SEQUENTIAL REGRESSION TECHNIQUES WITH SECOND ORDER METHODS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Burak Cevat Civek
July 2017

SEQUENTIAL REGRESSION TECHNIQUES WITH SECOND ORDER METHODS
By Burak Cevat Civek
July 2017

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Süleyman Serdar Kozat(Advisor)

Nail Akar

Çağatay Candan

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

SEQUENTIAL REGRESSION TECHNIQUES WITH SECOND ORDER METHODS

Burak Cevat Civek
M.S. in Electrical and Electronics Engineering
Advisor: Süleyman Serdar Kozat
July 2017

Sequential regression problem is one of the widely investigated topics in the machine learning and the signal processing literatures. In order to adequately model the underlying structure of the real life data sequences, many regression methods employ nonlinear modeling approaches. In this context, in the first chapter, we introduce highly efficient sequential 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. 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. 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 and achieve substantial performance compared to the state of the art. In the second chapter, we investigate the problem of sequential prediction for real life big data applications. The second order Newton-Raphson methods asymptotically achieve the performance of the "best" possible predictor much faster compared to the first order algorithms. However, their usage in real life big data applications is prohibited because of the extremely high computational needs. To this end, in order to enjoy the outstanding performance of the second order methods, we introduce a highly efficient implementation where the computational complexity is reduced from quadratic to linear scale. For both chapters, we demonstrate our gains over the well-known benchmark and real life data sets and provide performance results in an individual sequence manner guaranteed to hold without any statistical assumptions.

Keywords: Piecewise linear regression, hierarchical tree, second order methods, sequential prediction, big data.

ÖZET

İKİNCİ DERECEDEN YÖNTEMLER İLE ARDIŞIK BAĞLANIM TEKNİKLERİ

Burak Cevat Civek
Elektrik Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Temmuz 2017

Ardışık bağlanım problemi, makine öğrenmesi ve sinyal işleme literatürlerinde geniş ölçüde incelenmektedir. Gerçek hayat veri dizilerinin altında yatan yapıyı başarılı bir şekilde modelleyebilmek adına birçok bağlanım yöntemi doğrusal olmayan yaklaşımlar kullanmaktadır. Bu bağlamda, birinci bölümde, gerçek hayat uygulamaları için son derece verimli ardışık doğrusal olmayan bağlanım algoritmaları sunulmaktadır. Elde edilen veriler depolama ihtiyacı duyulmadan çevrimiçi olarak işlenmekedir. Doğrusal olmayan modelleme için, bağlanım uzayının uyarlanır olarak parçalandığı, karar ağacı kavramına dayalı hiyerarşik parçalı doğrusal bir yaklaşım kullanılmaktadır. Literatürde ilk defa, oldukça etkili ikinci dereceden Newton-Raphson yöntemleri ile, bölge sınırları ve her bir bölgedeki doğrusal modeller öğrenilmektedir. Parçalı doğrusal modellerin kullanılması ile, yaygın olarak karşılaşılan aşırı uyma sorunundan kaçınılmakta ve var olan algoritmalara kıyasla önemli ölçüde performans elde edilmektedir. İkinci bölümde, gerçek hayat büyük veri uygulamaları için ardışık kestirim problemi incelenmektedir. İkinci dereceden Newton-Raphson yöntemleri, birinci dereceden algoritmalara kıyasla "en iyi" kestirici performansını asimptotik olarak çok daha hızlı bir şekilde elde etmektedir. Fakat, oldukça yüksek hesaplama yükü sebebiyle bu yöntemlerin kullanımı büyük veri uygulamalarında kısıtlanmıştır. Bu durumda, ikinci dereceden yöntemlerin üstün performansından faydalanmak amacıyla, hesaplama karmaşıklığının karesel seviyeden doğrusal seviyeye indirgendiği bir uygulama sunulmaktadır. Her iki bölümde elde edilen kazanımlar iyi bilinen kıyaslama ve gerçek hayat veri setleri üzerinden gösterilmekte ve istatistiksel varsayımlar olmaksızın garantilenen performans sonuçları elde edilmektedir.

Anahtar sözcükler: Parçalı doğrusal bağlanım, hiyerarşik ağaç, ikinci dereceden yöntemler, ardışık kestirim, büyük veri.

Acknowledgement

I would first like to present my eternal gratitude to my advisor, Assoc. Prof. Süleyman Serdar Kozat, with my most sincere feelings. His priceless effort to guide me throughout my graduate studies helped me a lot to achieve this thesis. I am very grateful for completing my degree under his supervision. I believe that it would not be possible for me to achieve this work without his excellent support.

I would like to thank my parents and little sister for encouraging me in every step of my graduate studies. Their valuable support at the beginning lead me to continue this challenging period.

I would also like to thank TÜBİTAK for supporting me through BİDEB 2210-A Scholarship Program.

Finally, my greatest and deepest appreciation is for my wife and love. She always found a way to create an inspirational and peaceful environment with her elegance and courtesy. She has been my moral support and she will always be. She has provided me the most valuable assistance during my studies.

Contents

List of Figures

List of Tables

Chapter 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, many regression techniques are currently employing nonlinear modeling approaches since the real life data sequences have highly nonlinear structures.

In the first chapter, we investigate the nonlinear regression problem that is one of the most important topics in the machine learning and the 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]. From a unified point of view, in such problems, we sequentially observe a real valued 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.

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 sequential 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]. In order to introduce nonlinearities, we employ piecewise linear models due to their considerable power of adequately approximating nonlinear models. In piecewise linear modeling, the space of the regression vectors is partitioned into several distinct regions and a linear model is assigned to each region. We use the notion of decision trees to determine the region boundaries. Different from the current literature, we adaptively train both the region boundaries and the linear models in each partition using the second order adaptation algorithms. As a result of this chapter, we obtain highly efficient hierarchical sequential nonlinear regression algorithms.

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 [22, 23]. 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 [22, 23]. 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 [24]. 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 [25]. For instance, there is an algorithmic concern on the definitions of both

the exponentially weighted performance measure and the "universal weighting" coefficients [19, 23, 26, 27] 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 [23]. Besides these approaches, there exists an algorithm providing adaptive tree structure for the partitions, e.g., the Decision Adaptive Tree (DAT) [28]. 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 the second chapter of the thesis, we investigate the widely studied sequential prediction problem for high dimensional data streams. Efficient prediction algorithms specific to big data sequences have great importance for several real life applications such as high frequency trading [29], forecasting [11], trend analysis [12], financial market [30] and locational tracking [31]. Unfortunately, conventional methods in machine learning and signal processing literatures are inadequate to efficiently and effectively process high dimensional data sequences [32–34]. Even though today's computers have powerful processing units, traditional algorithms creates a bottleneck even for that processing power when the data is acquired

at high speeds and too large in size [32, 33]. These problems bring an essential requirement for an algorithm that is both computationally feasible and highly effective in terms of performance.

In the current literature, there exist second order sequential prediction algorithms, i.e., Newton-Raphson methods, that provide sufficiently high performance in terms of both convergence and steady-state error rates. However, the computational complexity of the second order methods is in the order of $O(M^2)$ for a feature vector of length M. Hence, their usage in real life big data applications is prohibited because of this extremely high computational need. Different from the second order methods, the first order algorithms offer a computational complexity in the order of $O(M)$, which leads the big data applications to use the first order algorithms. However, these methods perform insufficiently compared to the second order methods. As a result, in the sequential prediction literature, there is a need for outperforming algorithms with low computational demands.

In sequential prediction, the consecutive feature vectors are the shifted versions of each other, i.e., for a feature vector of $\boldsymbol{x}_t = [x_t, x_{t-1}, ..., x_{t-M}]^T$, the upcoming vector is in the form of $\boldsymbol{x}_{t+1} = [x_{t+1}, x_t, ..., x_{t-M+1}]^T$. To this end, we introduce second order methods for this important case with computational complexity only linear in the data dimension, i.e., $O(M)$. We achieve such an enormous reduction in computational complexity since there are only two entries changing from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$, where we avoid unnecessary calculations in each update. We do not use any statistical assumption on the data sequence other than the shifted nature of the feature vectors. Therefore, we present an approach that is highly appealing for big data applications since it provides the merits of the Newton-Raphson methods with a much lower computational cost.

Overall in this thesis, 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 $\boldsymbol{I}_n$. The n dimensional vector with all zero entries is denoted by $\boldsymbol{0}_n$

Chapter 2

Highly Efficient Hierarchical Sequential Nonlinear Regression Algorithms Using Second Order Methods

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 performance [10, 22, 35]. 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 [23]. 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 [36]. 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 [36]. We demonstrate our gains over the well known benchmark and real life 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.

This chapter is organized as follow. We first present the main framework for nonlinear regression and piecewise linear modeling in Section 2.1. Then, we propose three algorithms with regressor space partitioning and present guaranteed upper bounds on the performances in Section 2.2. We then demonstrate the performance of our algorithms through widely used benchmark and real life data sets in Section 2.3.

2.1 Problem Description

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. 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 [37]. Different from linear regressors, nonlinear functions are

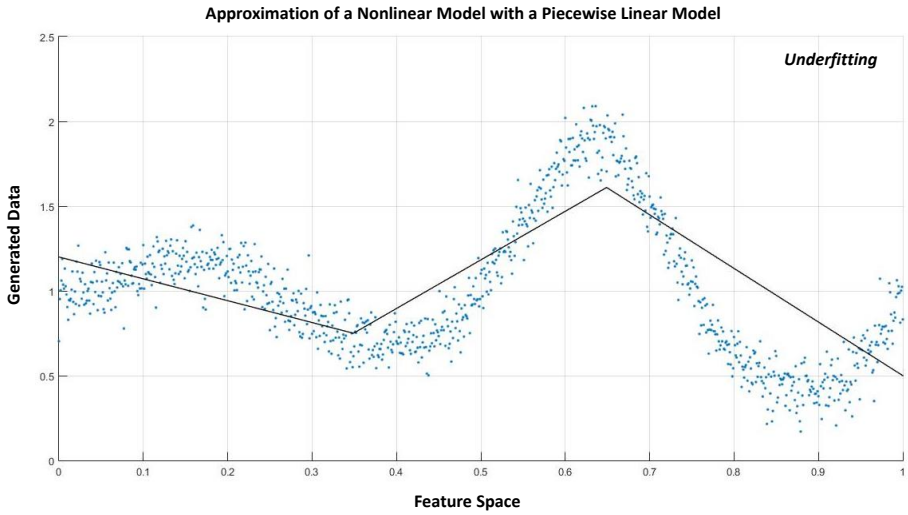

Figure 2.1: Inadequate approximation of a nonlinear model. The feature space is not partitioned sufficiently.

quite powerful and usually overfit in most real life cases [38]. To this end, we choose piecewise linear functions due to their capability of approximating most nonlinear models [39]. 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.,

$$\hat{y}_{t,i} = \boldsymbol{w}_{t,i}^T \boldsymbol{x}_t + c_{t,i}, \tag{2.1}$$

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 i^{th} region. We represent $\hat{y}_{t,i}$ in a more compact form as

$$\hat{y}_{t,i} = \boldsymbol{w}_{t,i}^T \boldsymbol{x}_t \tag{2.2}$$

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 Figs. 2.1, 2.2, 2.3, we present a one dimensional regression problem, where we generate the data sequence using the nonlinear model

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

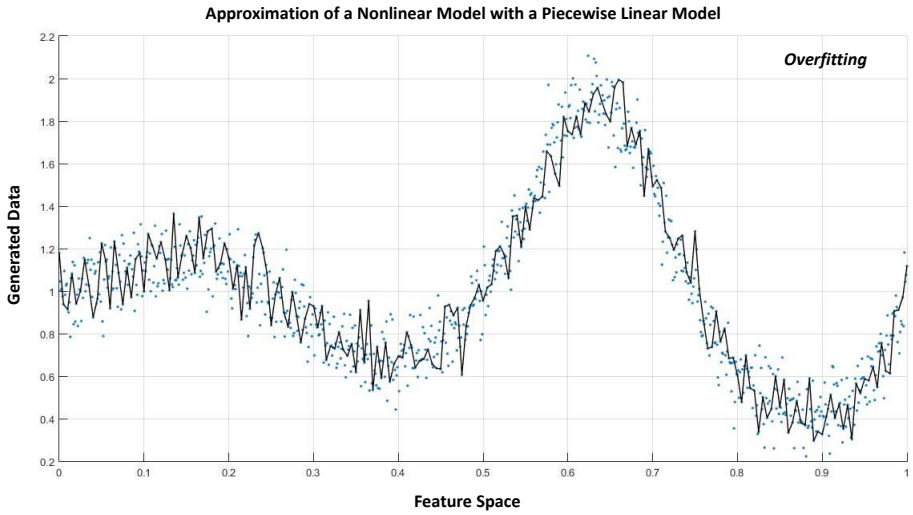

Figure 2.2: Approximation of a nonlinear model which results in overfitting problem.

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 Fig. 2.1, 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. This situation is known as the underfitting issue. Another problematic case is represented in Fig. 2.2, where the partitions are selected as quite smaller regions compared to the underfitting situation. In this case, the algorithm memorize the data structure rather than learning it. This case is known as the overfitting situation. In order to better model the generated data, we use the model shown in Fig. 2.3, where we have eight regions particularly selected according to the distribution of the data points. As these three cases 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 and overfitting cases demonstrated by Figs. 2.1 and 2.2 respectively [35]. The second problem is to find out the linear model that best fits the data in each distinct region in a sequential manner [23]. In this thesis, we solve both of these

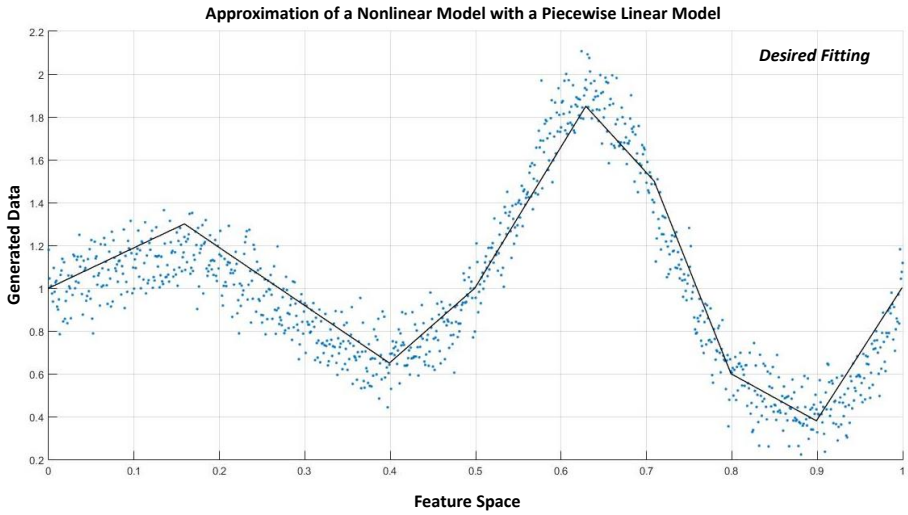

Figure 2.3: An adequate approximation of a nonlinear model. The underlying structure of the data sequence is sufficiently captured.

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 [40], for training of the region boundaries and the linear models.

2.2 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

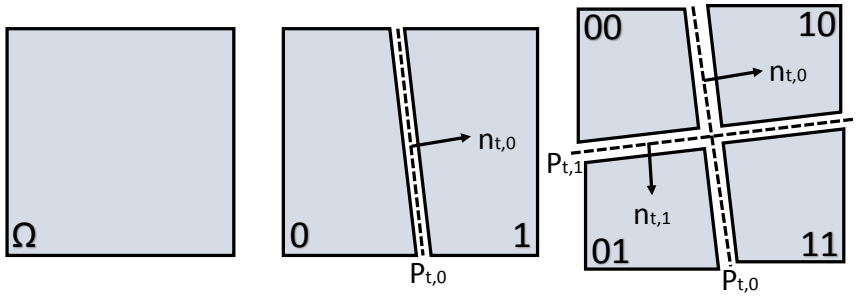

Figure 2.4: Straight Partitioning of The Regression Space

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. 2.4. 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. 2.4. 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. 2.5, i.e.,

$$p_{t,k} = \frac{1}{1 + e^{-\boldsymbol{x}_t^T \boldsymbol{n}_{t,k}}} \tag{2.3}$$

where the offset $c_{t,k}$ is included in the norm vector $\boldsymbol{n}_{t,k}$ as a bias term. In Fig. 2.5, 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

$$\hat{y}_{t,r} = \boldsymbol{w}_{t,r}^T \boldsymbol{x}_t \tag{2.4}$$

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

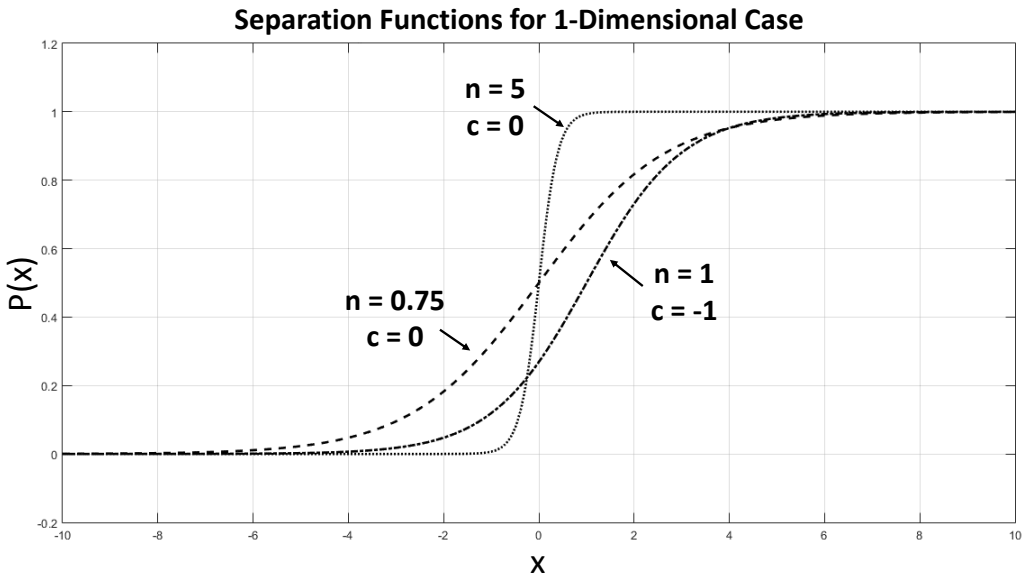

Figure 2.5: Separation Functions for 1-Dimensional Case where $\{n = 5, c = 0\}$, $\{n = 0.75, c = 0\}$ and $\{n = 1, c = -1\}$. Parameter n specifies the sharpness, as c determines the position or the offset on the x-axis.

our algorithms.

2.2.1 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.

2.2.1.1 Type 1 Partitioning

In this method, we allow each hyperplane to divide the whole space into two subspaces as shown in Fig. 2.4. 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. 2.4. 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

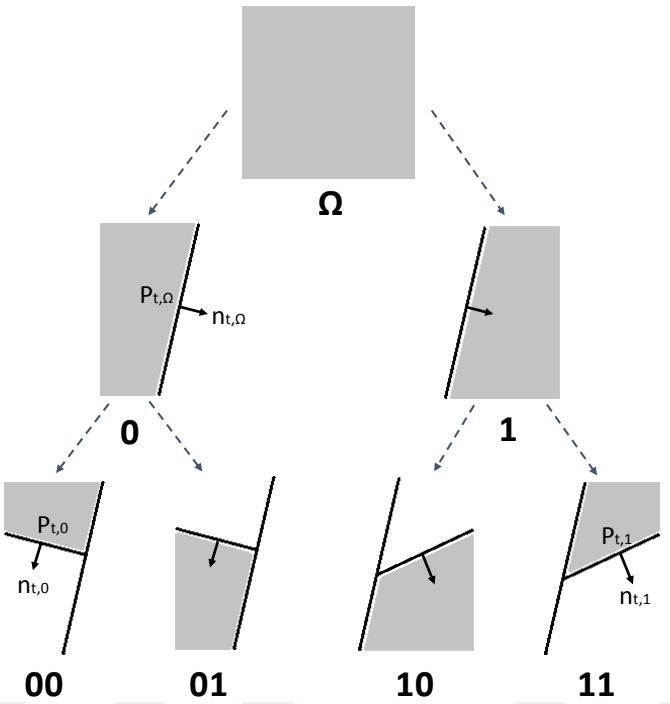

Figure 2.6: Tree Based Partitioning of The Regression Space

$\{\Omega\}$. 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.

2.2.1.2 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, 35]. We illustrate this method in Fig. 2.6 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. 2.6 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. 2.6.

Overall, Type 1 partitioning is defined as the straight lines splitting the whole regressor space into two separate regions. Differently, Type 2 partitioning has a recursive tree structure, where each separator function performs the splitting operation on just a single region, not on the whole space. Using Type 1 partitioning, the mean square error after n iteration is given by $\ell_1^n = \sum_{t=1}^{n}(y_t - \hat{y}_t^{(1)})^2$, where $\hat{y}_t^{(1)}$ is defined as

$$\hat{y}_t^{(1)} = p_{t,0}^{(1)} p_{t,1}^{(1)} \hat{y}_{t,00} + p_{t,0}^{(1)}(1-p_{t,1}^{(1)})\hat{y}_{t,01} + (1-p_{t,0}^{(1)}) p_{t,1}^{(1)} \hat{y}_{t,10} + (1-p_{t,0}^{(1)})(1-p_{t,1}^{(1)})\hat{y}_{t,11}.$$

When using Type 2 partitioning, the mean square error definition becomes $\ell_2^n = \sum_{t=1}^{n}(y_t - \hat{y}_t^{(2)})^2$, where $\hat{y}_t^{(2)}$ is defined as

$$\hat{y}_t^{(2)} = p_{t,\Omega}^{(2)} p_{t,0}^{(2)} \hat{y}_{t,00} + p_{t,\Omega}^{(2)}(1-p_{t,0}^{(2)})\hat{y}_{t,01} + (1-p_{t,\Omega}^{(2)}) p_{t,1}^{(2)} \hat{y}_{t,10} + (1-p_{t,\Omega}^{(2)})(1-p_{t,1}^{(2)})\hat{y}_{t,11}.$$

Here, starting from the initial step, i.e., $t=1$, Type 2 partitioning has the flexibility of choosing $p_{t,\Omega}^{(2)} = p_{t,0}^{(1)}$ and $p_{t,0}^{(2)} = p_{t,1}^{(2)} = p_{t,1}^{(1)}$ at each iteration. Hence, all possible models generated by Type 1 partitioning can also be achieved by Type 2. However, reverse is not possible if $p_{t,0}^{(2)} \neq p_{t,1}^{(2)}$. Consequently, the relation $\ell_1^n = \sum_{t=1}^{n}(y_t - \hat{y}_t^{(1)})^2 \geq \sum_{t=1}^{n}(y_t - \hat{y}_t^{(2)})^2 = \ell_2^n$ holds for all $t \geq 1$, implying that Type 2 partitioning achieves a better steady state mean square error performance compared to Type 1. However, Type 1 partitioning requires less number

of parameters to generate the same number of regions. To this end, in online settings, Type 1 partitioning might outperform in the transient region in terms of sequential regression error, even if Type 2 partitioning finally offers a better steady state performance. Therefore, both Type 1 and Type 2 partitioning have their own advantages.

2.2.2 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. 2.4, 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

$$\begin{aligned}\hat{y}_t = p_{t,0}p_{t,1}\hat{y}_{t,00} + p_{t,0}(1-p_{t,1})\hat{y}_{t,01} \\ + (1-p_{t,0})p_{t,1}\hat{y}_{t,10} + (1-p_{t,0})(1-p_{t,1})\hat{y}_{t,11}\end{aligned} \quad (2.5)$$

when we observe the feature vector $\boldsymbol{x}_t$. This result can be easily extended to the cases where we have more then 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 [40], to update both separator functions and region weights. To accomplish this, the weight vector assigned to the region $\{00\}$ is updated as

$$\begin{aligned}\boldsymbol{w}_{t+1,00} &= \boldsymbol{w}_{t,00} - \frac{1}{\beta}\boldsymbol{A}_t^{-1}\nabla e_t^2 \\ &= \boldsymbol{w}_{t,00} + \frac{2}{\beta}e_t p_{t,0}p_{t,1}\boldsymbol{A}_t^{-1}\boldsymbol{x}_t,\end{aligned} \quad (2.6)$$

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

$$\boldsymbol{A}_t = \sum_{i=1}^{t} \nabla_i \nabla_i^T + \epsilon \boldsymbol{I}_m, \tag{2.7}$$

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 [40].

Region boundaries are also updated in the same manner. For example, the direction vector specifying the separation function $p_{t,0}$ in Fig. 2.4, is updated as

$$\begin{aligned} \boldsymbol{n}_{t+1,0} &= \boldsymbol{n}_{t,0} - \frac{1}{\eta} \boldsymbol{A}_t^{-1} \nabla e_t^2 \\ &= \boldsymbol{n}_{t,0} + \frac{2}{\eta} e_t [p_{t,1} \hat{y}_{t,00} + (1 - p_{t,1}) \hat{y}_{t,01} \\ &\quad - p_{t,1} \hat{y}_{t,10} - (1 - p_{t,1}) \hat{y}_{t,11}] \boldsymbol{A}_t^{-1} \frac{\partial p_{t,0}}{\partial \boldsymbol{n}_{t,0}}, \end{aligned} \tag{2.8}$$

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 (3.4). Partial derivative of the separation function $p_{t,0}$ w.r.t. $\boldsymbol{n}_{t,0}$ is given by

$$\frac{\partial p_{t,0}}{\partial \boldsymbol{n}_{t,0}} = \frac{\boldsymbol{x}_t e^{-\boldsymbol{x}_t^T \boldsymbol{n}_{t,0}}}{(1 + e^{-\boldsymbol{x}_t^T \boldsymbol{n}_{t,0}})^2}. \tag{2.9}$$

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

$$\hat{y}_t = \sum_{r \in R} \hat{\psi}_{t,r}, \tag{2.10}$$

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. 2.4. Weighted estimate of each region is determined by

$$\hat{\psi}_{t,r} = \hat{y}_{t,r} \prod_{i=1}^{K} \hat{p}_{t,P(i)}, \tag{2.11}$$

where K is the number of separation functions, P represents the set of all separation function labels and $P(i)$ is the i^{th} element of set P, e.g. $P = \{0, 1\}, P(1) = 0$,

and $\hat{p}_{t,P(i)}$ is defined as

$$\hat{p}_{t,P(i)} = \begin{cases} p_{t,P(i)} & , r(i) = 0 \\ 1 - p_{t,P(i)} & , r(i) = 1 \end{cases}, \tag{2.12}$$

where $r(i)$ denotes the i^{th} binary character of label r, e.g. $r = 10$ and $r(1) = 1$. We reformulate the update rules defined in (2.6) and (2.8) 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

$$\begin{aligned} \boldsymbol{w}_{t+1,r} &= \boldsymbol{w}_{t,r} - \frac{1}{\beta}\boldsymbol{A}_t^{-1}\nabla e_t^2 \\ &= \boldsymbol{w}_{t,r} + \frac{2}{\beta}e_t\boldsymbol{A}_t^{-1}\frac{\partial \hat{y}_t}{\partial \boldsymbol{w}_{t,r}} \\ &= \boldsymbol{w}_{t,r} + \frac{2}{\beta}e_t\boldsymbol{A}_t^{-1}\sum_{r\in R}\frac{\partial\left(\hat{y}_{t,r}\prod_{i=1}^{K}\hat{p}_{t,P(i)}\right)}{\partial \boldsymbol{w}_{t,r}} \\ &= \boldsymbol{w}_{t,r} + \frac{2}{\beta}e_t\boldsymbol{A}_t^{-1}\boldsymbol{x}_t\prod_{i=1}^{K}\hat{p}_{t,P(i)} \end{aligned} \tag{2.13}$$

and the region boundaries are updated as

$$\begin{aligned} \boldsymbol{n}_{t+1,k} &= \boldsymbol{n}_{t,k} - \frac{1}{\eta}\boldsymbol{A}_t^{-1}\nabla e_t^2 \\ &= \boldsymbol{n}_{t,k} + \frac{2}{\eta}e_t\boldsymbol{A}_t^{-1}\frac{\partial \hat{y}_t}{\partial p_{t,k}}\frac{\partial p_{t,k}}{\partial \boldsymbol{n}_{t,k}} \\ &= \boldsymbol{n}_{t,k} + \frac{2}{\eta}e_t\boldsymbol{A}_t^{-1}\left[\sum_{r\in R}\frac{\partial \hat{\psi}_{t,r}}{\partial p_{t,k}}\right]\frac{\partial p_{t,k}}{\partial \boldsymbol{n}_{t,k}} \\ &= \boldsymbol{n}_{t,k} + \frac{2}{\eta}e_t\boldsymbol{A}_t^{-1}\left[\sum_{r\in R}\hat{y}_{t,r}\frac{\partial\left(\prod_{j=1}^{K}\hat{p}_{t,P(j)}\right)}{\partial p_{t,k}}\right]\frac{\partial p_{t,k}}{\partial \boldsymbol{n}_{t,k}} \\ &= \boldsymbol{n}_{t,k} + \frac{2}{\eta}e_t\boldsymbol{A}_t^{-1}\left[\sum_{r\in R}\hat{y}_{t,r}(-1)^{r(i)}\prod_{\substack{j=1 \\ j\neq i}}^{K}\hat{p}_{t,P(j)}\right]\frac{\partial p_{t,k}}{\partial \boldsymbol{n}_{t,k}} \end{aligned} \tag{2.14}$$

where we assign $k = P(i)$, i.e., separation function with label-k is the i^{th} entry of set P in the last equality. Since we use the logistic regression function, we can use the following equality to calculate the partial derivative of $p_{t,k}$ w.r.t. $\boldsymbol{n}_{t,k}$,

Algorithm 1 Straight Partitioning

$\boldsymbol{A}_0^{-1} = \frac{1}{\epsilon}\boldsymbol{I}_m$
for $t \leftarrow 1, n$ **do**
  $\hat{y}_t \leftarrow 0$
  for all $r \in R$ **do**
    $\hat{y}_{t,r} \leftarrow \boldsymbol{w}_{t,r}^T \boldsymbol{x}_t$
    $\hat{\psi}_{t,r} \leftarrow \hat{y}_{t,r}$
    $\nabla_{t,r} \leftarrow \boldsymbol{x}_t$
    for $i \leftarrow 1, K$ **do**
      if $r(i) := 0$ **then**
        $\hat{p}_{t,P(i)} \leftarrow p_{t,P(i)}$
      else
        $\hat{p}_{t,P(i)} \leftarrow 1 - p_{t,P(i)}$
      end if
      $\hat{\psi}_{t,r} \leftarrow \hat{\psi}_{t,r}\hat{p}_{t,P(i)}$
      $\nabla_{t,r} \leftarrow \nabla_{t,r}\hat{p}_{t,P(i)}$
    end for
    for $i \leftarrow 1, K$ **do**
      $\alpha_{t,P(i)} \leftarrow (-1)^{r(i)}(\hat{\psi}_{t,r}/\hat{p}_{t,P(i)})$
    end for
    $\hat{y}_t \leftarrow \hat{y}_t + \hat{\psi}_{t,r}$
  end for
  $e_t \leftarrow y_t - \hat{y}_t$
  for all $r \in R$ **do**
    $\nabla_{t,r} \leftarrow -2e_t\nabla_{t,r}$
    $\boldsymbol{A}_{t,r}^{-1} \leftarrow \boldsymbol{A}_{t-1,r}^{-1} - \dfrac{\boldsymbol{A}_{t-1}^{-1}\nabla_{t,r}\nabla_{t,r}^T\boldsymbol{A}_{t-1,r}^{-1}}{1 + \nabla_{t,r}^T\boldsymbol{A}_{t-1,r}^{-1}\nabla_{t,r}}$
    $\boldsymbol{w}_{t+1,r} \leftarrow \boldsymbol{w}_{t,r} - \frac{1}{\beta}\boldsymbol{A}_{t,r}^{-1}\nabla_{t,r}$
  end for
  for $i \leftarrow 1, K$ **do**
    $k \leftarrow P(i)$
    $\nabla_{t,k} \leftarrow -2e_t\alpha_{t,k}p_{t,k}(1 - p_{t,k})\boldsymbol{x_t}$
    $\boldsymbol{A}_{t,k}^{-1} \leftarrow \boldsymbol{A}_{t-1,k}^{-1} - \dfrac{\boldsymbol{A}_{t-1,k}^{-1}\nabla_{t,k}\nabla_{t,k}^T\boldsymbol{A}_{t-1,k}^{-1}}{1 + \nabla_{t,k}^T\boldsymbol{A}_{t-1,k}^{-1}\nabla_{t,k}}$
    $\boldsymbol{n}_{t+1,k} \leftarrow \boldsymbol{n}_{t,k} - \frac{1}{\eta}\boldsymbol{A}_{t,k}^{-1}\nabla_{t,k}$
  end for
end for

$$\begin{aligned}\frac{\partial p_{t,k}}{\partial \boldsymbol{n}_{t,k}} &= p_{t,k}(1-p_{t,k})\boldsymbol{x}_t \\ &= \frac{1}{(1+e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,k}})}\frac{e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,k}}}{(1+e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,k}})}\boldsymbol{x}_t \\ &= \frac{\boldsymbol{x}_t e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,k}}}{(1+e^{-\boldsymbol{x}_t^T\boldsymbol{n}_{t,k}})^2}.\end{aligned} \tag{2.15}$$

In order to avoid taking the inverse of an $m \times m$ matrix, $\boldsymbol{A}_t$, at each iteration in (2.13) and (2.14), we generate a recursive formula using matrix inversion lemma for $\boldsymbol{A}_t^{-1}$ given as [4]

$$\boldsymbol{A}_t^{-1} = \boldsymbol{A}_{t-1}^{-1} - \frac{\boldsymbol{A}_{t-1}^{-1}\nabla_t\nabla_t^T\boldsymbol{A}_{t-1}^{-1}}{1+\nabla_t^T\boldsymbol{A}_{t-1}^{-1}\nabla_t}, \tag{2.16}$$

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.

2.2.3 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, 22]. We follow the case given in Fig. 2.6. Here, we have three separation functions, $p_{t,\Omega}, p_{t,0}$ and $p_{t,1}$, partitioning the whole space into four subspaces. The corresponding direction vectors are given by $\boldsymbol{n}_{t,\Omega}, \boldsymbol{n}_{t,0}$ and $\boldsymbol{n}_{t,1}$ respectively. Using the individual estimates of all four regions, we find the final estimate by

$$\begin{aligned}\hat{y}_t = p_{t,\Omega}p_{t,0}\hat{y}_{t,00} + p_{t,\Omega}(1-p_{t,0})\hat{y}_{t,01} \\ + (1-p_{t,\Omega})p_{t,1}\hat{y}_{t,10} + (1-p_{t,\Omega})(1-p_{t,1})\hat{y}_{t,11}\end{aligned} \tag{2.17}$$

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. 2.7. The final estimate of this algorithm is given by the following generic formula

$$\hat{y}_t = \sum_{j=1}^{2^d} \hat{\psi}_{t,R_d(j)} \tag{2.18}$$

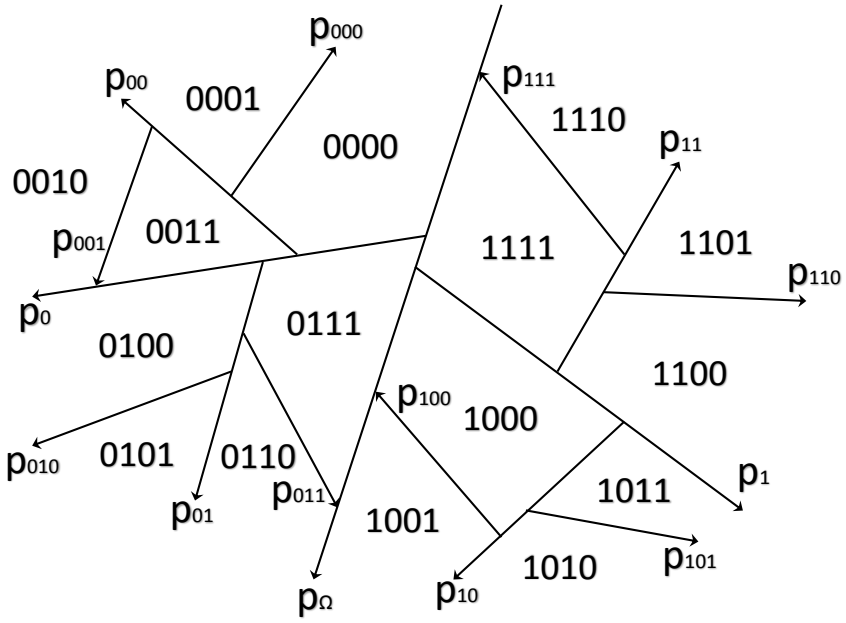

Figure 2.7: Labeling Example for the Depth-4 Case of the Finest Model

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 j^{th} entry of set R_d. Weighted estimate of each region is found as

$$\hat{\psi}_{t,r} = \hat{y}_{t,r} \prod_{i=1}^{d} \hat{p}_{t,r_i} \tag{2.19}$$

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

$$\hat{p}_{t,r_i} = \begin{cases} p_{t,r_i} & , r(i) = 0 \\ 1 - p_{t,r_i} & , r(i) = 1 \end{cases}. \tag{2.20}$$

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

$$\boldsymbol{w}_{t+1,r} = \boldsymbol{w}_{t,r} + \frac{2}{\beta} e_t \boldsymbol{A}_t \boldsymbol{x}_t \prod_{i=1}^{d} \hat{p}_{t,r_i} \tag{2.21}$$

Algorithm 2 Finest Model Partitioning (Part-1)

$A_0^{-1} \leftarrow \frac{1}{\epsilon}\boldsymbol{I}_m$
for $t \leftarrow 1, n$ **do**
  $\hat{y}_t \leftarrow 0$
  for $j \leftarrow 1, 2^d$ **do**
    $r \leftarrow R_d(j)$
    $\hat{y}_{t,r} \leftarrow \boldsymbol{w}_{t,r}^T \boldsymbol{x}_t$
    $\hat{\psi}_{t,r} \leftarrow \hat{y}_{t,r}$
    $\gamma_{t,r} \leftarrow 1$
    for $i \leftarrow 1, d$ **do**
      if $r(i) \leftarrow 0$ **then**
        $\hat{p}_{t,r_i} \leftarrow p_{t,r_i}$
      else
        $\hat{p}_{t,r_i} \leftarrow 1 - p_{t,r_i}$
      end if
      $\hat{\psi}_{t,r} \leftarrow \hat{\psi}_{t,r}\hat{p}_{t,r_i}$
      $\gamma_{t,r} \leftarrow \gamma_{t,r}\hat{p}_{t,r_i}$
    end for
    $\hat{y}_t \leftarrow \hat{y}_t + \hat{\psi}_{t,r}$
  end for
▷ Continues on the next page!

and the separator function updates are given by

$$\boldsymbol{n}_{t+1,k} = \boldsymbol{n}_{t,k} + \frac{2}{\eta} e_t \boldsymbol{A}_t^{-1} \left[\sum_{j=1}^{2^{d-\ell(k)}} \hat{y}_{t,r}(-1)^{r(\ell(k)+1)} \prod_{\substack{i=1 \\ r_i \neq k}}^{d} \hat{p}_{t,r_i} \right] \frac{\partial p_{t,k}}{\partial \boldsymbol{n}_{t,k}} \tag{2.22}$$

where r is the label string generated by concatenating separation function id k and the label kept in j^{th} 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.

Algorithm 3 Finest Model Partitioning (Part-2)

for $i \leftarrow 1, 2^d - 1$ **do**
  $k \leftarrow P(i)$
  for $j \leftarrow 1, 2^{d-\ell(k)}$ **do**
    $r \leftarrow concat[k : R_{d-\ell(k)}(j)]$
    $\alpha_{t,k} \leftarrow (-1)^{r(\ell(k)+1)}(\hat{\psi}_{t,r}/\hat{p}_{t,k})$
  end for
end for
$e_t \leftarrow y_t - \hat{y}_t$
for $j \leftarrow 1, 2^d$ **do**
  $r \leftarrow R_d(j)$
  $\nabla_{t,r} \leftarrow -2e_t\gamma_{t,r}\boldsymbol{x_t}$
  $\boldsymbol{A}_{t,r}^{-1} \leftarrow \boldsymbol{A}_{t-1,r}^{-1} - \dfrac{\boldsymbol{A}_{t-1}^{-1}\nabla_{t,r}\nabla_{t,r}^T\boldsymbol{A}_{t-1,r}^{-1}}{1+\nabla_{t,r}^T\boldsymbol{A}_{t-1,r}^{-1}\nabla_{t,r}}$
  $\boldsymbol{w}_{t+1,r} \leftarrow \boldsymbol{w}_{t,r} - \dfrac{1}{\beta}\boldsymbol{A}_{t,r}^{-1}\nabla_{t,r}$
end for
for $i \leftarrow 1, 2^d - 1$ **do**
  $k \leftarrow P(i)$
  $\nabla_{t,k} \leftarrow -2e_t\alpha_{t,k}p_{t,k}(1-p_{t,k})\boldsymbol{x_t}$
  $\boldsymbol{A}_{t,k}^{-1} \leftarrow \boldsymbol{A}_{t-1,k}^{-1} - \dfrac{\boldsymbol{A}_{t-1,k}^{-1}\nabla_{t,k}\nabla_{t,k}^T\boldsymbol{A}_{t-1,k}^{-1}}{1+\nabla_{t,k}^T\boldsymbol{A}_{t-1,k}^{-1}\nabla_{t,k}}$
  $\boldsymbol{n}_{t+1,k} \leftarrow \boldsymbol{n}_{t,k} - \dfrac{1}{\eta}\boldsymbol{A}_{t,k}^{-1}\nabla_{t,k}$
end for
end for
▷ End of the Algorithm

2.2.4 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. 2.8. We emphasize that the last partition is the finest model we use in the previous algorithm. Following the case in Fig. 2.8, we generate five distinct piecewise linear models and estimates of

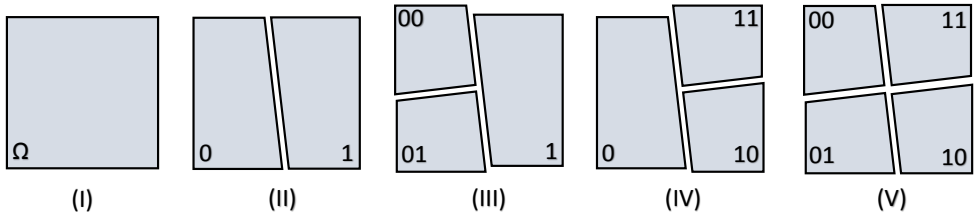

Figure 2.8: All Possible Models for the Depth-2 Tree Based Partitioning

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

$$\hat{y}_t = \boldsymbol{v}_t^T \hat{\boldsymbol{\phi}}_t \tag{2.23}$$

where $\hat{\boldsymbol{\phi}}_t = [\hat{\phi}_{t,1}, \hat{\phi}_{t,2}, ..., \hat{\phi}_{t,M}]^T$, $\boldsymbol{v}_t \in \mathbb{R}^M$ 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 2.2.3. Linear combination weights, $\boldsymbol{v}_t$, are also adaptively trained using the second order Newton-Raphson methods, i.e.

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t - \frac{1}{\mu}\boldsymbol{A}_t^{-1}\nabla e_t^2(\boldsymbol{v}_t) \tag{2.24}$$

$$= \boldsymbol{v}_t + \frac{2}{\mu}\boldsymbol{A}_t^{-1} e_t(\boldsymbol{v}_t)\boldsymbol{\phi}_t, \tag{2.25}$$

where μ and $\boldsymbol{A}_t^{-1}$ have the same definitions given in the paper, ∇ is the gradient operator w.r.t. the weight vector $\boldsymbol{v}_t$ and $e_t(\boldsymbol{v}_t)$ is defined as

$$e_t(\boldsymbol{v}_t) = y_t - \boldsymbol{v}_t^T \boldsymbol{\phi}_t. \tag{2.26}$$

2.2.5 Universality of the Presented Algorithm

In this thesis, we propose generic methods applicable to nonlinear regression. For the nonlinear modeling, we use piecewise linear models because of their ability

to sufficiently approximate nonlinear models while providing resilience to overfitting. However, it is also possible to use, for example, spline modeling within the presented tree based setting since the proposed approach is generic. This choice results in an algorithm, where a polynomial model is assigned for each region instead of a linear model. First, assume we use Type 1 partitioning, for the region $\{00\}$, a basic quadratic m-dimensional spline model is defined as

$$\hat{y}_{t,00} = \boldsymbol{x}_t^T \boldsymbol{w}_{t,00}^{(1)} + (\boldsymbol{x}_t^T \circ \boldsymbol{x}_t^T)\boldsymbol{w}_{t,00}^{(2)}, \tag{2.27}$$

where $\boldsymbol{x}_t \in \mathbb{R}^m$ is the observation vector, $\boldsymbol{w}_{t,00}^{(1)}, \boldsymbol{w}_{t,00}^{(2)} \in \mathbb{R}^m$ are the weight vectors of region $\{00\}$ corresponding to linear and quadratic terms respectively and $\circ$ denotes the Hadamard product. We ignore the offset term just for the sake of notational simplicity. The final estimate is the same as that is given previously, i.e.,

$$\begin{aligned} \hat{y}_t = {} & p_{t,0}p_{t,1}\hat{y}_{t,00} + p_{t,0}(1-p_{t,1})\hat{y}_{t,01} + (1-p_{t,0})p_{t,1}\hat{y}_{t,10} \\ & + (1-p_{t,0})(1-p_{t,1})\hat{y}_{t,11} \end{aligned} \tag{2.28}$$

In these settings, the weight vectors can be updated using the same adaptation algorithm that uses straight partitioning method with minor differences. We update the weight vectors separately, i.e., the vector $\boldsymbol{w}_{t,00}^{(1)}$ is updated as

$$\begin{aligned} \boldsymbol{w}_{t+1,00}^{(1)} &= \boldsymbol{w}_{t,00}^{(1)} - \frac{1}{\mu}\boldsymbol{A}_t^{-1}\nabla e_t^2(\boldsymbol{w}_{t,00}^{(1)}, \boldsymbol{w}_{t,00}^{(2)}) \\ &= \boldsymbol{w}_{t,00}^{(1)} + \frac{2}{\mu}\boldsymbol{A}_t^{-1}p_{t,0}p_{t,1}\boldsymbol{x}_t e_t(\boldsymbol{w}_{t,00}^{(1)}, \boldsymbol{w}_{t,00}^{(2)}), \end{aligned} \tag{2.29}$$

where ∇ is w.r.t. the $\boldsymbol{w}_{t,00}^{(1)}$ and $\boldsymbol{A}_t^{-1}$ has the same definition given in the original manuscript. Similarly, the vector $\boldsymbol{w}_{t,00}^{(2)}$ is updated as

$$\begin{aligned} \boldsymbol{w}_{t+1,00}^{(2)} &= \boldsymbol{w}_{t,00}^{(2)} - \frac{1}{\mu}\boldsymbol{A}_t^{-1}\nabla e_t^2(\boldsymbol{w}_{t,00}^{(1)}, \boldsymbol{w}_{t,00}^{(2)}) \\ &= \boldsymbol{w}_{t,00}^{(2)} + \frac{2}{\mu}\boldsymbol{A}_t^{-1}p_{t,0}p_{t,1}(\boldsymbol{x}_t \circ \boldsymbol{x}_t)e_t(\boldsymbol{w}_{t,00}^{(1)}, \boldsymbol{w}_{t,00}^{(2)}), \end{aligned} \tag{2.30}$$

where ∇ is w.r.t. the $\boldsymbol{w}_{t,00}^{(2)}$. Consequently, we demonstrate that the update relation of the weight vectors corresponding to the first order terms are not affected by the selection of spline modeling. We only need to perform a secondary update operation since there exist weights corresponding to the second order terms.

Table 2.1: 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	$O(m2^d)$	$O(md)$	$O(m^n n^n)$	$O(m^n)$	$O(m^n)$

However, as shown in (2.30), there is only a minor change from $\boldsymbol{x}_t$ to $(\boldsymbol{x}_t \circ \boldsymbol{x}_t)$ for this update rule. As a final comment, the proposed tree based approach can be implemented by using any other function that is differentiable without a significant difference in the algorithm, hence, it is universal in terms of the possible selection of functions.

2.2.6 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 [28]. 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 form 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 2.1, 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 [28]. "S-DAT" denotes the Decision Adaptive Tree with second order update rules. "CTW" is used for Context Tree Weighting [23], "GKR" represents Gaussian-Kernel regressor [41], "VF" represents Volterra Filter [42], "FNF" and "EMFNF" stand for the Fourier and Even Mirror Fourier Nonlinear Filter [43] respectively.

2.2.7 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 [40], 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.,

$$\boldsymbol{w}_n^* = \arg\min_{\boldsymbol{w}} \sum_{t=1}^{n} e_t^2(\boldsymbol{w}) \tag{2.31}$$

and express the following inequality

$$e_t^2(\boldsymbol{w}_t) - e_t^2(\boldsymbol{w}_n^*) \leq \nabla_t^T(\boldsymbol{w}_t - \boldsymbol{w}_n^*) - \frac{\beta}{2}(\boldsymbol{w}_t - \boldsymbol{w}_n^*)^T \nabla_t \nabla_t^T (\boldsymbol{w}_t - \boldsymbol{w}_n^*) \tag{2.32}$$

using the Lemma 3 of [40], 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., $\|\nabla_t\| \leq G$. We give the update rule for regressor weights as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{1}{\beta}\boldsymbol{A}_t^{-1}\nabla_t. \tag{2.33}$$

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

$$\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^* = \boldsymbol{w}_t - \boldsymbol{w}_n^* - \frac{1}{\beta}\boldsymbol{A}_t^{-1}\nabla_t \tag{2.34}$$

$$\boldsymbol{A}_t(\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*) = \boldsymbol{A}_t(\boldsymbol{w}_t - \boldsymbol{w}_n^*) - \frac{1}{\beta}\nabla_t \tag{2.35}$$

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

$$\begin{aligned} \nabla_t(\boldsymbol{w}_t - \boldsymbol{w}_n^*) = \frac{1}{2\beta}\nabla_t^T \boldsymbol{A}_t^{-1}\nabla_t + \frac{\beta}{2}(\boldsymbol{w}_t - \boldsymbol{w}_n^*)^T \boldsymbol{A}_t(\boldsymbol{w}_t - \boldsymbol{w}_n^*) \\ - \frac{\beta}{2}(\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*)^T \boldsymbol{A}_t(\boldsymbol{w}_{t+1} - \boldsymbol{w}_n^*). \end{aligned} \tag{2.36}$$

Then, summing up all T iteration gives,

$$\begin{aligned}\sum_{t=1}^{n}\nabla_t(\boldsymbol{w}_t-\boldsymbol{w}_n^*) &= \frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t \\ &+\frac{\beta}{2}\sum_{t=1}^{n}(\boldsymbol{w}_t-\boldsymbol{w}_n^*)^T(\boldsymbol{A}_t-\boldsymbol{A}_{t-1})(\boldsymbol{w}_t-\boldsymbol{w}_n^*) \\ &+\frac{\beta}{2}(\boldsymbol{w}_1-\boldsymbol{w}_n^*)^T\boldsymbol{A}_0(\boldsymbol{w}_1-\boldsymbol{w}_n^*) \\ &-\frac{\beta}{2}(\boldsymbol{w}_{T+1}-\boldsymbol{w}_n^*)^T\boldsymbol{A}_T(\boldsymbol{w}_{T+1}-\boldsymbol{w}_n^*) \\ &\leq \frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t \\ &+\frac{\beta}{2}\sum_{t=1}^{n}(\boldsymbol{w}_t-\boldsymbol{w}_n^*)^T(\boldsymbol{A}_t-\boldsymbol{A}_{t-1})(\boldsymbol{w}_t-\boldsymbol{w}_n^*) \\ &+\frac{\beta}{2}(\boldsymbol{w}_1-\boldsymbol{w}_n^*)^T\boldsymbol{A}_0(\boldsymbol{w}_1-\boldsymbol{w}_n^*)\end{aligned} \tag{2.37}$$

where the inequality is satisfied by removing the term $\frac{\beta}{2}(\boldsymbol{w}_{T+1}-\boldsymbol{w}_n^*)^T\boldsymbol{A}_T(\boldsymbol{w}_{T+1}-\boldsymbol{w}_n^*)$, which is indeed positive, since $\boldsymbol{A}_T$ is guaranteed to be positive definite. We transfer the term $\frac{\beta}{2}\sum_{t=1}^{n}(\boldsymbol{w}_t-\boldsymbol{w}_n^*)^T(\boldsymbol{A}_t-\boldsymbol{A}_{t-1})(\boldsymbol{w}_t-\boldsymbol{w}_n^*)$ to the left hand side to get the right hand side of inequality (10). Thus, we have

$$\sum_{t=1}^{n}S_t \leq \frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t+\frac{\beta}{2}(\boldsymbol{w}_1-\boldsymbol{w}_n^*)^T\boldsymbol{A}_0(\boldsymbol{w}_1-\boldsymbol{w}_n^*), \tag{2.38}$$

where S_t is defined as

$$S_t \triangleq \nabla_t^T(\boldsymbol{w}_t-\boldsymbol{w}_n^*)-\frac{\beta}{2}(\boldsymbol{w}_t-\boldsymbol{w}_n^*)^T\nabla_t\nabla_t^T(\boldsymbol{w}_t-\boldsymbol{w}_n^*). \tag{2.39}$$

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

$$\begin{aligned}\sum_{t=1}^{n}e_t^2(\boldsymbol{w}_t)-\sum_{t=1}^{n}e_t^2(\boldsymbol{w}_n^*) &\leq \frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t+\frac{\beta}{2}\epsilon\delta^2 \\ &\leq \frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T\boldsymbol{A}_t^{-1}\nabla_t+\frac{1}{2\beta},\end{aligned} \tag{2.40}$$

where we choose $\epsilon=\frac{1}{\beta^2A^2}$ 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 [40], to get the following bound

$$\frac{1}{2\beta}\sum_{t=1}^{n}\nabla_t^T \boldsymbol{A}_t^{-1}\nabla_t \leq \frac{m}{2\beta}\log\left(\frac{G^2 n}{\epsilon}+1\right) = \frac{m}{2\beta}\log(G^2 n\beta^2 A^2+1) \leq \frac{m}{2\beta}\log(n), \tag{2.41}$$

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 $\frac{1}{\beta} \leq 8(GA+\frac{1}{\alpha})$. Therefore, we present the final logarithmic regret bound as

$$\sum_{t=1}^{n} e_t^2(\boldsymbol{w}_t) - \sum_{t=1}^{n} e_t^2(\boldsymbol{w}_n^*) \leq 5\left(GA+\frac{1}{\alpha}\right)m\log(n). \tag{2.42}$$

2.3 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 [44], Kinematics [44] and Elevators [44]. 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.

2.3.1 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

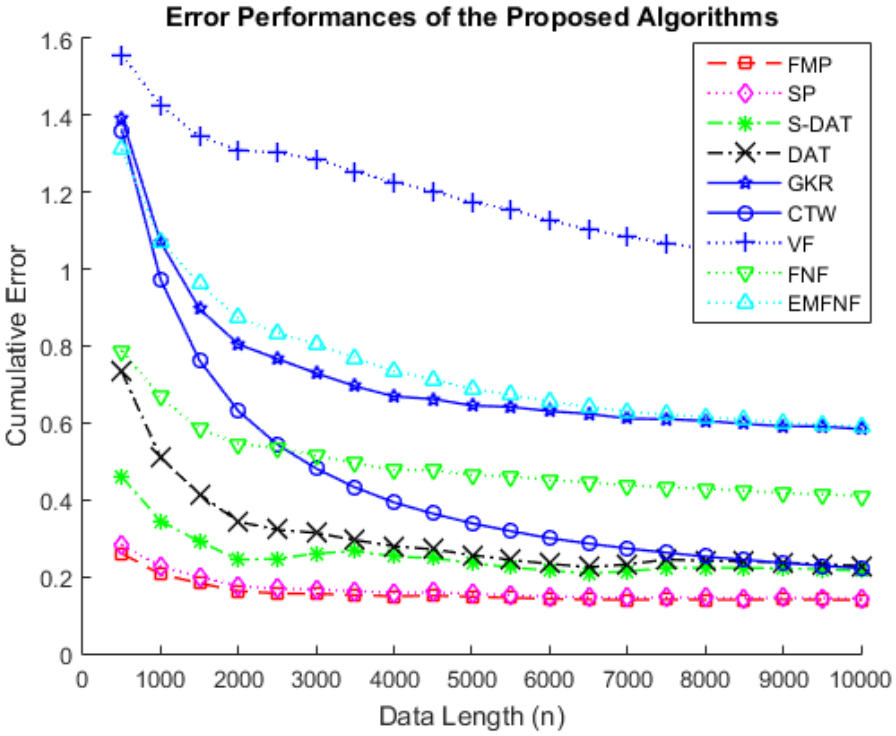

Figure 2.9: Regression error performances for the matched partitioning case using model (2.43).

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,

$$\hat{y}_t = \begin{cases} \boldsymbol{w}_1^T \boldsymbol{x}_t + \upsilon_t & , \boldsymbol{x}_t^T \boldsymbol{n}_0 \geq 0 \text{ and } \boldsymbol{x}_t^T \boldsymbol{n}_1 \geq 0 \\ \boldsymbol{w}_2^T \boldsymbol{x}_t + \upsilon_t & , \boldsymbol{x}_t^T \boldsymbol{n}_0 \geq 0 \text{ and } \boldsymbol{x}_t^T \boldsymbol{n}_1 < 0 \\ \boldsymbol{w}_2^T \boldsymbol{x}_t + \upsilon_t & , \boldsymbol{x}_t^T \boldsymbol{n}_0 < 0 \text{ and } \boldsymbol{x}_t^T \boldsymbol{n}_1 \geq 0 \\ \boldsymbol{w}_1^T \boldsymbol{x}_t + \upsilon_t & , \boldsymbol{x}_t^T \boldsymbol{n}_0 < 0 \text{ and } \boldsymbol{x}_t^T \boldsymbol{n}_1 < 0 \end{cases} \tag{2.43}$$

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. 2.9, 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. 2.9, 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.

2.3.2 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,

$$\hat{y}_t = \begin{cases} \boldsymbol{w}_1^T \boldsymbol{x}_t + \upsilon_t & , \boldsymbol{x}_t^T \boldsymbol{n}_0 \geq 0.5 \text{ and } \boldsymbol{x}_t^T \boldsymbol{n}_1 \geq -0.5 \\ \boldsymbol{w}_2^T \boldsymbol{x}_t + \upsilon_t & , \boldsymbol{x}_t^T \boldsymbol{n}_0 \geq 0.5 \text{ and } \boldsymbol{x}_t^T \boldsymbol{n}_1 < -0.5 \\ \boldsymbol{w}_2^T \boldsymbol{x}_t + \upsilon_t & , \boldsymbol{x}_t^T \boldsymbol{n}_0 < 0.5 \text{ and } \boldsymbol{x}_t^T \boldsymbol{n}_2 \geq -0.5 \\ \boldsymbol{w}_1^T \boldsymbol{x}_t + \upsilon_t & , \boldsymbol{x}_t^T \boldsymbol{n}_0 < 0.5 \text{ and } \boldsymbol{x}_t^T \boldsymbol{n}_2 < -0.5 \end{cases} \tag{2.44}$$

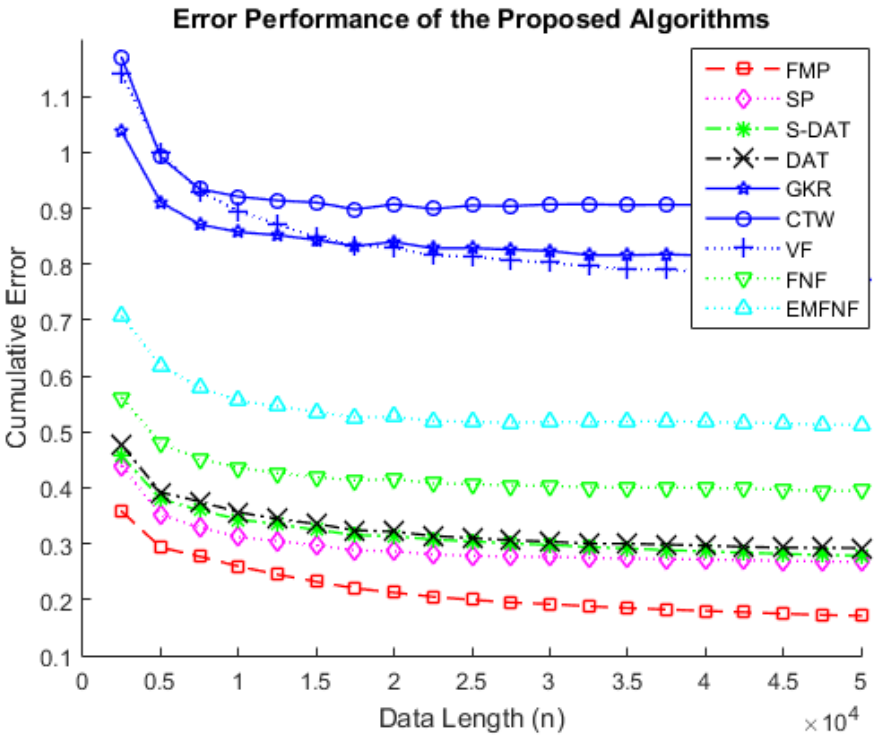

Figure 2.10: Regression error performances for the mismatched partitioning case using model (2.44).

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. 2.10, 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

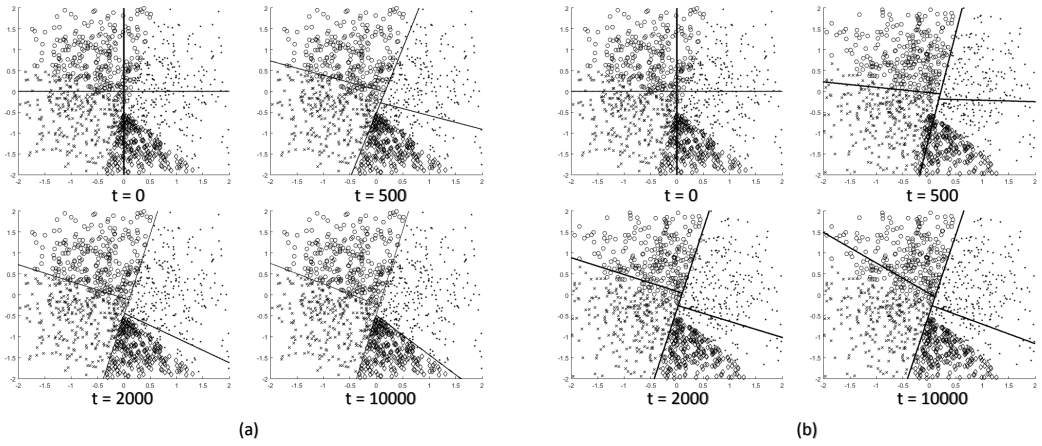

Figure 2.11: Training of the separation functions for the mismatched partitioning scenario (a) FMP Algorithm (b) DAT Algorithm.

update their partitions in accordance with the data distribution. Comparing these four algorithms, Fig. 2.10 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 (2.44).

We illustrate how the FMP and the DAT algorithms update their region boundaries in Fig. 2.11. 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 10000, i.e., $t = 10000$, 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 $t = 10000$. 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.

2.3.3 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

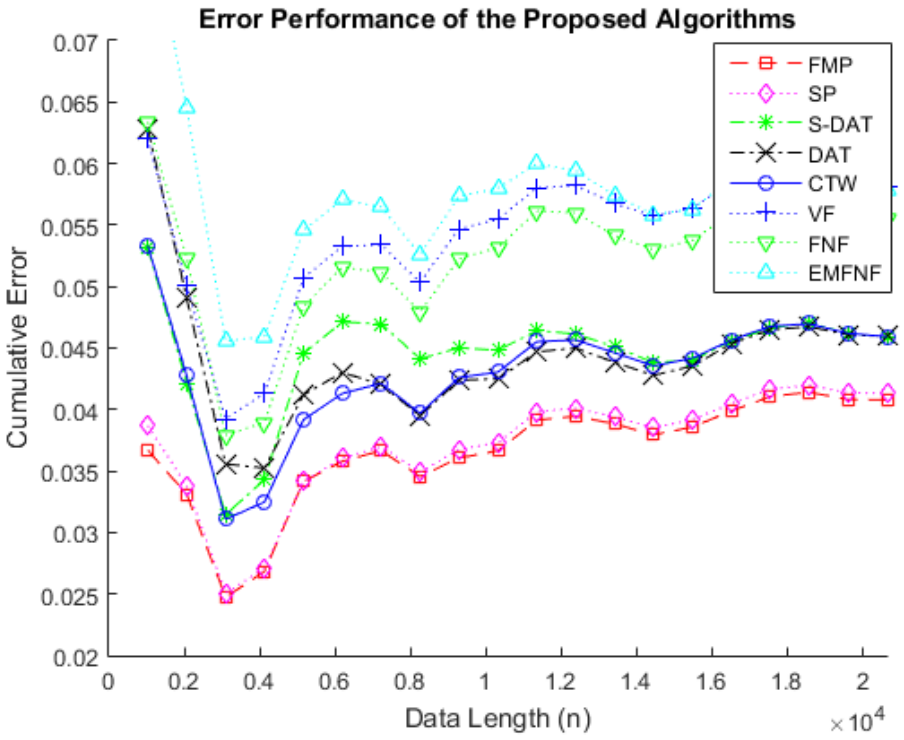

Figure 2.12: Time accumulated error performances of the proposed algorithms for California Housing Data Set.

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 [44]. There exist more than 20000 data samples for this dataset. For this experiment, we set the learning rates 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. 2.12 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 [44]. 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 [44]. This dataset provides more than

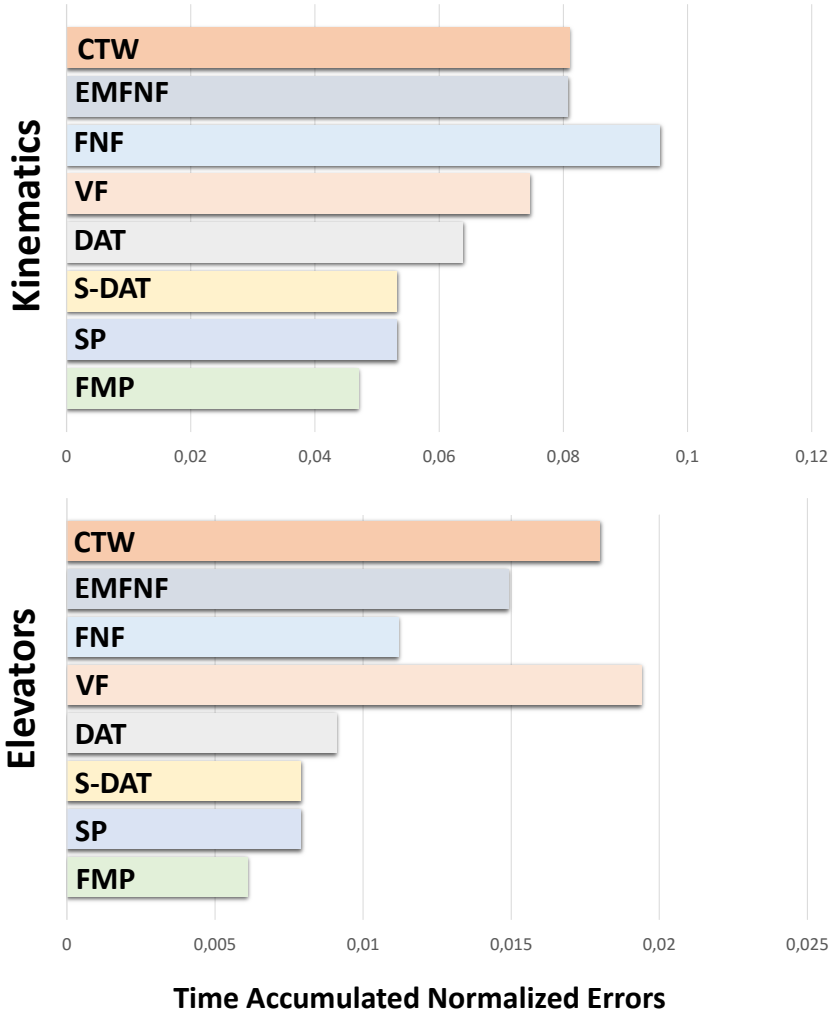

Figure 2.13: Time accumulated error performances of the proposed algorithms for Kinematics and Elevators Data Set.

50000 samples. In Fig. 2.13, 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

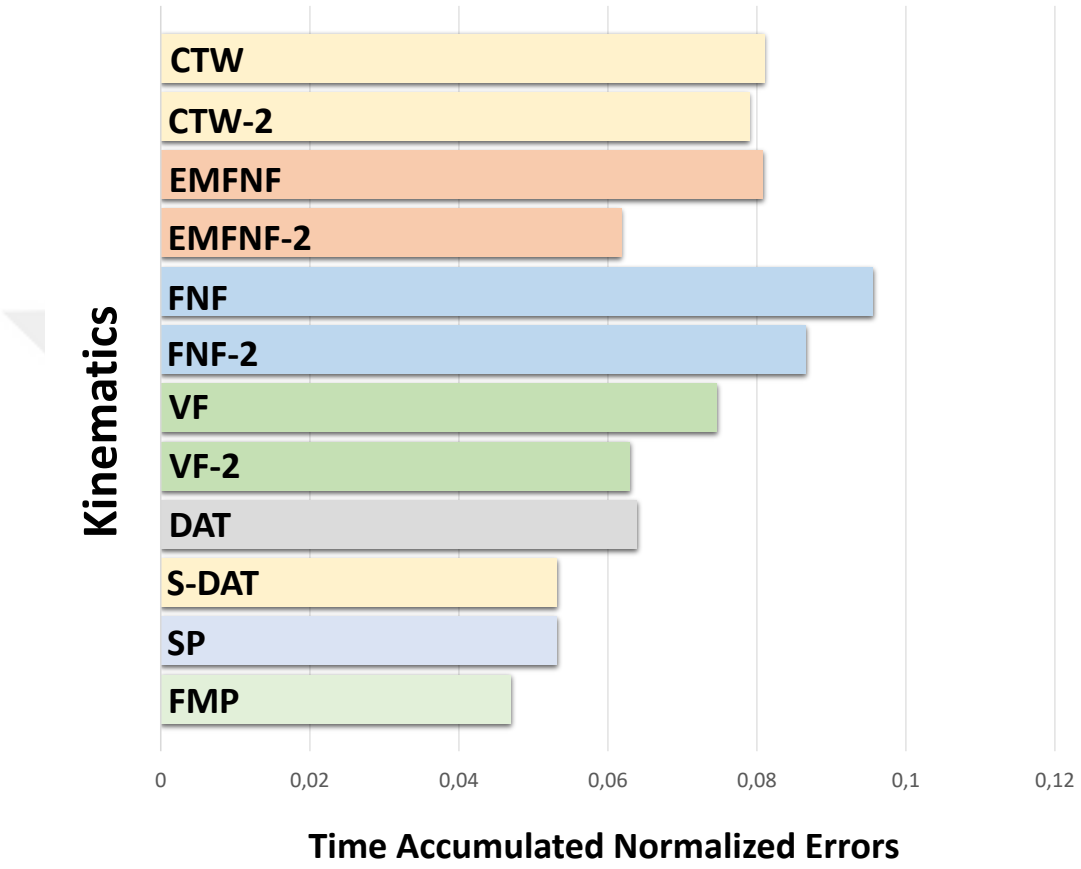

Figure 2.14: Time accumulated error performances of the proposed algorithms for Kinematics Data Set. The second order adaptation methods are used for all algorithms.

computational demand. Here, we modified the adaptation part of these algorithms and use the second order Newton-Raphson methods instead. In Fig. 2.14, 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. 2.14, the CTW-2, the EMFNF-2, 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

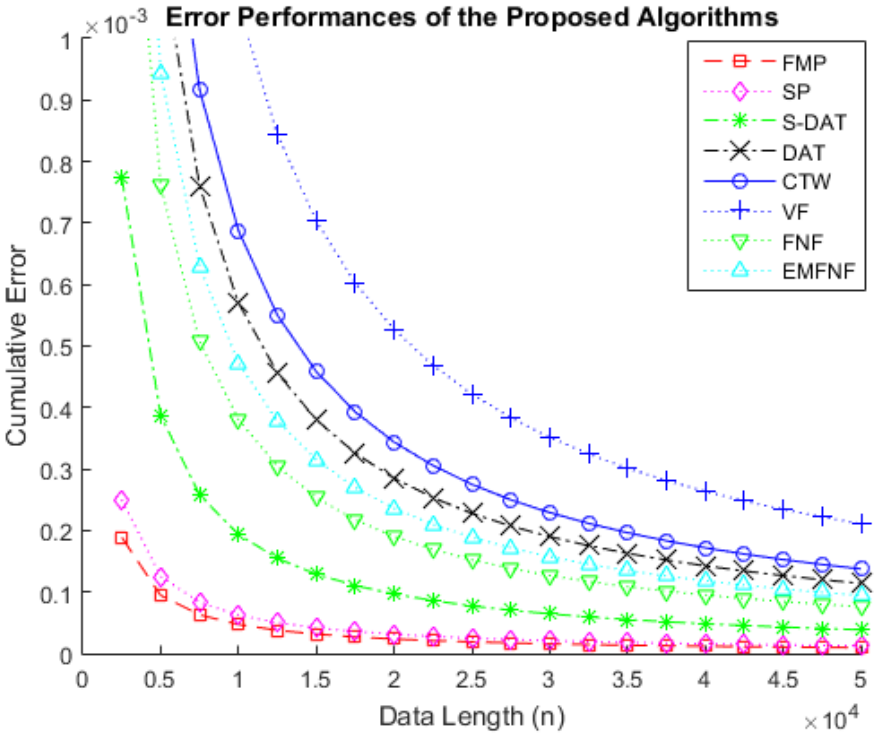

Figure 2.15: Regression Error Rates for the Gauss map.

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.

2.3.4 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.,

$$y_t = \exp\left(-\alpha x_t^2\right) + \beta \tag{2.45}$$

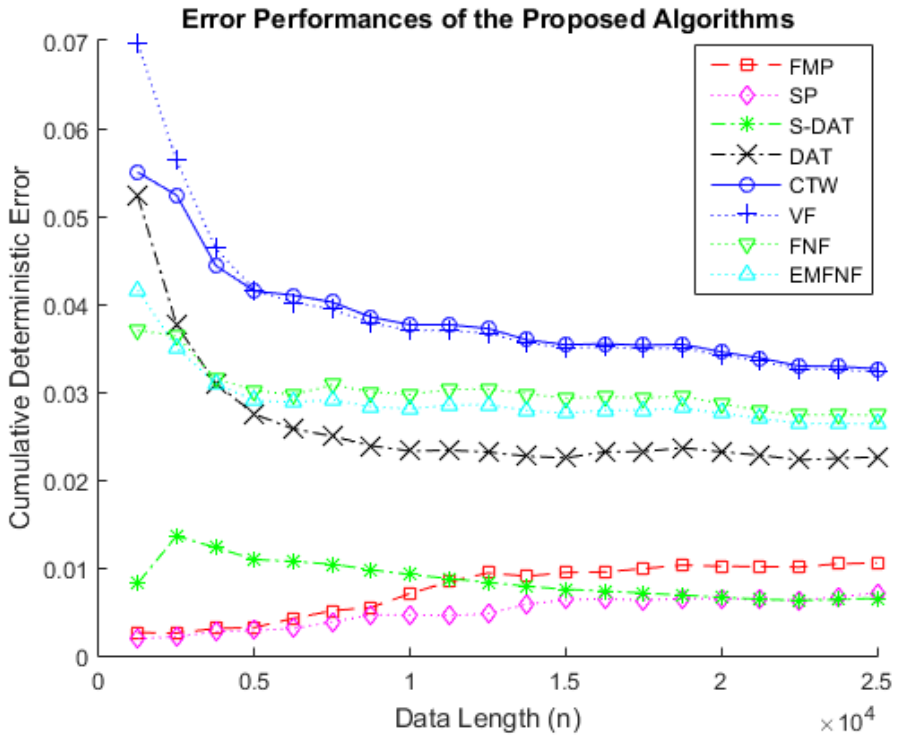

Figure 2.16: Regression Error Rates for the Lorenz attractor.

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

$$y_t = y_{t-1} + (\sigma(u_{t-1} - y_{t-1}))dt \tag{2.46}$$

$$u_t = u_{t-1} + (y_{t-1}(\rho - v_{t-1}) - u_{t-1})dt \tag{2.47}$$

$$v_t = v_{t-1} + (y_{t-1}u_{t-1} - \beta v_{t-1})dt, \tag{2.48}$$

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 Fig. 2.15 and 2.16, 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.

Chapter 3

Efficient Implementation Of Newton-Raphson Methods For Sequential Prediction

There exists a significant amount of data flow through the recently arising applications such as large-scale sensor networks, information sensing mobile devices and web based social networks [3, 17, 45, 46]. The size as well as the dimensionality of this data strain the limits of current architectures. Since processing and storing such massive amount of data results in an excessive computational cost, efficient machine learning and signal processing algorithms are needed [6, 47].

In this chapter, we investigate the sequential prediction problem, which is a specific case of sequential regression where the consecutive feature vectors are the shifted versions of each other, for high dimensional data streams. In order to mitigate the problem of excessive computational cost, we introduce sequential, i.e., online, processing, where the data is neither stored nor reused, and avoid "batch" processing. [34, 48]. One family of the well known online learning algorithms in the machine learning literature is the family of first order methods, e.g., Online Gradient Descent [36, 40]. These methods only use the gradient information to minimize the overall prediction cost. They achieve logarithmic regret bounds,

which are theoretically guaranteed to hold under certain assumptions [40]. Gradient based methods are computationally more efficient and feasible compared to other families of online learning algorithms, i.e., for a sequence of M-dimensional feature vectors $\{\boldsymbol{x}_t\}_{t\geq 0}$, where $\boldsymbol{x}_t \in \mathbb{R}^M$, the computational complexity is only in the order of $O(M)$. However, their convergence rate remains significantly slow when achieving an optimal solution, since no statistics other than the gradient is used [16, 34, 36]. Hence, it requires much more iteration for the first order algorithms to reach the steady state, compared to more complicated learning methods. Even though gradient based methods suffer from this convergence issue, they are extensively used in big data applications due to such low computational demand [49].

Different from the gradient based algorithms, the well known second order Newton-Raphson methods, e.g, Online Newton Step, use the second order statistics, i.e., Hessian of the feature vector, in each iteration to obtain the optimal solution [40]. Hence, they asymptotically achieve the performance of the "best" possible predictor much faster [48]. Existence of logarithmic regret bounds are theoretically guaranteed for this family of algorithms as well [40]. Additionally, the second order methods are robust and prone to highly varying data statistics, compared to the first order methods, since they keep track of the second order information [48, 50]. Therefore, in the sense of convergence rate and steady state error performances, Newton-Raphson methods considerably outperform the first order methods [34,36,38,48]. However, the second order methods offer a quadratic increase in the computational complexity, i.e., $O(M^2)$, while the gradient based algorithms provide a linear relation, i.e., $O(M)$. Hence, the computation time required to perform a single iteration is $O(M)$ times longer for the Newton-Raphson methods. As a consequence, it is not usually feasible for real-life big data applications to utilize the merits of the second order algorithms [49].

Overall, in this chapter, we introduce an online sequential prediction algorithm that *i*) process only the currently available data without any storage, *ii*) efficiently implements the Newton-Raphson methods, i.e., the second order methods *iii*) outperforms the gradient based methods in terms of performance, *iv*) has $O(M)$ computational complexity same as the first order methods and *v*) requires no

statistical assumptions on the data sequence. We illustrate the outstanding gains of our algorithm in terms of computational efficiency using several sequential real life and synthetic big datasets and compare the resulting error performances with the regular Newton-Raphson methods.

This chapter is organized as follow. We first provide the problem description in Section 3.1. We then present an highly efficient implementation for the second order methods in Section 3.2. We then demonstrate the performance of our algorithm through simulations based on two different real life big datasets in Section 3.3.

3.1 Problem Description

We study sequential prediction, where we sequentially observe a real valued data sequence $\{x_t\}_{t\geq 0}$, $x_t \in \mathbb{R}$. At each time t, after observing $\{x_t, x_{t-1}, ..., x_{t-M+1}\}$, we generate an estimate of the desired data, $\hat{x}_{t+1} \in \mathbb{R}$, using a linear model as

$$\hat{x}_{t+1} = \boldsymbol{w}_t^T \boldsymbol{x}_t + c_t, \tag{3.1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^M$ represents the feature vector of previous M samples, i.e., $\boldsymbol{x}_t = [x_t, x_{t-1}, ..., x_{t-M+1}]^T$. Here, $\boldsymbol{w}_t \in \mathbb{R}^M$ and $c_t \in \mathbb{R}$ are the corresponding weight vector and the offset variable respectively at time t. With an abuse of notation, we combine the weight vector $\boldsymbol{w}_t$ with the offset variable c_t, and denote it by $\boldsymbol{w}_t = [\boldsymbol{w}_t; c_t]$, yielding

$$\hat{x}_{t+1} = \boldsymbol{w}_t^T \boldsymbol{x}_t, \tag{3.2}$$

where $\boldsymbol{x}_t = [\boldsymbol{x}_t; 1]$. As the performance criterion, we use the widely studied instantaneous absolute loss as our cost function, i.e., $\ell_t(\boldsymbol{w}_t) = \|e_t\|$, where the prediction error at each time instant is given by

$$e_t = x_t - \hat{x}_t.$$

We adaptively learn the weight vector coefficients to asymptotically achieve the best possible fixed weight vector $\hat{\boldsymbol{w}}_n$, which minimizes the total prediction

error after n iteration, i.e.,

$$\hat{\boldsymbol{w}}_n = \arg\min_{\boldsymbol{w}\in\mathbb{R}^M} \sum_{t=0}^{n} \|x_{t+1} - \boldsymbol{w}^T \boldsymbol{x}_t\|,$$

for any n. The definition of $\hat{\boldsymbol{w}}_n$ is given for the absolute loss case. To this end, we use the second order Online Newton Step (ONS) algorithm to train the weight vectors. The ONS algorithm significantly outperforms the first order Online Gradient Descent (OGD) algorithm in terms of convergence rate and steady state error performance since it keeps track of the second order statistics of the data sequence [34, 36, 40]. The weight vector at each time is updated as

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{1}{\mu} \boldsymbol{A}_t^{-1} \nabla_t, \tag{3.3}$$

where $\mu \in \mathbb{R}$ is the step size and $\nabla_t \in \mathbb{R}^M$ corresponds to the gradient of the cost function $\ell_t(\boldsymbol{w}_t)$ at time t w.r.t. $\boldsymbol{w}_t$, i.e., $\nabla_t \triangleq \nabla \ell_t(\boldsymbol{w}_t)$. Here, the $M \times M$ dimensional matrix $\boldsymbol{A}_t$ is given by

$$\boldsymbol{A}_t = \sum_{i=0}^{t} \nabla_i \nabla_i^T + \alpha \boldsymbol{I}_M, \tag{3.4}$$

where $\alpha > 0$ is chosen to guarantee that $\boldsymbol{A}_t$ is positive definite, i.e., $\boldsymbol{A}_t > 0$, and hence, invertible. Selection of the parameters μ and α is crucial for good performance [40]. Note that for the first order OGD algorithm, we have $\boldsymbol{A}_t = \boldsymbol{I}_M$ for all t, i.e., we do not use the second order statistics but only the gradient information.

The requirement of the matrix inversion operation results in a computational complexity in the order of $O(M^3)$ for an $M \times M$ dimensional matrix [34]. Hence, direct use of (3.4) and performing an inversion operation at each iteration causes a bottleneck for big data applications with high dimensional feature vectors. However, note that (3.4) has a recursive structure, where, at each time t, we update the matrix $\boldsymbol{A}_t$ using the relation

$$\boldsymbol{A}_t = \boldsymbol{A}_{t-1} + \nabla_t \nabla_t^T, \tag{3.5}$$

with an initial value of $\boldsymbol{A}_{-1} = \alpha \boldsymbol{I}_M$. We then get a straight update from $\boldsymbol{A}_{t-1}^{-1}$ to $\boldsymbol{A}_t^{-1}$ using the matrix inversion lemma [4]

$$\boldsymbol{A}_t^{-1} = \boldsymbol{A}_{t-1}^{-1} - \frac{\boldsymbol{A}_{t-1}^{-1} \nabla_t \nabla_t^T \boldsymbol{A}_{t-1}^{-1}}{1 + \nabla_t^T \boldsymbol{A}_{t-1}^{-1} \nabla_t}. \tag{3.6}$$

Then, the computational complexity for computing the matrix $\boldsymbol{A}_t^{-1}$ is reduced to the order of $O(M^2)$. Multiplying both sides of (3.6) with ∇_t yields

$$\boldsymbol{A}_t^{-1}\nabla_t = \frac{\boldsymbol{A}_{t-1}^{-1}\nabla_t}{1+\nabla_t^T\boldsymbol{A}_{t-1}^{-1}\nabla_t}. \tag{3.7}$$

Using (3.7) in (3.3) yields

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \frac{1}{\mu}\left[\frac{\boldsymbol{A}_{t-1}^{-1}\nabla_t}{1+\nabla_t^T\boldsymbol{A}_{t-1}^{-1}\nabla_t}\right]. \tag{3.8}$$

Although the second order update algorithms provide faster convergence rates and better steady state performances, computational complexity issue prohibits their usage in most real life applications [4,36,38]. Since each update in (3.6) requires the multiplication of an $M \times M$ dimensional matrix with an M dimensional vector for $\boldsymbol{x}_t \in \mathbb{R}^M$, the computational complexity is in the order of $O(M^2)$, while the first order algorithms just need $O(M)$ multiplication and addition. As an example, in protein structure prediction, we have $M = 1000$ deeming the second order methods 1000 times slower than the first order OGD algorithm [51]. Hence, when the data dimension grows, the computation time required to perform each iteration increases dramatically.

In the next section, we introduce a sequential prediction algorithm, which achieves the performance of the Newton-Raphson methods, while offering $O(M)$ computational complexity same as the first order methods. To generate the highly efficient algorithm, we utilize the sequential nature of the feature vectors $\boldsymbol{x}_t$, i.e., only two entries change from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$. We aim to eliminate the unnecessary calculations without losing any information.

3.2 Efficient Implementation for Complexity Reduction

In this section, we construct an efficient implementation that is based on the low rank property of the update matrices. Instead of directly implementing the

second order methods as in (3.6) and (3.8), we use unitary and hyperbolic transformations to update the weight vector $\boldsymbol{w}_t$ and the inverse of the Hessian-related matrix $\boldsymbol{A}_t^{-1}$.

We work on time series data sequences, which directly implies that the feature vectors $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$ are highly related. More precisely, we have the following relation between these two consecutive vectors as

$$[x_{t+1}, \boldsymbol{x}_t^T] = [\boldsymbol{x}_{t+1}^T, x_{t-M+1}]. \tag{3.9}$$

This relation shows that consecutive data vectors carry quite the same information, which is the basis of our algorithm. We use the instantaneous absolute loss, which is defined as

$$\ell_t(\boldsymbol{w}_t) = \|\hat{x}_{t+1} - \boldsymbol{w}_t^T \boldsymbol{x}_t\|. \tag{3.10}$$

Although the absolute loss is widely used in the machine learning applications, it is not differentiable when $e_t = 0$. However, we resolve this issue by setting a threshold ϵ close to zero and not updating the weight vector when the absolute error is below this threshold, $\|e_t\| < \epsilon$. From (3.6) and (3.8), the absolute loss results in the following update rules for $\boldsymbol{w}_t$ and $\boldsymbol{A}_t^{-1}$,

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} \pm \frac{1}{\mu}\left[\frac{\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t}{1 + \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t}\right], \tag{3.11}$$

$$\boldsymbol{A}_t^{-1} = \boldsymbol{A}_{t-1}^{-1} - \frac{\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t\boldsymbol{x}_t^T\boldsymbol{A}_{t-1}^{-1}}{1 + \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t}, \tag{3.12}$$

since $\nabla_t = \pm\boldsymbol{x}_t$ depending on the sign of the error.

3.2.1 Givens and Hyperbolic Transformations

The proposed algorithm requires unitary and hyperbolic transformations and we briefly give the definitions of Givens and Hyperbolic rotations in this subsection. These transformations are used to zero out the specific elements of a vector. We introduce the definitions for two dimensional vectors, which is sufficient for our case.

3.2.1.1 Givens Rotation

This rotation is called as a unitary transformation, which implies that the $N \times N$ transformation matrix $\boldsymbol{H}_G$ satisfies the equality $\boldsymbol{H}_G \boldsymbol{H}_G^T = \boldsymbol{H}_G^T \boldsymbol{H}_G = \boldsymbol{I}_N$, i.e., $\boldsymbol{H}_G$ is orthogonal [52]. For the 2-dimensional vector $\boldsymbol{v} = [v_1, v_2]^T$, we seek for an orthogonal 2×2 dimensional matrix $\boldsymbol{H}_G$ such that the transformed vector $\tilde{\boldsymbol{v}}$, which is given as $\tilde{\boldsymbol{v}}^T = \boldsymbol{v}^T \boldsymbol{H}_G$ has the form $[v_3, 0]^T$, where the norm is preserved, i.e., $v_3^2 = v_1^2 + v_2^2$. Such a transformation matrix $\boldsymbol{H}_G$ is given as

$$\boldsymbol{H}_G = \frac{v_1}{\sqrt{v_1^2 + v_2^2}} \begin{bmatrix} 1 & -\dfrac{v_2}{v_1} \\ \dfrac{v_2}{v_1} & 1 \end{bmatrix}, \tag{3.13}$$

where we assume that $v_1 \neq 0$ [52]. When $v_1 = 0$ case occurs, we choose $\boldsymbol{H}_G$ as

$$\boldsymbol{H}_G = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{3.14}$$

which simply interchange the elements of the transformed vector.

3.2.1.2 Hyperbolic Rotation

Different from the unitary Givens rotation, the Θ-unitary transformation matrix $\boldsymbol{H}_{HB}$ is required to satisfy the equality $\boldsymbol{H}_{HB} \Theta \boldsymbol{H}_{HB}^T = \boldsymbol{H}_{HB}^T \Theta \boldsymbol{H}_{HB} = \Theta$, where Θ is a diagonal matrix with entries ± 1 [52]. For the 2-dimensional vector $\boldsymbol{v} = [v_1, v_2]^T$, 2×2 dimensional Hyperbolic rotation matrix $\boldsymbol{H}_{HB}$ has two different forms depending on the ratio $|\frac{v_1}{v_2}|$. When $|\frac{v_1}{v_2}| > 1$, $\boldsymbol{H}_{HB}$ is given as

$$\boldsymbol{H}_{HB} = \frac{v_1}{\sqrt{v_1^2 - v_2^2}} \begin{bmatrix} 1 & -\dfrac{v_2}{v_1} \\ -\dfrac{v_2}{v_1} & 1 \end{bmatrix}, \tag{3.15}$$

and the transformed vector $\tilde{\boldsymbol{v}}$ results in the form of $\tilde{\boldsymbol{v}} = [v_3, 0]^T$, where $v_3^2 = v_1^2 + v_2^2$. For the opposite case, i.e., $|\frac{v_1}{v_2}| < 1$, the transformation $\boldsymbol{H}_{HB}$ is given by

$$\boldsymbol{H}_{HB} = \frac{v_2}{\sqrt{v_2^2 - v_1^2}} \begin{bmatrix} 1 & -\dfrac{v_1}{v_2} \\ -\dfrac{v_1}{v_2} & 1 \end{bmatrix}, \tag{3.16}$$

where the transformed vector $\tilde{\boldsymbol{v}}$ has the form $\tilde{\boldsymbol{v}} = [0, v_3]^T$ where $v_3^2 = v_1^2 + v_2^2$ [52]. In the following subsection, we introduce our algorithm based on the elimination property of Givens and Hyperbolic transformations.

3.2.2 Construction of the Algorithm

It is clear that the complexity of the second order algorithms essentially results from the matrix-vector multiplication, $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$ as in (3.11). Rather than getting matrix $\boldsymbol{A}_{t-1}^{-1}$ from $\boldsymbol{A}_{t-2}^{-1}$ and then calculating the multiplication $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$ individually at each iteration, we develop a direct and compact update rule, which calculates $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$ from $\boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}$ without any explicit knowledge of the $M \times M$ dimensional matrix $\boldsymbol{A}_{t-1}^{-1}$.

Similar to [4], we first define the normalization term of the update rule given in (3.11) as

$$\eta_t = 1 + \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t. \tag{3.17}$$

Then, the difference between the consecutive terms η_t and η_{t-1} is given by

$$\eta_t - \eta_{t-1} = \boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1} \boldsymbol{x}_t - \boldsymbol{x}_{t-1}^T \boldsymbol{A}_{t-2}^{-1} \boldsymbol{x}_{t-1}. \tag{3.18}$$

We define the $(M+1) \times 1$ dimensional extended vector $\tilde{\boldsymbol{x}}_t = [x_t, \boldsymbol{x}_{t-1}^T]^T$ and get the following two equalities using the relation given in (3.9),

$$\eta_t = 1 + \tilde{\boldsymbol{x}}_t^T \begin{bmatrix} \boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & 0 \end{bmatrix} \tilde{\boldsymbol{x}}_t, \tag{3.19}$$

$$\eta_{t-1} = 1 + \tilde{\boldsymbol{x}}_t^T \begin{bmatrix} 0 & \boldsymbol{0}_M^T \\ \boldsymbol{0}_M & \boldsymbol{A}_{t-2}^{-1} \end{bmatrix} \tilde{\boldsymbol{x}}_t. \tag{3.20}$$

Therefore, (3.18) becomes

$$\eta_t - \eta_{t-1} = \tilde{\boldsymbol{x}}_t^T \Delta_{t-1} \tilde{\boldsymbol{x}}_t, \tag{3.21}$$

where the update term Δ_{t-1} is defined as

$$\Delta_{t-1} \triangleq \begin{bmatrix} \boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M \\ \boldsymbol{0}_M^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & \boldsymbol{0}_M^T \\ \boldsymbol{0}_M & \boldsymbol{A}_{t-2}^{-1} \end{bmatrix}. \tag{3.22}$$

This equation implies that we do not need the exact values of $\boldsymbol{A}_{t-1}^{-1}$ and $\boldsymbol{A}_{t-2}^{-1}$ individually and it is sufficient to know the value of the defined difference Δ_{t-1} for the calculation of η_t. Moreover, we observe that the update term can be expressed in terms of rank 2 matrices, which is the key point for the reduction of complexity.

Initially, we assume that $x_t = 0$ for $t < 0$, which directly implies $\boldsymbol{A}_{-1}^{-1} = \boldsymbol{A}_{-2}^{-1} = \frac{1}{\alpha}\boldsymbol{I}_M$ using (3.4) and (3.5). Therefore, Δ_{-1} is found as

$$\Delta_{-1} = \frac{1}{\alpha}\,\mathrm{diag}\{1, 0, \ldots, 0, -1\}. \tag{3.23}$$

At this point, we define the $(M+1) \times 2$ dimensional matrix Λ_{-1} and the 2×2 dimensional matrix Π_{-1} as

$$\Lambda_{-1} = \sqrt{\frac{1}{\alpha}} \begin{bmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{bmatrix}^T, \Pi_{-1} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \tag{3.24}$$

to achieve the equality given by

$$\Delta_{-1} = \Lambda_{-1}\Pi_{-1}\Lambda_{-1}^T. \tag{3.25}$$

Here, we make an initial assumption that the low rank property of Δ_t holds for all $t \geq 0$. At the end of the discussion, we show that the assumption holds. Therefore, by using the reformulation of the difference term, we restate the η_t term given in (3.21) as

$$\eta_t = \eta_{t-1} + \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1}\Pi_{t-1}\Lambda_{t-1}^T \tilde{\boldsymbol{x}}_t. \tag{3.26}$$

For the further discussion, we prefer matrix notation and represent (3.26) as

$$\begin{bmatrix} \sqrt{\eta_t} & \mathbf{0}_2^T \end{bmatrix} \begin{bmatrix} \sqrt{\eta_t} \\ \mathbf{0}_2 \end{bmatrix} = \begin{bmatrix} \sqrt{\eta_{t-1}} & \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1} \end{bmatrix} \Theta_{t-1} \begin{bmatrix} \sqrt{\eta_{t-1}} \\ \Lambda_{t-1}^T \tilde{\boldsymbol{x}}_t \end{bmatrix}, \tag{3.27}$$

where Θ_{t-1} is defined as

$$\Theta_{t-1} \triangleq \begin{bmatrix} 1 & \mathbf{0}_2^T \\ \mathbf{0}_2 & \Pi_{t-1} \end{bmatrix}. \tag{3.28}$$

We first employ a unitary Givens transformation $\boldsymbol{H}_{G,t}$ in order to zero out the second element of the vector $[\sqrt{\eta_{t-1}}, \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1}]$ and then use a Θ_{t-1}-unitary Hyperbolic rotation $\boldsymbol{H}_{HB}$, i.e., $\boldsymbol{H}_{HB,t}\Theta_{t-1}\boldsymbol{H}_{HB,t}^T = \Theta_{t-1}$, to eliminate the last term.

Consequently, we achieve the following update rule

$$\begin{bmatrix} \sqrt{\eta_t} & \mathbf{0}_2^T \end{bmatrix} = \begin{bmatrix} \sqrt{\eta_{t-1}} & \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1} \end{bmatrix} \boldsymbol{H}_t, \tag{3.29}$$

where $\boldsymbol{H}_t$ represents the overall transformation process. Existence of these transformation matrices is guaranteed [4]. This update gives the next normalization term η_t, however, for the $(t+1)^{th}$ update, we also need the updated value of Λ_{t-1}, i.e., Λ_t, explicitly. Moreover, even calculating the Λ_t term is not sufficient, since we also need the individual value of the vector $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$ to update the weight vector coefficients.

We achieve the following equalities based on the same argument that we used to get (3.19) and (3.20)

$$\begin{bmatrix} \boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t \\ 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{t-1}^{-1} & \mathbf{0}_M \\ \mathbf{0}_M^T & 0 \end{bmatrix} \tilde{\boldsymbol{x}}_t, \tag{3.30}$$

$$\begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix} = \begin{bmatrix} 0 & \mathbf{0}_M^T \\ \mathbf{0}_M & \boldsymbol{A}_{t-2}^{-1} \end{bmatrix} \tilde{\boldsymbol{x}}_t. \tag{3.31}$$

Here, by subtracting these two equations, we get

$$\begin{bmatrix} \boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix} + \Delta_{t-1}\tilde{\boldsymbol{x}}_t. \tag{3.32}$$

We emphasize that the same transformation $\boldsymbol{H}_t$, which we used to get $\sqrt{\eta_t}$, also transforms Λ_{t-1} to Λ_t and $\boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}$ to $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$, if we extend the transformed vector as follows

$$\begin{bmatrix} \sqrt{\eta_{t-1}} & \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1} \\ \dfrac{1}{\sqrt{\eta_{t-1}}} \begin{bmatrix} 0 \\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1} \end{bmatrix} & \Lambda_{t-1} \end{bmatrix} \boldsymbol{H}_t = \begin{bmatrix} \sqrt{\eta_t} & \mathbf{0}_2^T \\ \boldsymbol{q} & \boldsymbol{Q} \end{bmatrix}, \tag{3.33}$$

where we show that $\boldsymbol{q} = \frac{1}{\sqrt{\eta_t}}[\boldsymbol{x}_t^T \boldsymbol{A}_{t-1}^{-1}, 0]^T$ and $\boldsymbol{Q} = \Lambda_t$. We denote (3.33) as $\boldsymbol{B}\boldsymbol{H}_t = \tilde{\boldsymbol{B}}$, where $\boldsymbol{B}$ represents the input matrix and $\tilde{\boldsymbol{B}}$ states the output matrix of the transformation. Then, the following equality is achieved

$$\tilde{\boldsymbol{B}}\Theta_{t-1}\tilde{\boldsymbol{B}}^T = \boldsymbol{B}\Theta_{t-1}\boldsymbol{B}^T \tag{3.34}$$

since $\boldsymbol{H}_t$ is Θ_{t-1} unitary, i.e., $\boldsymbol{B}\boldsymbol{H}_t\Theta_{t-1}\boldsymbol{H}_t^T\boldsymbol{B}^T = \boldsymbol{B}\Theta_{t-1}\boldsymbol{B}^T$. Equating the elements of matrices in both sides of (3.34) yields

$$\begin{aligned}\boldsymbol{q}\sqrt{\eta_t} &= \begin{bmatrix}0\\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}\end{bmatrix} + \Delta_{t-1}\tilde{\boldsymbol{x}}_t,\\ \boldsymbol{q}\boldsymbol{q}^T + \boldsymbol{Q}\Pi_{t-1}\boldsymbol{Q}^T &= \frac{1}{\eta_{t-1}}\begin{bmatrix}0\\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}\end{bmatrix}\begin{bmatrix}0\\ \boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}\end{bmatrix}^T + \Delta_{t-1}.\end{aligned} \tag{3.35}$$

We know from (3.32) that the left hand side of the first term in (3.35) equals to $[\boldsymbol{x}_t^T\boldsymbol{A}_{t-1}^{-1}, 0]^T$ and $\boldsymbol{q}$ is given by

$$\boldsymbol{q} = \frac{1}{\sqrt{\eta_t}}\begin{bmatrix}\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t\\ 0\end{bmatrix}. \tag{3.36}$$

Hence, we identify the value of $\boldsymbol{Q}$ matrix using the second term in (3.35) as

$$\begin{aligned}\boldsymbol{Q}\Pi_{t-1}\boldsymbol{Q}^T &= \begin{bmatrix}0 & \boldsymbol{0}_M^T\\ \boldsymbol{0}_M & \dfrac{\boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}\boldsymbol{x}_{t-1}^T\boldsymbol{A}_{t-2}^{-1}}{\eta_{t-1}}\end{bmatrix}\\ &+ \left(\begin{bmatrix}\boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & 0\end{bmatrix} - \begin{bmatrix}0 & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & \boldsymbol{A}_{t-2}^{-1}\end{bmatrix}\right)\\ &- \boldsymbol{q}\boldsymbol{q}^T,\end{aligned} \tag{3.37}$$

where we expand the Δ_{t-1} term using its definition given in (3.22). We know that the term $\frac{1}{\eta_{t-1}}\boldsymbol{A}_{t-2}^{-1}\boldsymbol{x}_{t-1}\boldsymbol{x}_{t-1}^T\boldsymbol{A}_{t-2}^{-1}$ equals to the difference $\boldsymbol{A}_{t-2}^{-1} - \boldsymbol{A}_{t-1}^{-1}$ using the update relation (3.12). Therefore, substituting this equality and inserting the value of $\boldsymbol{q}$ yields

$$\begin{aligned}\boldsymbol{Q}\Pi_{t-1}\boldsymbol{Q}^T &= \left(\begin{bmatrix}0 & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & \boldsymbol{A}_{t-2}^{-1}\end{bmatrix} - \begin{bmatrix}0 & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & \boldsymbol{A}_{t-1}^{-1}\end{bmatrix}\right)\\ &+ \left(\begin{bmatrix}\boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & 0\end{bmatrix} - \begin{bmatrix}0 & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & \boldsymbol{A}_{t-2}^{-1}\end{bmatrix}\right)\\ &- \left(\begin{bmatrix}\boldsymbol{A}_{t-1}^{-1} & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & 0\end{bmatrix} - \begin{bmatrix}\boldsymbol{A}_t^{-1} & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & 0\end{bmatrix}\right)\\ &= \begin{bmatrix}\boldsymbol{A}_t^{-1} & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & 0\end{bmatrix} - \begin{bmatrix}0 & \boldsymbol{0}_M\\ \boldsymbol{0}_M^T & \boldsymbol{A}_{t-1}^{-1}\end{bmatrix}\\ &= \Delta_t\\ &= \Lambda_t\Pi_t\Lambda_t^T.\end{aligned} \tag{3.38}$$

Algorithm 4 Fast Online Newton Step

Choose $\alpha > 0$
Choose window size M
Determine the step size μ
$\Lambda_{-1} = \sqrt{\frac{1}{\alpha}} \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}^T$
$\Pi = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
$\Theta = \begin{bmatrix} 1 & \mathbf{0}_2^T \\ \mathbf{0}_2 & \Pi \end{bmatrix}$
$\boldsymbol{x}_0 = \mathbf{0}_M$
$\boldsymbol{w}_0 = \mathbf{0}_M$
$\eta_{-1} = 1$
$\boldsymbol{\rho}_{-1} = \mathbf{0}_M$
while $t \geq 0$ **do**
    $\tilde{\boldsymbol{x}}_t = [x_t, \boldsymbol{x}_t^T]^T$
    $\hat{x}_{t+1} = \boldsymbol{w}_t^T \boldsymbol{x}_t$
    $e_t = x_{t+1} - \hat{x}_{t+1}$
    $\boldsymbol{B} = \begin{bmatrix} \sqrt{\eta_{t-1}} & \tilde{\boldsymbol{x}}_t^T \Lambda_{t-1} \\ \begin{bmatrix} 0 \\ \boldsymbol{\rho}_{t-1} \end{bmatrix} & \Lambda_{t-1} \end{bmatrix}$
    Determine a Givens rotation $\boldsymbol{H}_{G,t}$ for $\boldsymbol{B}$
    $\acute{\boldsymbol{B}} = \boldsymbol{B}\boldsymbol{H}_{G,t}$
    Determine a Hyperbolic rotation $\boldsymbol{H}_{HB,t}$ for $\acute{\boldsymbol{B}}$
    $\begin{bmatrix} \sqrt{\eta_t} & \mathbf{0}_2^T \\ \begin{bmatrix} \boldsymbol{\rho}_t \\ 0 \end{bmatrix} & \Lambda_t \end{bmatrix} = \acute{\boldsymbol{B}}\boldsymbol{H}_{HB,t}$
    if $e_t > \epsilon$ **then**
        $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1}{\mu} \begin{bmatrix} \boldsymbol{\rho}_t \sqrt{\eta_t} \\ \eta_t \end{bmatrix}$
    else if $e_t < -\epsilon$ **then**
        $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{1}{\mu} \begin{bmatrix} \boldsymbol{\rho}_t \sqrt{\eta_t} \\ \eta_t \end{bmatrix}$
    else
        $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t$
    end if
    $\boldsymbol{x}_t = [x_t, x_{t-1}, \dots, x_{t-M+1}]^T$
end while

This equality implies that Π is time invariant, i.e., $\Pi_{t-1} = \Pi_t$ and $\boldsymbol{Q}$ is given as

$$\boldsymbol{Q} = \Lambda_t. \tag{3.39}$$

Hence, we show that when the low rank property of the difference term Δ_t is achieved for $t = i - 1$, it is preserved for the iteration $t = i$, for $i \geq 0$. Therefore, the transformation in (3.33) gives all the necessary information and provides a complete update rule. As a result, the weight vector is updated as

$$\boldsymbol{w}_t = \begin{cases} \boldsymbol{w}_{t-1} + \dfrac{1}{\mu}\left[\dfrac{\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t}{\eta_t}\right], & \text{if } e_t > \epsilon \\ \boldsymbol{w}_{t-1} - \dfrac{1}{\mu}\left[\dfrac{\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t}{\eta_t}\right], & \text{if } e_t < -\epsilon \\ \boldsymbol{w}_{t-1}, & \text{otherwise} \end{cases}, \tag{3.40}$$

where the individual value of $\boldsymbol{A}_{t-1}^{-1}\boldsymbol{x}_t$ is found by multiplying (3.36) by $\sqrt{\eta_t}$, which is the left upper most entry of the transformed matrix $\tilde{\boldsymbol{B}}$, and taking the first M elements. The complete algorithm is provided in Algorithm 4 with all initializations and required updates.

The processed matrix $\boldsymbol{B}$ in each iteration has the dimensions $(M + 2) \times 3$, which results in the computational complexity in the order of $O(M)$. Hence, the presented implementation reduces the computational complexity of the second order ONS algorithm significantly for the sequential prediction framework. Moreover, since there is no statistical assumptions, we obtain the same error rates compared to the regular implementation.

3.3 Simulations

In this section, we illustrate the efficiency of our algorithm on widely used synthetic and real life sequential big datasets. We implement the proposed fast Online Newton Step (F-ONS), the regular Online Newton Step (R-ONs) and the first order Online Gradient Descent (OGD) with different settings to compare their computational loads. We first work on a large chaotic sequence, i.e., Henon map [55], with 0.5 Billion samples and illustrate the total computation

time and the corresponding mean square error (MSE) curves of each algorithm. Then, we implement each algorithm on a large pseudo periodic time series which again contains 0.5 Billion time instances and represent the total elapsed time for each algorithm to reach the steady state. We finally use two different real life big sequential datasets, one of which is the CMU ARCTIC speech dataset where a professional US English male speaker reads 1132 distinct utterances [53]. The recording is performed at 16 KHz and there exist more than 50 million samples. The second real life dataset is a weather forecasting trace provided by Davis Weather station in Amherst, Massachusetts [54]. Temperature data was collected every 5 minutes during a period of 7 years from 2006 to 2013. There exist more than 600 thousand samples. Hence, all four datasets are suitable for simulating big data scenarios. The data sequences are scaled such that all samples are in between $[-1, 1]$.

3.3.1 Computational Complexity Analysis

As the first set of experiments, we examine the computation time of the proposed F-ONS, the standard R-ONS and the OGD algorithms.

3.3.1.1 Chaotic Data Sequence

We first work on a chaotic sequence, e.g., Henon map [55], which is widely used in sequential regression literature for evaluation purposes. The sequence is generated from the following structure

$$x_t = 1 - \alpha x_{t-1}^2 + \beta x_{t-2}. \tag{3.41}$$

This structure is known to exhibit a chaotic structure when the parameters are selected as $\alpha = 1.4$ and $\beta = 0.5$ [55]. The initialization values x_{-2} and x_{-1} are chosen from a normal distribution with zero-mean and unit-variance. Using (3.41), we have generated 0.5 Billion sequential data instances and used them to evaluate the performance of the proposed efficient algorithm. In Fig. 3.1,

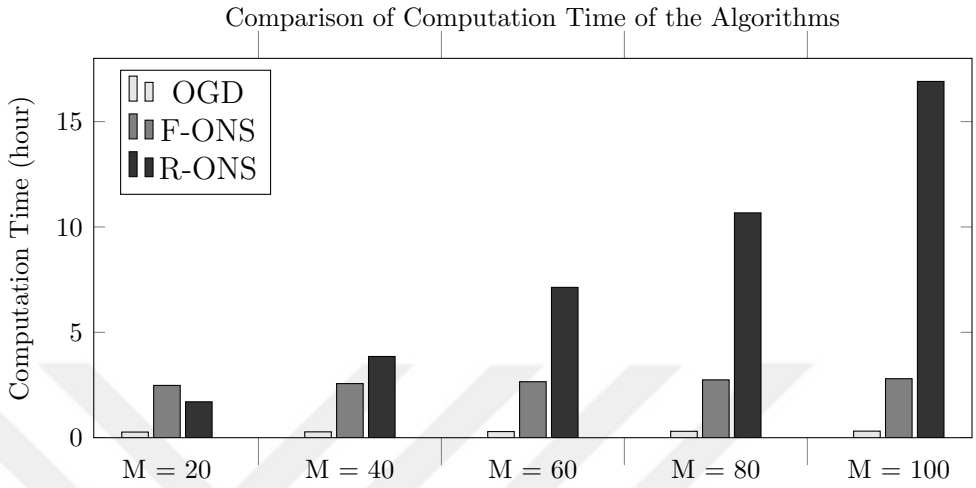

Figure 3.1: Comparison of the total computation time with different feature dimension for processing 0.5 billion data points generated from (1). F-ONS : Fast Online Newton Step, R-ONS : Regular Online Newton Step, OGD : Online Gradient Descent.

we illustrate the total computation time of each algorithms, e.g., the proposed F-ONS, the standard R-ONS and the first order OGD, with respect to different feature dimensions for processing a total of 0.5 Billion samples.

On this large chaotic sequence, we demonstrate that the proposed F-ONS algorithm provides a significant reduction on the computation time especially when the dimension of the feature vector M is high. Even though both the OGD and the F-ONS have the same order of time complexity, i.e., $O(M)$, the computation time of the first order OGD is smaller. This is an expected result since the F-ONS performs additional transformation operations in each iterations. The most significant observation of Fig. 3.1 is that increasing the dimensionality of features results in only a linear increase on the time complexity of the proposed F-ONS, even though it is a second order algorithm.

3.3.1.2 Pseudo Periodic Synthetic Time Series

As the second dataset, we use a Pseudo Periodic Synthetic Time Series [56], obtained from UCI KDD dataset archive. The sequence is generated by the

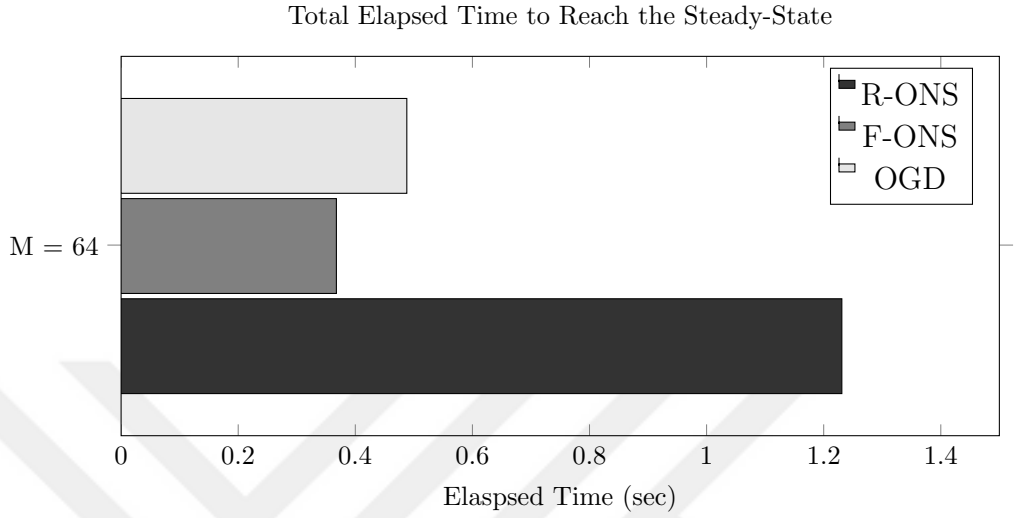

Figure 3.2: Comparison of the total elapsed time for which the algorithms reach the steady-state. F-ONS : Fast Online Newton Step, R-ONS : Regular Online Newton Step, OGD : Online Gradient Descent.

following function

$$\bar{y} = \sum_{i=3}^{7} \sin\left(2\pi\big(2^{2+i} + \text{rand}(2^i)\big)\bar{t}\right), \tag{3.42}$$

where the vector $\bar{t}$ consists of uniform samples on the $[0, 1]$ interval with a fixed step size of $2 \cdot 10^{-9}$. The sequence generated from (3.42) appears highly periodic but it never exactly repeats itself. Therefore, this dataset creates a reasonable framework for sequential prediction problem. The total number of instances in this pseudo periodic sequence is 0.5 billion similar to the previous dataset. The function $\text{rand}(2^i)$ generates a random value from the uniform distribution between 0 and 2^i. Here, all the data points are scaled to the $[-1, 1]$ interval. For this dataset, we illustrate the total elapsed time of each algorithm to reach their steady state error rates.

In Fig. 3.2, we represent the total elapsed time of each algorithm to reach their steady-state regions for $M = 64$ case. It is a significant observation that the second order F-ONS reaches the steady-state faster compared to the first order OGD algorithm. The reason is that F-ONS requires much less number of samples for convergence [34]. Even though the R-ONS processes the same number

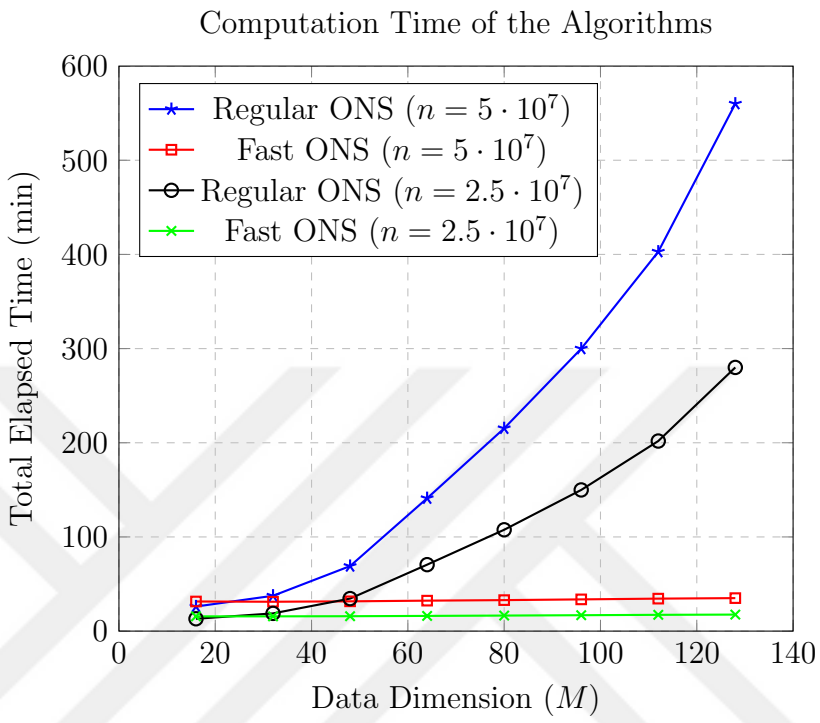

Figure 3.3: Total elapsed times for both Regular and Fast implementations of Online Newton Step algorithm with different data dimensions M. Two different dataset length n is chosen to examine the corresponding effect.

of samples with the F-ONS, it takes much more time for the R-ONS to complete processing.

3.3.1.3 Real Life Speech Dataset

We now examine the computation time of both the proposed efficient ONS and the regular ONS algorithms on a real life speech dataset. We first work on the two partitions of CMU ARCTIC speech dataset with lengths $n = 5 \cdot 10^7$ and $n = 2.5 \cdot 10^7$, and measure the corresponding total processing times. Sequences of different lengths are chosen to illustrate the effect of increasing data length. For both sequences, we choose feature vectors, $\boldsymbol{x}_t \in \mathbb{R}^M$, with several dimensions ranging from $M = 16$ to $M = 128$. In Fig. 3.3, we demonstrate the computation time comparisons of the regular and the fast implementations of ONS algorithm.

As expected from our results, complexity of the regularly implemented ONS algorithm shows a quadratic increase with respect to the dimension of the feature vectors, M. However, the proposed efficient implementation provides a linear growth on the computational complexity of the ONS algorithm. A substantial observation from Fig. 3.3 is that, with an increasing dimensionality of the space of feature vectors, the reduction in the complexity becomes outstanding. We also observe that the growth in the dataset length causes the same linear effect on both algorithms, i.e., doubling the total length n results in the doubled computation times.

3.3.1.4 Real Life Temperature Trace Dataset

We then consider the real life weather forecasting temperature dataset, where in this case the total length of data sequence is not as much as the previous ones. Therefore, we specifically concentrate on much larger values for the dimension of the feature vectors. Here, we choose the dimension of the space of feature vectors ranging from $M = 100$ to $M = 1000$ and the total length of data sequence is $n = 6 \cdot 10^5$. In Fig. 3.4, we illustrate the relative gain of the introduced fast ONS algorithm with respect to the regular implementation of ONS and the OGD algorithm in terms of the total computation times. We observe that the relative computation time gain of the presented algorithm shows a significant improvement in comparison with the regular ONS, as the data dimension increases. However, we also observe that the relative gain falls into the negative region when compared with the first order OGD algorithm. This is an expected result, since the OGD uses only an M dimensional vector in each iteration, whereas the fast ONS uses a $(M+2) \times 3$ dimensional matrix and performs additional transformation operations to update the weight vectors. Besides, the negative gain remains constant, since both algorithms eventually have the complexity in the same order of $O(M)$.

As a result, all these experiments illustrate examples of why it is nor preferred to implement the second order methods for the real life big data applications, even though they outperform the first order algorithms in the sense of error

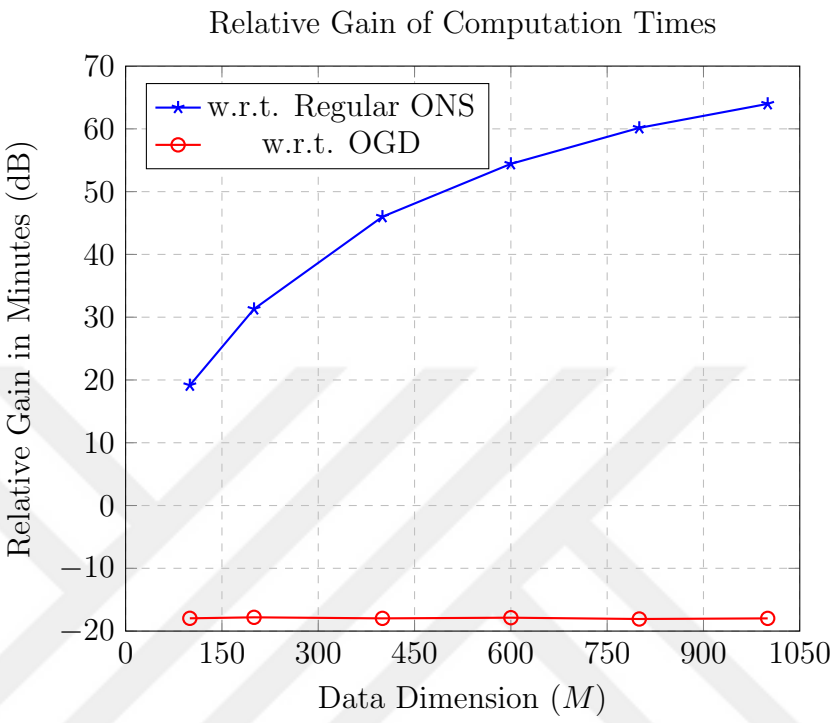

Figure 3.4: Relative gain of computation time for Fast implementation of Online Newton Step algorithm with respect to the Regular implementation of Online Newton Step and the Online Gradient Descent algorithms with different data dimensions M. Total data length is chosen as $n = 6 \cdot 10^5$.

performance. Hence, our algorithm provides the well-known merits of the second order methods, while offering a considerably low computational complexity.

3.3.2 Numerical Stability Analysis

We theoretically show that the introduced algorithm efficiently implements the R-ONS algorithm without any statistical assumptions or any information loss. Hence, both the R-ONS and the F-ONS offer the same error performances. However, there might occur negligible numerical differences as a consequence of the finite precision of real life computing systems. In the second part of the experiments, we examine the effects of the numerical calculations on the MSE curves. We also represent the MSE curve of the OGD for performance comparison. For each algorithm, the learning rates are selected to achieve the best performance

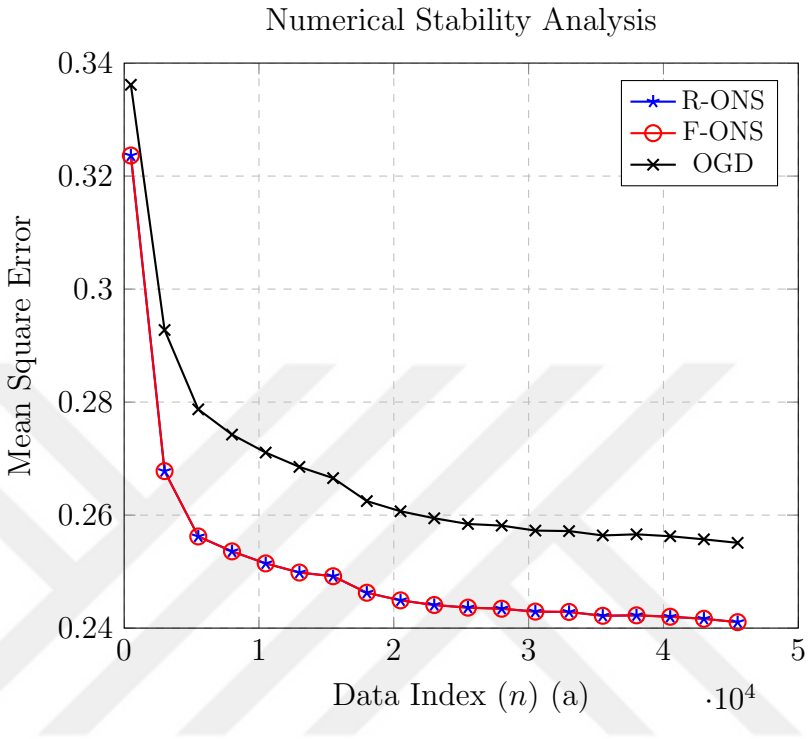

Figure 3.5: Mean square error rates of each algorithm are presented for the data dimension of $M = 64$.

in each case. We work on all the dataset used for computational complexity analysis.

3.3.2.1 Chaotic Data Sequence

The mean square error curves for the chaotic data sequence, e.g., Henon map [55], are represented in Fig 3.5 for all the F-ONS, the R-ONS and the first order OGD algorithms.

Here, we illustrate that the proposed efficient implementation is numerically stable for the processed chaotic large data sequence. It is also demonstrated that the first order OGD algorithm cannot outperform the second order ONS algorithms as expected. In this experiment, we only present the first 50 thousand samples since the algorithms reach the steady state after this much samples. The data dimension is selected as $M = 64$ for each algorithm.

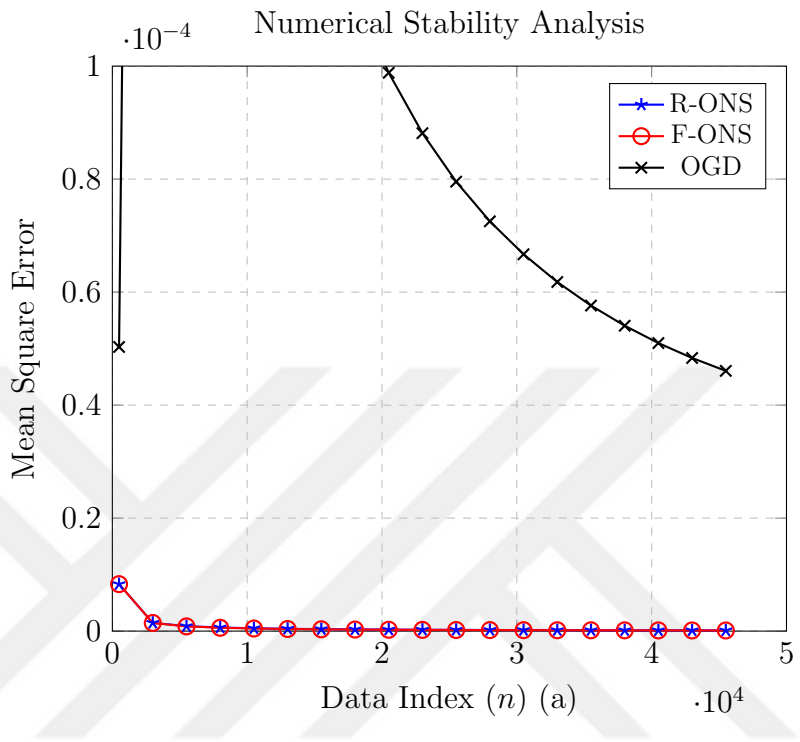

Figure 3.6: Mean square error rates of each algorithm are presented for the data dimension of $M = 64$.

3.3.2.2 Pseudo Periodic Synthetic Time Series

For the second analysis, we again use the Pseudo Periodic Synthetic Time Series dataset [56] and illustrate the mean square error curves of each algorithm in Fig. 3.6. Same as the first scenario, this experiment also represents that F-ONS is not affected by any computational precision related issues. Moreover, the considerable difference on the convergence rates of the first and the second order algorithms is also illustrated for this case.

3.3.2.3 Real Life Speech Dataset

In this case, we consider the real life CMU ARCTIC speech dataset with $n = 5{\cdot}10^4$ samples since we observe that the F-ONS and the R-ONS algorithms reach the steady state for this n. The dimension of the feature vectors is again chosen as $M = 64$, and the learning rates are determined as 0.003 for both algorithms. We

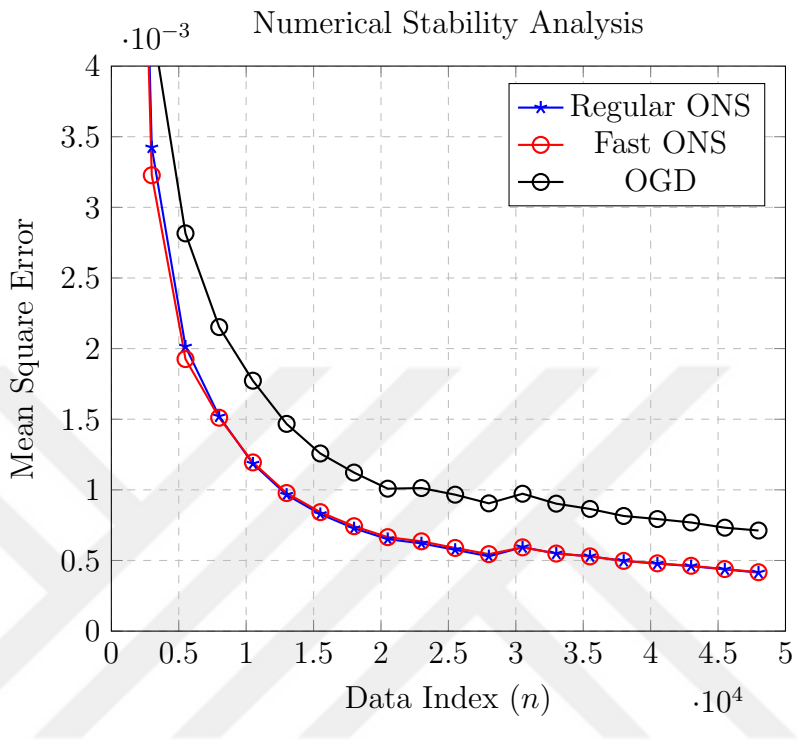

Figure 3.7: Mean Square Error curves for both Regular and Fast implementations of Online Newton Step algorithm. Data dimension is chosen as $M = 64$ for both cases.

demonstrate the comparison of mean square error curves in Fig. 3.7. A direct and significant observation is that the efficient implementation is again numerically stable. There is no observable difference between the mean square error curves in terms of both convergence and steady state for this real life case.

3.3.2.4 Real Life Temperature Trace Dataset

We work on the temperature tracking dataset as well for the numerical stability analysis. In this case, we increase the dimension of feature vectors to $M = 400$ for all algorithms. Here, we examine the effect of large dimensionality on the numerical performance of the proposed algorithm and also compare the error performances with the first order OGD algorithm. Similar to the previous cases, we only represent the first 500 samples since the second order algorithms reach the steady state at this point. The learning rates are set to 0.001 for the regular

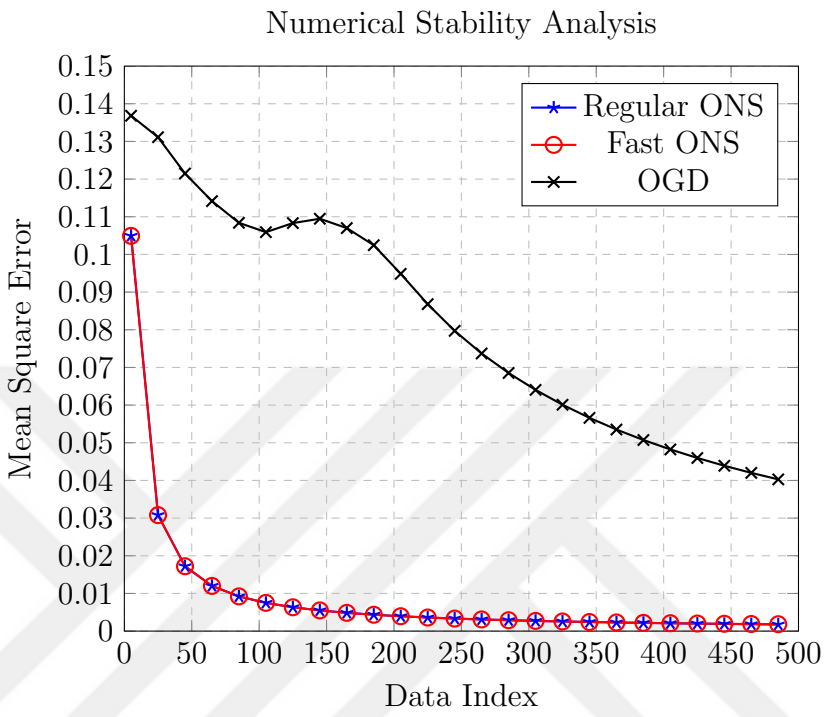

Figure 3.8: Mean Square Error curves for both Regular and Fast implementations of Online Newton Step algorithm and the Online Gradient Descent algorithm. Data dimension is chosen as $M = 400$ for all cases.

and the fast ONS and 0.1 for the OGD algorithm. In Fig. 3.8, we illustrate the corresponding mean square error curves. Same as the previous analysis, the fast ONS algorithm shows numerically no difference compared to the regular ONS algorithm. Hence, even for such high dimensional feature vectors, the proposed algorithm remains numerically stable. Additionally, we illustrate that the first order OGD algorithm again shows less than adequate performance in terms of convergence rate.

To conclude, the negative gain of computation time observed in the previous experiment becomes insignificant when we consider the mean square error analysis given in this section. Considering the MSE curves of the OGD algorithm, we observe that the second order F-ONS and the R-ONS algorithms considerably outperforms the OGD algorithm in terms of both convergence and steady-state error rates. This reveals the significance of complexity reduction for the second order algorithms. By the proposed efficient F-ONS algorithm, we provide the

merits of the second order methods with a reduced complexity that is on the same level with the first order algorithms.

Chapter 4

Conclusion

In the first chapter, 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.

In the second chapter, we investigate online sequential data prediction problem for high dimensional data sequences. Even though the second order Newton-Raphson methods achieve superior performance, compared to the gradient based algorithms, the problem of extremely high computational cost prohibits their usage in real life big data applications. For an M dimensional feature vector, the

computational complexity of these methods increases in the order of $O(M^2)$. To this end, we introduce a highly efficient implementation that reduces the computational complexity of the Newton-Raphson methods from $O(M^2)$ to $O(M)$. The presented algorithm does not require any statistical assumption on the data sequence. We only use the similarity between the consecutive feature vectors without any information loss. Hence, our algorithm offers the outstanding performance of the second order methods with the low computational cost of the first order methods. We illustrate that the efficient implementation of Newton-Raphson methods attains significant computational gains, as the data dimension grows. We also show that our algorithm is numerically stable.

Bibliography

[1] A. Ingle, J. Bucklew, W. Sethares, and T. Varghese, "Slope estimation in noisy piecewise linear functions," *Signal Processing*, vol. 108, pp. 576 – 588, 2015.

[2] M. Scarpiniti, D. Comminiello, R. Parisi, and A. Uncini, "Nonlinear spline adaptive filtering," *Signal Processing*, vol. 93, no. 4, pp. 772 – 783, 2013.

[3] Y. Yilmaz and X. Wang, "Sequential distributed detection in energy-constrained wireless sensor networks," *IEEE Transactions on Signal Processing*, vol. 17, no. 4, pp. 335–339, 2014.

[4] A. H. Sayed, *Fundamentals of Adaptive Filtering*. NJ: John Wiley & Sons, 2003.

[5] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 97–107, Jan 2014.

[6] T. Moon and T. Weissman, "Universal FIR MMSE filtering," *IEEE Transactions on Signal Processing*, vol. 57, no. 3, pp. 1068–1083, 2009.

[7] S. S. Kozat, A. C. Singer, and A. J. Bean, "A tree-weighting approach to sequential decision problems with multiplicative loss," *Signal Processing*, vol. 91, no. 4, pp. 890 – 905, 2011.

[8] N. Asadi, J. Lin, and A. de Vries, "Runtime optimizations for tree-based machine learning models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 2281–2292, Sept 2014.

[9] A. C. Singer, G. W. Wornell, and A. V. Oppenheim, "Nonlinear autoregressive modeling and estimation in the presence of noise," *Digital Signal Processing*, vol. 4, no. 4, pp. 207–221, 1994.

[10] O. J. J. Michel, A. O. Hero, and A.-E. Badel, "Tree-structured nonlinear signal modeling and prediction," *IEEE Transactions on Signal Processing*, vol. 47, no. 11, pp. 3027–3041, 1999.

[11] P. Ghosh and V. L. R. Chinthalapati, "Financial time series forecasting using agent based models in equity and fx markets," in *2014 6th Computer Science and Electronic Engineering Conference (CEEC)*, pp. 97–102, Sept 2014.

[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)*, vol. 2, pp. 1160–1163, Dec 2013.

[13] K. mei Zheng, X. Qian, and N. An, "Supervised non-linear dimensionality reduction techniques for classification in intrusion detection," in *2010 International Conference on Artificial Intelligence and Computational Intelligence (AICI)*, vol. 1, pp. 438–442, Oct 2010.

[14] S. Kabbur and G. Karypis, "Nlmf: Nonlinear matrix factorization methods for top-n recommender systems," in *2014 IEEE International Conference on Data Mining Workshop (ICDMW)*, pp. 167–174, Dec 2014.

[15] R. Couillet and M. Debbah, "Signal processing in large systems," *IEEE Signal Processing Magazine*, vol. 24, pp. 211–317, 2013.

[16] R. D'Ambrosio, W. Belhajali, and M. Barlaud, "Boosting stochastic newton descent for bigdata large scale classification," in *2014 IEEE International Conference on Big Data*, pp. 36–41, Oct 2014.

[17] T. Wu, S. H. Yu, W. Liao, and C. S. Chang, "Temporal bipartite projection and link prediction for online social networks," in *2014 IEEE International Conference on Big Data*, pp. 52–59, Oct 2014.

[18] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge: Cambridge University Press, 2006.

[19] A. C. Singer, S. S. Kozat, and M. Feder, "Universal linear least squares prediction: upper and lower bounds," *IEEE Transactions on Information Theory*, vol. 48, no. 8, pp. 2354–2362, 2002.

[20] S. S. Kozat, A. T. Erdogan, A. C. Singer, and A. H. Sayed, "Steady state MSE performance analysis of mixture approaches to adaptive filtering," *IEEE Transactions on Signal Processing*, vol. 58, pp. 4050–4063, August 2010.

[21] Y. Yilmaz and S. Kozat, "Competitive randomized nonlinear prediction under additive noise," *Signal Processing Letters, IEEE*, vol. 17, pp. 335–339, April 2010.

[22] D. P. Helmbold and R. E. Schapire, "Predicting nearly as well as the best pruning of a decision tree," *Machine Learning*, vol. 27, no. 1, pp. 51–68, 1997.

[23] S. S. Kozat, A. C. Singer, and G. C. Zeitler, "Universal piecewise linear prediction via context trees," *IEEE Transactions on Signal Processing*, vol. 55, no. 7, pp. 3730–3745, 2007.

[24] E. D. Kolaczyk and R. D. Nowak, "Multiscale generalised linear models for nonparametric function estimation," *Biometrika*, vol. 92, no. 1, pp. 119–133, 2005.

[25] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context-tree weighting method: basic properties," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 653–664, 1995.

[26] A. C. Singer and M. Feder, "Universal linear prediction by model order weighting," *IEEE Transactions on Signal Processing*, vol. 47, no. 10, pp. 2685–2699, 1999.

[27] A. Gyorgy, T. Linder, and G. Lugosi, "Efficient adaptive algorithms and minimax bounds for zero-delay lossy source coding," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2337–2347, 2004.

[28] N. Vanli and S. Kozat, "A comprehensive approach to universal piecewise nonlinear regression based on trees," *IEEE Transactions on Signal Processing*, vol. 62, pp. 5471–5486, Oct 2014.

[29] R. Savani, "High-frequency trading: The faster, the better?," *IEEE Intelligent Systems*, vol. 27, pp. 70–73, July 2012.

[30] W. Cao, L. Cao, and Y. Song, "Coupled market behavior based financial crisis detection," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, Aug 2013.

[31] Y. Ding, H. Tan, W. Luo, and L. M. Ni, "Exploring the use of diverse replicas for big location tracking data," in *2014 IEEE 34th International Conference on Distributed Computing Systems (ICDCS)*, pp. 83–92, June 2014.

[32] L. Bottou and Y. Le Cun, "On-line learning for very large data sets," *Applied Stochastic Models in Business and Industry*, vol. 21, no. 2, pp. 137–151, 2005.

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

[34] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge: Cambridge University Press, 2006.

[35] S. Dasgupta and Y. Freund, "Random projection trees for vector quantization," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3229–3242, 2009.

[36] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*. Athena scientific series in optimization and neural computation, Belmont (Mass.): Athena Scientific, 1997.

[37] M. S. D. Raghunath S. Holambe, *Advances in Nonlinear Modeling for Speech Processing*. Adaptive computation and machine learning series, Springer, 2012.

[38] K. P. Murphy, *Machine learning : A probabilistic perspective*. Adaptive computation and machine learning series, Cambridge (Mass.): MIT Press, 2012.

[39] M. Mattavelli, J. Vesin, E. Amaldi, and R. Gruter, "A new approach to piecewise linear modeling of time series," in *IEEE Digital Signal Processing Workshop Proceedings*, pp. 502–505, Sep 1996.

[40] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.

[41] R. Rosipal and L. J. Trejo, "Kernel partial least squares regression in reproducing kernel Hilbert space," *J. Mach. Learn. Res.*, vol. 2, pp. 97–123, Mar. 2002.

[42] M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems*. NJ: John Wiley & Sons, 1980.

[43] A. Carini and G. L. Sicuranza, "Fourier nonlinear filters," *Signal Processing*, vol. 94, no. 0, pp. 183 – 194, 2014.

[44] L. Torgo, "Regression data sets."

[45] L. G. Rios and J. A. I. Diguez, "Big data infrastructure for analyzing data generated by wireless sensor networks," in *2014 IEEE International Congress on Big Data*, pp. 816–823, June 2014.

[46] S. V. Georgakopoulos, S. K. Tasoulis, and V. P. Plagianakos, "Efficient change detection for high dimensional data streams," in *2015 IEEE International Conference on Big Data*, pp. 2219–2222, Oct 2015.

[47] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 97–107, Jan 2014.

[48] A. C. Singer, S. S. Kozat, and M. Feder, "Universal linear least squares prediction: upper and lower bounds," *IEEE Transactions on Information Theory*, vol. 48, pp. 2354–2362, Aug 2002.

[49] E. K. P. Chong and S. H. Zak, *An introduction to optimization*. Wiley-Interscience series in discrete mathematics and optimization, New York: Wiley, 2008.

[50] S. S. Kozat, A. T. Erdogan, A. C. Singer, and A. H. Sayed, "Steady-state mse performance analysis of mixture approaches to adaptive filtering," *IEEE Transactions on Signal Processing*, vol. 58, pp. 4050–4063, Aug 2010.

[51] J. Cheng, A. N. Tegge, and P. Baldi, "Machine learning methods for protein structure prediction," *IEEE Reviews in Biomedical Engineering*, vol. 1, pp. 41–49, 2008.

[52] A. H. Sayed, *Adaptive Filters*. NJ: John Wiley & Sons, 2008.

[53] J. Kominek and A. W. Black, "Cmu arctic databases."

[54] M. Liberatore, "Umass trace repository."

[55] M. Henon, "A two-dimensional mapping with a strange attractor," *Commun. Math. Phys*, vol. 50, pp. 69–77, 1976.

[56] D. L. S. Park and W. W. Chu, "Fast retrieval of similar subsequences in long sequence databases," *3rd IEEE Knowledge and Data Engineering Exchange Workshop (KDEX)*, 1999.