cost function is convex in the sense of individual region weights. In order to construct the upper bounds, we first let $\boldsymbol{w}_n^*$ be the best predictor in hindsight, i.e.,

$$\boldsymbol{w}_n^* = \arg\min_{\boldsymbol{w}} \sum_{t=1}^{n} e_t^2(\boldsymbol{w}) \tag{21}$$

and express the following inequality

$$e_t^2(\boldsymbol{w}_t) - e_t^2(\boldsymbol{w}_n^*) \le \nabla_t^T(\boldsymbol{w}_t - \boldsymbol{w}_n^*) - \frac{\beta}{2}(\boldsymbol{w}_t - \boldsymbol{w}_n^*)^T \nabla_t \nabla_t^T (\boldsymbol{w}_t - \boldsymbol{w}_n^*) \tag{22}$$

using the Lemma 3 of [34], since our cost function is α-exp-concave, i.e., $\exp(-\alpha e_t^2(\boldsymbol{w}_t))$ is concave for $\alpha > 0$ and has an upper bound G on its gradient, i.e., $\|\nabla_t\| \le G$. We give the update rule for regressor weights as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{1}{\beta}\boldsymbol{A}_t^{-1}\nabla_t. \tag{23}$$

When we subtract the optimal weight from both sides, we get

$$\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^* = \boldsymbol{w}_t - \boldsymbol{w}_n^* - \frac{1}{\beta}\boldsymbol{A}_t^{-1}\nabla_t \tag{24}$$

$$\boldsymbol{A}_t(\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*) = \boldsymbol{A}_t(\boldsymbol{w}_t - \boldsymbol{w}_n^*) - \frac{1}{\beta}\nabla_t \tag{25}$$

and multiply second equation with the transpose of the first equation to get

$$\nabla_t(\boldsymbol{w}_t - \boldsymbol{w}_n^*) = \frac{1}{2\beta}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t + \frac{\beta}{2}(\boldsymbol{w}_t - \boldsymbol{w}_n^*)^T\boldsymbol{A}_t(\boldsymbol{w}_t - \boldsymbol{w}_n^*) - \frac{\beta}{2}(\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*)^T\boldsymbol{A}_t(\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*). \tag{26}$$

By following a similar discussion [34], except that we have equality in (26) and in the proceeding parts, we achieve the inequality

$$\sum_{t=1}^{n} S_t \le \frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t + \frac{\beta}{2}(\boldsymbol{w}_1 - \boldsymbol{w}_n^*)^T\boldsymbol{A}_0(\boldsymbol{w}_1 - \boldsymbol{w}_n^*), \tag{27}$$

where S_t is defined as

$$S_t \triangleq \nabla_t^T(\boldsymbol{w}_t - \boldsymbol{w}_n^*) - \frac{\beta}{2}(\boldsymbol{w}_t - \boldsymbol{w}_n^*)^T\nabla_t\nabla_t^T(\boldsymbol{w}_t - \boldsymbol{w}_n^*). \tag{28}$$

Since we define $\boldsymbol{A}_0 = \epsilon\boldsymbol{I}_m$ and have a finite space of regression vectors, i.e., $\|\boldsymbol{w}_t - \boldsymbol{w}_n^*\|^2 \le A^2$, we get

$$\sum_{t=1}^{n} e_t^2(\boldsymbol{w}_t) - \sum_{t=1}^{n} e_t^2(\boldsymbol{w}_n^*) \le \frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t + \frac{\beta}{2}\epsilon\delta^2 \le \frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t + \frac{1}{2\beta}, \tag{29}$$

where we choose $\epsilon = \frac{1}{\beta^2 A^2}$ and use the inequalities (10) and (17). Now, we specify an upper bound for the first term in LHS of the inequality (19). We make use of Lemma 11 given in [34], to get the following bound

$$\frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t \le \frac{m}{2\beta}\log\left(\frac{G^2 n}{\epsilon} + 1\right) = \frac{m}{2\beta}\log(G^2 n\beta^2 A^2 + 1) \le \frac{m}{2\beta}\log(n), \tag{30}$$

where in the last inequality, we use the choice of β, i.e., $\beta = \frac{1}{2}\min\{\frac{1}{4GA}, \alpha\}$, which implies that $\frac{1}{\beta} \le 8(GA + \frac{1}{\alpha})$. Therefore, we present the final logarithmic regret bound as

$$\sum_{t=1}^{n} e_t^2(\boldsymbol{w}_t) - \sum_{t=1}^{n} e_t^2(\boldsymbol{w}_n^*) \le 5\left(GA + \frac{1}{\alpha}\right)m\log(n). \tag{31}$$

4. Simulations

In this section, we evaluate the performance of the proposed algorithms under different scenarios. In the first set of simulations, we aim to provide a better understanding of our algorithms. To this end, we first consider the regression of a signal that is generated by a piecewise linear model whose partitions match the initial partitioning of our algorithms. Then we examine the case of mismatched initial partitions to illustrate the learning process of the presented algorithms. As the second set of simulation, we mainly assess the merits of our algorithms by using the well known real and synthetic benchmark datasets that are extensively used in the signal processing and the machine learning literatures, e.g., California Housing [38], Kinematics [38] and Elevators [38]. We then perform two more experiments with two chaotic processes, e.g., the Gauss map and the Lorenz attractor, to demonstrate the merits of our algorithms. All data sequences used in the simulations are scaled to the range $[-1, 1]$ and the learning rates are selected to obtain the best steady state performance of each algorithm.

4.1. Matched partition

In this subsection, we consider the regression of a signal generated using a piecewise linear model whose partitions match with the initial partitioning of the proposed algorithms. The main goal of this experiment is to provide an insight on the working principles of the proposed algorithms. Hence, this experiment is not designated to assess the performance of our algorithms with respect to the ones that are not based on piecewise linear modeling. This is only an illustration of how it is possible to achieve a performance gain when the data sequence is generated by a nonlinear system.

We use the following piecewise linear model to generate the data sequence,

$$\hat{y}_t = \begin{cases} \boldsymbol{w}_1^T\boldsymbol{x}_t + \upsilon_t, & \boldsymbol{x}_t^T\boldsymbol{n}_0 \ge 0 \text{ and } \boldsymbol{x}_t^T\boldsymbol{n}_1 \ge 0 \\ \boldsymbol{w}_2^T\boldsymbol{x}_t + \upsilon_t, & \boldsymbol{x}_t^T\boldsymbol{n}_0 \ge 0 \text{ and } \boldsymbol{x}_t^T\boldsymbol{n}_1 < 0 \\ \boldsymbol{w}_2^T\boldsymbol{x}_t + \upsilon_t, & \boldsymbol{x}_t^T\boldsymbol{n}_0 < 0 \text{ and } \boldsymbol{x}_t^T\boldsymbol{n}_1 \ge 0 \\ \boldsymbol{w}_1^T\boldsymbol{x}_t + \upsilon_t, & \boldsymbol{x}_t^T\boldsymbol{n}_0 < 0 \text{ and } \boldsymbol{x}_t^T\boldsymbol{n}_1 < 0 \end{cases} \tag{32}$$

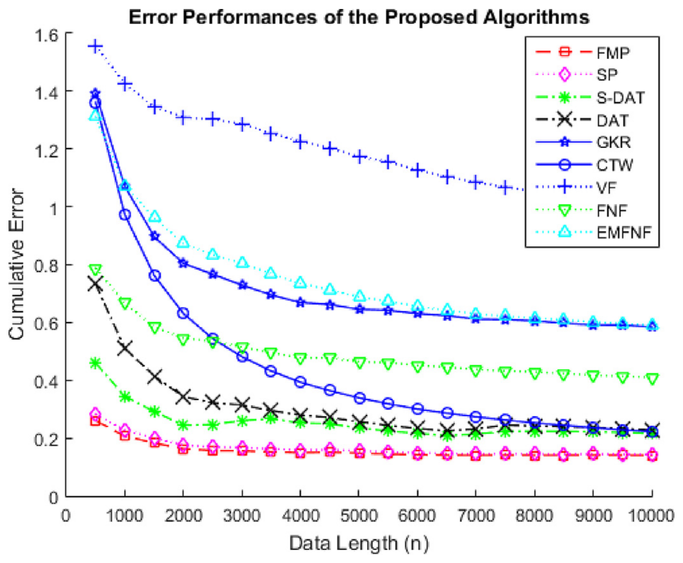

Fig. 7. Regression error performances for the matched partitioning case using model (32).

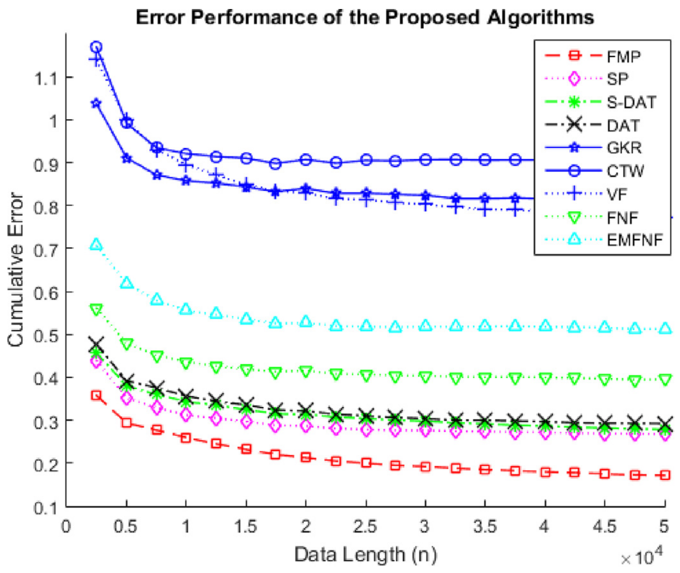

Fig. 8. Regression error performances for the mismatched partitioning case using model (33).

where $\boldsymbol{w}_1 = [1, 1]^T$, $\boldsymbol{w}_2 = [-1, -1]^T$, $\boldsymbol{n}_0 = [1, 0]^T$ and $\boldsymbol{n}_1 = [0, 1]^T$. The feature vector $\boldsymbol{x}_t = [x_{t,1}, x_{t,2}]^T$ is composed of two jointly Gaussian processes with $[0, 0]^T$ mean and $\boldsymbol{I}_2$ variance. υ_t is a sample taken from a Gaussian process with zero mean and 0.1 variance. The generated data sequence is represented by $\hat{y}_t$. In this scenario, we set the learning rates to 0.125 for the FMP, 0.0625 for the SP, 0.005 for the S-DAT, 0.01 for the DAT, 0.5 for the GKR, 0.004 for the CTW, 0.025 for the VF and the EMFNF, 0.005 for the FNF.

In Fig. 7, we represent the deterministic error performance of the specified algorithms. The algorithms VF, EMFNF, GKR and FNF cannot capture the characteristic of the data model, since these algorithms are constructed to achieve satisfactory results for smooth nonlinear models, but we examine a highly nonlinear and discontinuous model. On the other hand, the algorithms FMP, SP, S-DAT, CTW and DAT attain successive performance due to their capability of handling highly nonlinear models. As seen in Fig. 7, our algorithms, the FMP and the SP, significantly outperform their competitors and achieve almost the same performance result, since the data distribution is completely captured by both algorithms. Although the S-DAT algorithm does not perform as well as the FMP and the SP algorithms, still obtains a better convergence rate compared to the DAT and the CTW algorithms.

4.2. Mismatched partition

In this subsection, we consider the case where the desired data is generated by a piecewise linear model whose partitions do not match with the initial partitioning of the proposed algorithms. This experiment mainly focuses on to demonstrate how the proposed algorithms learn the underlying data structure. We also aim to emphasize the importance of adaptive structure.

We use the following piecewise linear model to generate the data sequence,

$$\hat{y}_t = \begin{cases} \boldsymbol{w}_1^T\boldsymbol{x}_t + \upsilon_t, & \boldsymbol{x}_t^T\boldsymbol{n}_0 \geq 0.5 \text{ and } \boldsymbol{x}_t^T\boldsymbol{n}_1 \geq -0.5 \\ \boldsymbol{w}_2^T\boldsymbol{x}_t + \upsilon_t, & \boldsymbol{x}_t^T\boldsymbol{n}_0 \geq 0.5 \text{ and } \boldsymbol{x}_t^T\boldsymbol{n}_1 < -0.5 \\ \boldsymbol{w}_2^T\boldsymbol{x}_t + \upsilon_t, & \boldsymbol{x}_t^T\boldsymbol{n}_0 < 0.5 \text{ and } \boldsymbol{x}_t^T\boldsymbol{n}_2 \geq -0.5 \\ \boldsymbol{w}_1^T\boldsymbol{x}_t + \upsilon_t, & \boldsymbol{x}_t^T\boldsymbol{n}_0 < 0.5 \text{ and } \boldsymbol{x}_t^T\boldsymbol{n}_2 < -0.5 \end{cases} \quad (33)$$

where $\boldsymbol{w}_1 = [1, 1]^T$, $\boldsymbol{w}_2 = [1, -1]^T$, $\boldsymbol{n}_0 = [2, -1]^T$, $\boldsymbol{n}_1 = [-1, 1]^T$ and $\boldsymbol{n}_2 = [2, 1]^T$. The feature vector $\boldsymbol{x}_t = [x_{t,1}, x_{t,2}]^T$ is composed of two jointly Gaussian processes with $[0, 0]^T$ mean and $\boldsymbol{I}_2$ variance. υ_t is a sample taken from a Gaussian process with zero mean and 0.1 variance. The generated data sequence is represented by $\hat{y}_t$. The learning rates are set to 0.04 for the FMP, 0.025 for the SP, 0.005 for the S-DAT, the CTW and the FNF, 0.025 for the EMFNF and the VF, 0.5 for the GKR.

In Fig. 8, we demonstrate the normalized time accumulated error performance of the proposed algorithms. Different from the matched partition scenario, we emphasize that the CTW algorithm performs even worse than the VF, the FNF and the EMFNF algorithms, which are not based on piecewise linear modeling. The reason is that the CTW algorithm has fixed regions that are mismatched with the underlying partitions. Besides, the adaptive algorithms, FMP, SP, S-DAT and DAT achieve considerably better performance, since these algorithms update their partitions in accordance with the data distribution. Comparing these four algorithms, Fig. 8 exhibits that the FMP notably outperforms its competitors, since this algorithm exactly matches its partitioning to the partitions of the piecewise linear model given in (33).

We illustrate how the FMP and the DAT algorithms update their region boundaries in Fig. 9. Both algorithms initially partition the regression space into 4 equal quadrant, i.e., the cases shown in $t = 0$. We emphasize that when the number of iterations reaches 10,000, i.e., $t = 10,000$, the FMP algorithm trains its region boundaries such that its partitions substantially match the partitioning of the piecewise linear model. However, the DAT algorithm cannot capture the data distribution yet, when $t = 10,000$. Therefore, the FMP algorithm, which uses the second order methods for training, has a faster convergence rate compared to the DAT algorithm, which updates its region boundaries using first order methods.

4.3. Real and synthetic data sets

In this subsection, we mainly focus on assessing the merits of our algorithms. We first consider the regression of a benchmark real-life problem that can be found in many data set repositories such as: California Housing, which is an $m = 8$ dimensional database consisting of the estimations of median house prices in the California area [38]. There exist more than 20,000 data samples for this dataset. For this experiment, we set the learning rates

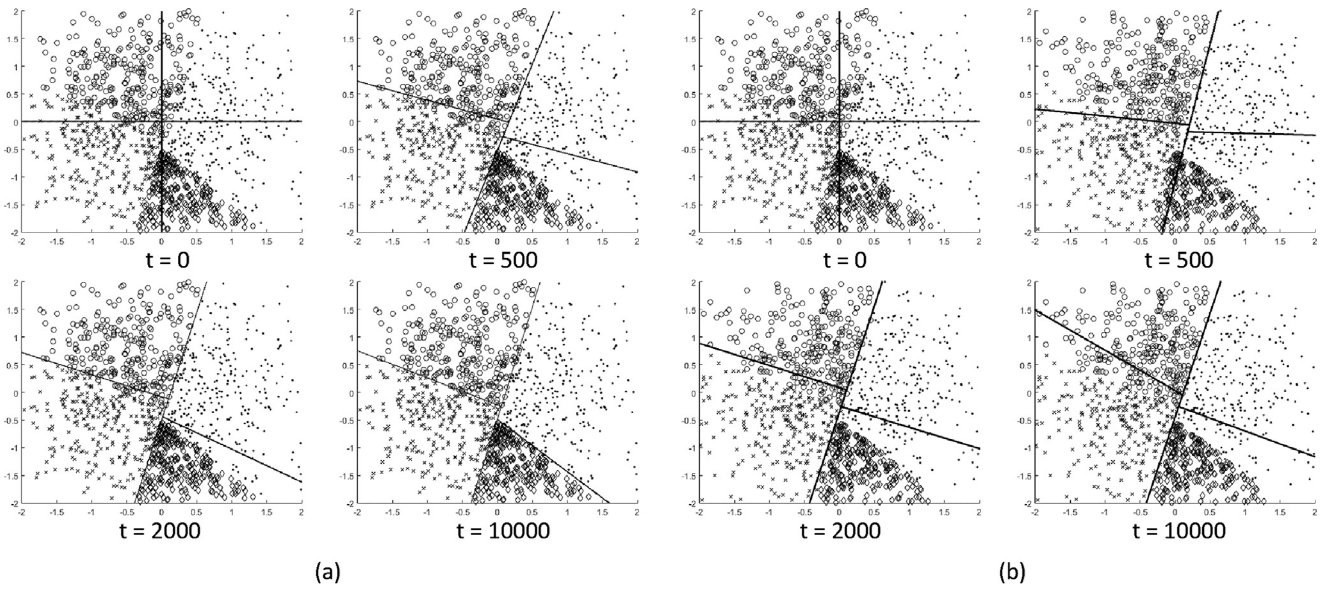

Fig. 9. Training of the separation functions for the mismatched partitioning scenario (a) FMP Algorithm (b) DAT Algorithm.

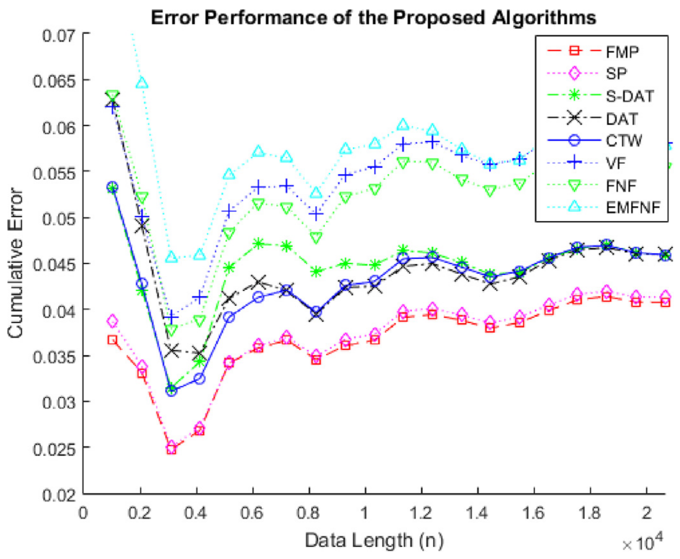

Fig. 10. Time accumulated error performances of the proposed algorithms for California Housing Data Set.

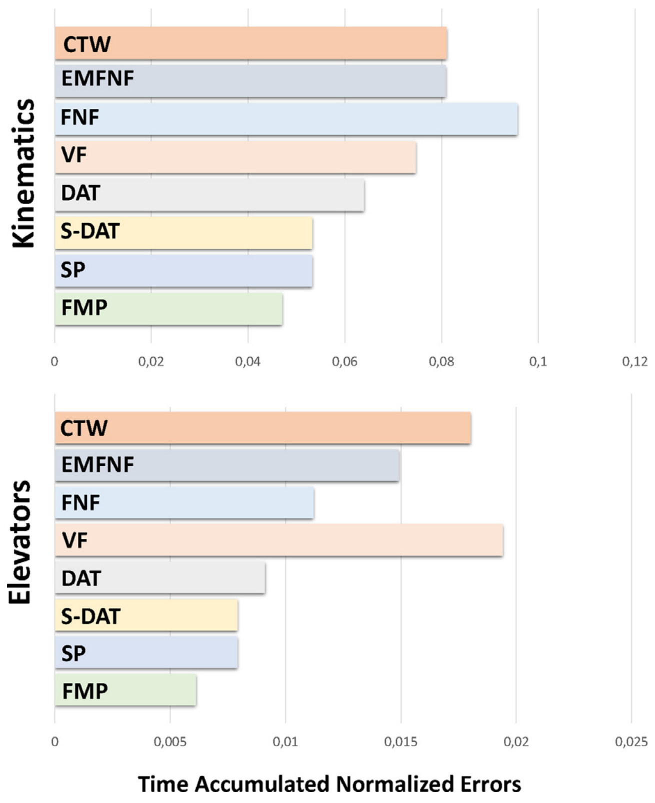

Fig. 11. Time accumulated error performances of the proposed algorithms for Kinematics and Elevators Data Sets.

to 0.004 for FMP and SP, 0.01 for the S-DAT and the DAT, 0.02 for the CTW, 0.05 for the VF, 0.005 for the FNF and the EMFNF. Fig. 10 illustrates the normalized time accumulated error rates of the stated algorithms. We emphasize that the FMP and the SP significantly outperforms the state of the art.

We also consider two more real and synthetic data sets. The first one is Kinematics, which is an $m = 8$ dimensional dataset where a realistic simulation of an 8 link robot arm is performed [38]. The task is to predict the distance of the end-effector from a target. There exist more than 50000 data samples. The second one is Elevators, which has an $m = 16$ dimensional data sequence obtained from the task of controlling an F16 aircraft [38]. This dataset provides more than 50000 samples. In Fig. 11, we present the steady state error performances of the proposed algorithms. We emphasize that our algorithms achieve considerably better performance compared to the others for both datasets.

Special to this subsection, we perform an additional experiment using the Kinematics dataset to illustrate the effect of using second order methods for the adaptation. Usually, algorithms like CTW, FNF, EMFNF, VF and DAT use the gradient based first order methods for the adaptation algorithm due to their low computational demand. Here, we modified the adaptation part of these algorithms and use the second order Newton–Raphson methods instead. In Fig. 12, we illustrate a comparison that involves the final error rates of both the modified and the original algorithms. We also keep our algorithms in their original settings to demonstrate the effect of using piecewise linear functions when the same adaptation algorithm is used. In Fig. 12, the CTW-2, the EMFNF-2,

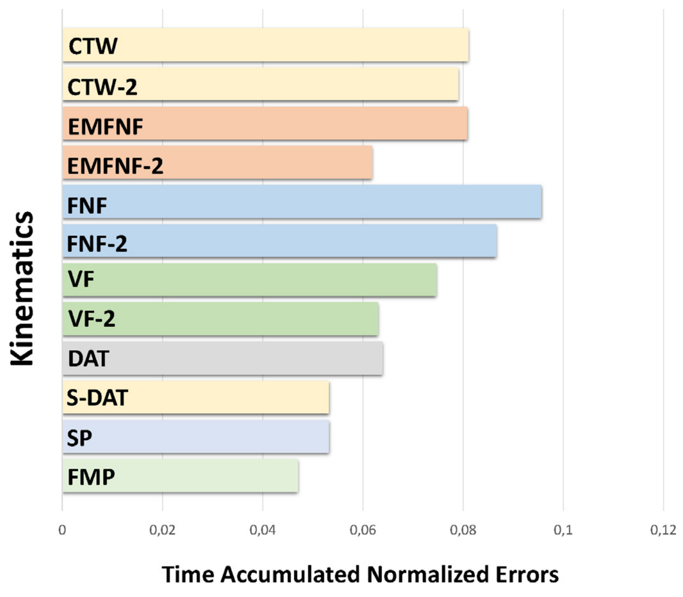

Fig. 12. Time accumulated error performances of the proposed algorithms for Kinematics Data Set. The second order adaptation methods are used for all algorithms.

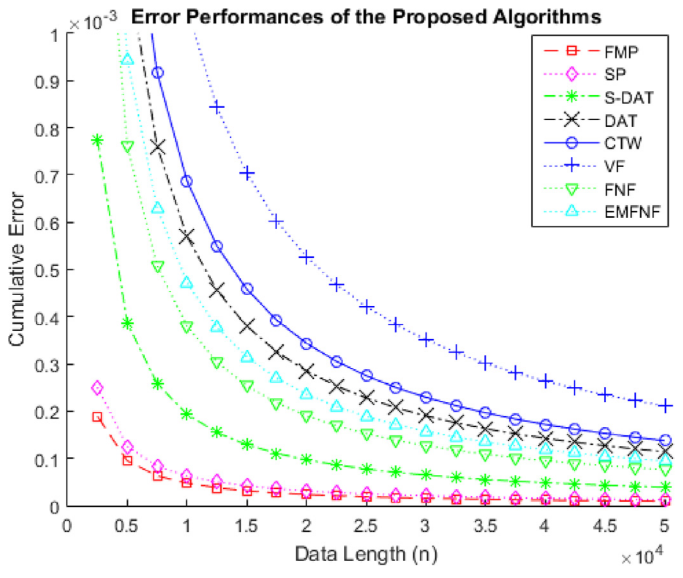

Fig. 13. Regression error rates for the Gauss map.

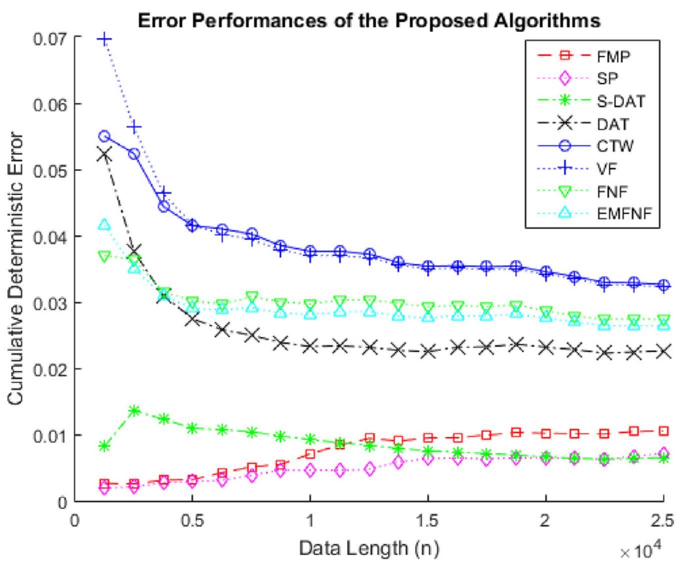

Fig. 14. Regression error rates for the Lorenz attractor.

the FNF-2 and the VF-2 state for the algorithms using the second order methods for the adaptation. The presented S-DAT algorithm already corresponds to the DAT algorithm with the second order adaptation methods. Even though this modification decreases the final error of all algorithms, our algorithms still outperform their competitors. Additionally, in terms of the computational complexity, the algorithms EMFNF-2, FNF-2 and VF-2 become more costly compared to the proposed algorithms since they now use the second order methods for the adaptation. There exist only one algorithm, i.e., CTW-2, that is more efficient, but it does not achieve a significant gain on the error performance.

4.4. Chaotic signals

Finally, we examine the error performance of our algorithms when the desired data sequence is generated using chaotic processes, e.g. the Gauss map and the Lorenz attractor. We first consider the case where the data is generated using the Gauss map, i.e.,

$$y_t = \exp(-\alpha x_t^2) + \beta \tag{34}$$

which exhibits a chaotic behavior for $\alpha = 4$ and $\beta = 0.5$. The desired data sequence is represented by y_t and $x_t \in \mathbb{R}$ corresponds to y_{t-1}. x_0 is a sample from a Gaussian process with zero-mean and unit variance. The learning rates are set to 0.004 for the FMP, 0.04 for the SP, 0.05 for the S-DAT and the DAT, 0.025 for the VF, the FNF, the EMFNF and the CTW.

As the second experiment, we consider a scenario where we use a chaotic signal that is generated from the Lorenz attractor, which is a set of chaotic solutions for the Lorenz system. Hence, the desired signal y_t is modeled by

$$y_t = y_{t-1} + (\sigma(u_{t-1} - y_{t-1}))dt \tag{35}$$

$$u_t = u_{t-1} + (y_{t-1}(\rho - v_{t-1}) - u_{t-1})dt \tag{36}$$

$$v_t = v_{t-1} + (y_{t-1}u_{t-1} - \beta v_{t-1})dt, \tag{37}$$

where $\beta = 8/3$, $\sigma = 10$, $\rho = 28$ and $dt = 0.01$. Here, u_t and v_t are used to represent the two dimensional regression space, i.e., the data vector is formed as $\boldsymbol{x}_t = [u_t, v_t]^T$. We set the learning rates to 0.005 for the FMP, 0.006 for the SP, 0.0125 for the S-DAT, 0.01 for the DAT, the VF, the FNF, the EMFNF and the CTW.

In Figs. 13 and 14, we represent the error performance of the proposed algorithms for the Gauss map and the Lorenz attractor cases respectively. In both cases, the proposed algorithms attain substantially faster convergence rate and better steady state error performance compared to the state of the art. Even for the Lorenz attractor case, where the desired signal has a dependence on more than one past output samples, our algorithms outperform the competitors.

Before concluding the Simulation section, we need to emphasize that it is a difficult task to provide completely fair scenarios for assessing the performance of nonlinear filters. The reason is that, for any particular nonlinear method, it is very likely to find a specific case where this method outperforms its competitors. Therefore, there might exist some other situations where our methods would not perform as well as they do for the cases given above. Nevertheless, we focus on the above scenarios and the datasets since they are well-known and highly used in signal processing literature for performance assessment. Hence, they provide a significant insight about the overall performance of our algorithms.

5. Concluding remarks

In this paper, we introduce three different highly efficient and effective nonlinear regression algorithms for online learning problems suitable for real life applications. We process only the currently available data for regression and then discard it, i.e., there is no need for storage. For nonlinear modeling, we use piecewise linear models, where we partition the regressor space using linear separators and fit linear regressors to each partition. We construct our algorithms based on two different approaches for the partitioning of the space of the regressors. As the first time in the literature, we adaptively update both the region boundaries and the linear regressors in each region using the second order methods, i.e., Newton-Raphson Methods. We illustrate that the proposed algorithms attain outstanding performance compared to the state of art even for the highly nonlinear data models. We also provide the individual sequence results demonstrating the guaranteed regret performance of the introduced algorithms without any statistical assumptions.

Acknowledgment

This work is supported in part by Turkish Academy of Sciences Outstanding Researcher Programme, TUBITAK Contract No. 113E517, and Turk Telekom Communications Services Incorporated.

References

[1] A. Ingle, J. Bucklew, W. Sethares, T. Varghese, Slope estimation in noisy piecewise linear functions, Signal Process. 108 (2015) 576–588, doi:10.1016/j.sigpro.2014.10.003.
[2] M. Scarpiniti, D. Comminiello, R. Parisi, A. Uncini, Nonlinear spline adaptive filtering, Signal Process. 93 (4) (2013) 772–783, doi:10.1016/j.sigpro.2012.09.021.
[3] Y. Yilmaz, X. Wang, Sequential distributed detection in energy-constrained wireless sensor networks, IEEE Trans. Signal Process. 17 (4) (2014) 335–339.
[4] A.H. Sayed, Fundamentals of Adaptive Filtering, John Wiley & Sons, NJ, 2003.
[5] X. Wu, X. Zhu, G.-Q. Wu, W. Ding, Data mining with big data, IEEE Trans. Knowl. Data Eng. 26 (1) (2014) 97–107, doi:10.1109/TKDE.2013.109.
[6] T. Moon, T. Weissman, Universal FIR MMSE filtering, IEEE Trans. Signal Process. 57 (3) (2009) 1068–1083, doi:10.1109/TSP.2008.2009894.
[7] S.S. Kozat, A.C. Singer, A.J. Bean, A tree-weighting approach to sequential decision problems with multiplicative loss, Signal Process. 91 (4) (2011) 890–905, doi:10.1016/j.sigpro.2010.09.007.
[8] N. Asadi, J. Lin, A. de Vries, Runtime optimizations for tree-based machine learning models, IEEE Trans. Knowl. Data Eng. 26 (9) (2014) 2281–2292, doi:10.1109/TKDE.2013.73.
[9] A.C. Singer, G.W. Wornell, A.V. Oppenheim, Nonlinear autoregressive modeling and estimation in the presence of noise, Digital Signal Process. 4 (4) (1994) 207–221.
[10] O.J.J. Michel, A.O. Hero, A.-E. Badel, Tree-structured nonlinear signal modeling and prediction, IEEE Trans. Signal Process. 47 (11) (1999) 3027–3041, doi:10.1109/78.796437.
[11] W. Cao, L. Cao, Y. Song, Coupled market behavior based financial crisis detection, in: The 2013 International Joint Conference on Neural Networks (IJCNN), 2013, pp. 1–8, doi:10.1109/IJCNN.2013.6706966.
[12] L. Deng, Long-term trend in non-stationary time series with nonlinear analysis techniques, in: 2013 6th International Congress on Image and Signal Processing (CISP), 2, 2013, pp. 1160–1163, doi:10.1109/CISP.2013.6745231.
[13] K. mei Zheng, X. Qian, N. An, Supervised non-linear dimensionality reduction techniques for classification in intrusion detection, in: 2010 International Conference on Artificial Intelligence and Computational Intelligence (AICI), 1, 2010, pp. 438–442, doi:10.1109/AICI.2010.98.
[14] S. Kabbur, G. Karypis, Nlmf: Nonlinear matrix factorization methods for top-n recommender systems, in: 2014 IEEE International Conference on Data Mining Workshop (ICDMW), 2014, pp. 167–174, doi:10.1109/ICDMW.2014.108.
[15] R. Couillet, M. Debbah, Signal processing in large systems, IEEE Signal Process. Mag. 24 (2013) 211–317.
[16] L. Bottou, Y.L. Cun, Online learning for very large data sets, Appl. Stochastic Models Bus. Ind. 21 (2005) 137–151.
[17] L. Bottou, O. Bousquet, The tradeoffs of large scale learning, in: Advances in Neural Information Processing (NISP), 2007, pp. 1–8.
[18] N. Cesa-Bianchi, G. Lugosi, Prediction, Learning, and Games, Cambridge University Press, Cambridge, 2006.
[19] A.C. Singer, S.S. Kozat, M. Feder, Universal linear least squares prediction: upper and lower bounds, IEEE Trans. Inf. Theory 48 (8) (2002) 2354–2362, doi:10.1109/TIT.2002.800489.
[20] 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 (8) (2010) 4050–4063.
[21] Y. Yilmaz, S. Kozat, Competitive randomized nonlinear prediction under additive noise, Signal Process. Lett., IEEE 17 (4) (2010) 335–339, doi:10.1109/LSP.2009.2039950.
[22] S. Dasgupta, Y. Freund, Random projection trees for vector quantization, IEEE Trans. Inf. Theory 55 (7) (2009) 3229–3242, doi:10.1109/TIT.2009.2021326.
[23] D.P. Helmbold, R.E. Schapire, Predicting nearly as well as the best pruning of a decision tree, Mach. Learn. 27 (1) (1997) 51–68.
[24] S.S. Kozat, A.C. Singer, G.C. Zeitler, Universal piecewise linear prediction via context trees, IEEE Trans. Signal Process. 55 (7) (2007) 3730–3745.
[25] D. Bertsimas, J.N. Tsitsiklis, Introduction to Linear Optimization, Athena scientific series in optimization and neural computation, Athena Scientific, Belmont (Mass.), 1997. URL http://opac.inria.fr/record=b1094316
[26] E.D. Kolaczyk, R.D. Nowak, Multiscale generalised linear models for nonparametric function estimation, Biometrika 92 (1) (2005) 119–133, doi:10.1093/biomet/92.1.119. URL http://biomet.oxfordjournals.org/content/92/1/119.abstract
[27] F.M.J. Willems, Y.M. Shtarkov, T.J. Tjalkens, The context-tree weighting method: basic properties, IEEE Trans. Inf. Theory 41 (3) (1995) 653–664, doi:10.1109/18.382012.
[28] A.C. Singer, M. Feder, Universal linear prediction by model order weighting, IEEE Trans. Signal Process. 47 (10) (1999) 2685–2699, doi:10.1109/78.790651.
[29] A. Gyorgy, T. Linder, G. Lugosi, Efficient adaptive algorithms and minimax bounds for zero-delay lossy source coding, IEEE Trans. Signal Process. 52 (8) (2004) 2337–2347, doi:10.1109/TSP.2004.831128.
[30] N. Vanli, S. Kozat, A comprehensive approach to universal piecewise nonlinear regression based on trees, IEEE Trans. Signal Process. 62 (20) (2014) 5471–5486, doi:10.1109/TSP.2014.2349882.
[31] M.S.D. Raghunath S. Holambe, Advances in Nonlinear Modeling for Speech Processing, Adaptive computation and machine learning series, Springer, 2012.
[32] K.P. Murphy, Machine learning : A probabilistic perspective, Adaptive computation and machine learning series, MIT Press, Cambridge (Mass.), 2012. URL http://opac.inria.fr/record=b1134263
[33] M. Mattavelli, J. Vesin, E. Amaldi, R. Gruter, A new approach to piecewise linear modeling of time series, in: Digital Signal Processing Workshop Proceedings, 1996., IEEE, 1996, pp. 502–505, doi:10.1109/DSPWS.1996.555572.
[34] E. Hazan, A. Agarwal, S. Kale, Logarithmic regret algorithms for online convex optimization, Mach. Learn. 69 (2-3) (2007) 169–192.
[35] R. Rosipal, L.J. Trejo, Kernel partial least squares regression in reproducing kernel hilbert space, J. Mach. Learn. Res. 2 (2002) 97–123. URL http://dl.acm.org/citation.cfm?id=944790.944806
[36] M. Schetzen, The Volterra and Wiener Theories of Nonlinear Systems, John Wiley & Sons, NJ, 1980.
[37] A. Carini, G.L. Sicuranza, Fourier nonlinear filters, Signal Process. 94 (0) (2014) 183–194, doi:10.1016/j.sigpro.2013.06.018.
[38] L. Torgo, Regression data sets. URL http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html.