2015 IEEE International Congress on Big Data

A Scalable Approach for Online Hierarchical Big Data Mining

N. Denizcan Vanli*, Muhammed O. Sayin*, Ibrahim Delibalta†≠ and Suleyman S. Kozat*
*Department of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey
Email: {vanli,sayin,kozat}@ee.bilkent.edu.tr
†AveaLabs, AVEA Communications Services Inc., Istanbul, Turkey
≠Graduate School of Social Sciences and Humanities, Koc University, Istanbul, Turkey
Email: idelibalta13@ku.edu.tr

—We study online compound decision problems in the context of sequential prediction of real valued sequences. In particular, we consider finite state (FS) predictors that are constructed based on the sequence history, whose length is quite large for applications involving big data. To mitigate overtraining problems, we define hierarchical equivalence classes and apply the exponentiated gradient (EG) algorithm to achieve the performance of the best state assignment defined on the hierarchy. For a sequence history of length h, we combine more than $2^{(h/e)^h}$ different FS predictors each corresponding to a different combination of equivalence classes and asymptotically achieve the performance of the best FS predictor with computational complexity only linear in the pattern length h. Our approach is generic in the sense that it can be applied to general hierarchical equivalence class definitions. Although we work under accumulated square loss as the performance measure, our results hold for a wide range of frameworks and loss functions as detailed in the paper.

***Keywords*-Hierarchical data mining, online learning, sequential prediction**

I. Introduction

We investigate online compound decision problems that arise in several different data mining applications, such as financial market [1] and trend [2] analyses, intrusion detection [3], service discovery [4]–[8], energy management [9], and cloud computing [10]. From a unified point of view, in such problems, we sequentially observe a real valued sequence $x_1, x_2, \ldots$ and produce a decision (or an action) $\hat{d}_t$ at each time t based on the past $x_1, x_2, \ldots, x_t$. 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. As an example, in generic online prediction problems under the square error loss, the output at time t, i.e., $\hat{d}_t$, corresponds to an estimate of the next data point x_{t+1}, where the algorithm suffers the loss $(x_{t+1} - \hat{d}_t)^2$ after x_{t+1} is revealed. The algorithm can then adjust itself in order to reduce the future losses.

We investigate a specific version of the sequential compound decision problems, where we consider FS predictors. In particular, we use a hierarchical structure (such as the relative ordering pattern of the sequence history [11], [12]) to construct our states. However, our algorithm and results are comprehensive such that they hold for most hierarchical state definitions as explained later in the paper. We also work in a deterministic setup where we refrain from any statistical assumptions and our results are guaranteed to hold in an individual sequence manner [13], [14].

Sequential predictors using FS machines (or certain pattern matching algorithms) are extensively studied both in computational learning theory [13], [15] and signal processing [16], [17] literatures since this structure naturally arises in different real life applications [1]–[3]. As an example, hierarchical models are widely used in service classification/discovery [4] and recommendation systems [8]. In a similar context, there exist universal binary prediction algorithms [18] that achieve the performance of any finite state predictor in the long run. However, note that the results of [18] are asymptotical, i.e., the data length goes to infinity, and states of the finite state machines or transitions between them are fixed, which is not the case for finite length sequences or for the framework considered in here.

In this paper, we first use the relative ordering patterns of the sequence history as our states for ease of exposition (whereas our results are generic for any hierarchical structure). In particular, at each time t, we use the last h samples of the sequence $(x_{t-h+1}, \ldots, x_t)$ to define relative ordering patterns. Since there exist a massive amount of data to be processed, we need to use large values of h in order to exploit the information in the sequence and attain a satisfactory performance. A sequence history of length h can have $h! \approx (h/e)^h$ different ordering patterns, e.g., for $h = 10$, there are approximately $10! \approx (10/e)^{10} = 4.54 \times 10^5$ different ordering patterns. Hence, training a sequential FS predictor directly using this number of patterns as states is impractical.

To mitigate this overtraining issue, we define a hierarchical structure where we tie together certain patterns to form intermediate states or equivalence classes each representing a collection of the original patterns. Here, we use the location of the largest element for grouping the patterns in a recursive manner. Certain specific combinations of these equivalence classes can be used to construct sequential FS predictors. With our approach we represent more than $2^{(h/e)^h}$ different FS predictors corresponding to different permutations

978-1-4673-7278-7/15 $31.00 © 2015 IEEE
DOI 10.1109/BigDataCongress.2015.11

A Scalable Approach for Online Hierarchical Big Data Mining

N. Denizcan Vanli*, Muhammed O. Sayin*, Ibrahim Delibalta†‡ and Suleyman S. Kozat*
*Department of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey
Email: {vanli,sayin,kozat}@ee.bilkent.edu.tr
†AveaLabs, AVEA Communications Services Inc., Istanbul, Turkey
‡Graduate School of Social Sciences and Humanities, Koc University, Istanbul, Turkey
Email: idelibalta13@ku.edu.tr

—We study online compound decision problems in the context of sequential prediction of real valued sequences. In particular, we consider finite state (FS) predictors that are constructed based on the sequence history, whose length is quite large for applications involving big data. To mitigate overtraining problems, we define hierarchical equivalence classes and apply the exponentiated gradient (EG) algorithm to achieve the performance of the best state assignment defined on the hierarchy. For a sequence history of length h, we combine more than $2^{(h/e)^h}$ different FS predictors each corresponding to a different combination of equivalence classes and asymptotically achieve the performance of the best FS predictor with computational complexity only linear in the pattern length h. Our approach is generic in the sense that it can be applied to general hierarchical equivalence class definitions. Although we work under accumulated square loss as the performance measure, our results hold for a wide range of frameworks and loss functions as detailed in the paper.

***Keywords*-Hierarchical data mining, online learning, sequential prediction.**

I. Introduction

We investigate online compound decision problems that arise in several different data mining applications, such as financial market [1] and trend [2] analyses, intrusion detection [3], service discovery [4]–[8], energy management [9], and cloud computing [10]. From a unified point of view, in such problems, we sequentially observe a real valued sequence $x_1, x_2, \dots$ and produce a decision (or an action) $\hat{d}_t$ at each time t based on the past $x_1, x_2, \dots, x_t$. 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. As an example, in generic online prediction problems under the square error loss, the output at time t, i.e., $\hat{d}_t$, corresponds to an estimate of the next data point x_{t+1}, where the algorithm suffers the loss $(x_{t+1}-\hat{d}_t)^2$ after x_{t+1} is revealed. The algorithm can then adjust itself in order to reduce the future losses.

We investigate a specific version of the sequential compound decision problems, where we consider FS predictors. In particular, we use a hierarchical structure (such as the relative ordering pattern of the sequence history [11], [12]) to construct our states. However, our algorithm and results are comprehensive such that they hold for most hierarchical state definitions as explained later in the paper. We also work in a deterministic setup where we refrain from any statistical assumptions and our results are guaranteed to hold in an individual sequence manner [13], [14].

Sequential predictors using FS machines (or certain pattern matching algorithms) are extensively studied both in computational learning theory [13], [15] and signal processing [16], [17] literatures since this structure naturally arises in different real life applications [1]–[3]. As an example, hierarchical models are widely used in service classification/discovery [4] and recommendation systems [8]. In a similar context, there exist universal binary prediction algorithms [18] that achieve the performance of any finite state predictor in the long run. However, note that the results of [18] are asymptotical, i.e., the data length goes to infinity, and states of the finite state machines or transitions between them are fixed, which is not the case for finite length sequences or for the framework considered in here.

In this paper, we first use the relative ordering patterns of the sequence history as our states for ease of exposition (whereas our results are generic for any hierarchical structure). In particular, at each time t, we use the last h samples of the sequence $(x_{t-h+1}, \dots, x_t)$ to define relative ordering patterns. Since there exist a massive amount of data to be processed, we need to use large values of h in order to exploit the information in the sequence and attain a satisfactory performance. A sequence history of length h can have $h! \approx (h/e)^h$ different ordering patterns, e.g., for $h = 10$, there are approximately $10! \approx (10/e)^{10} = 4.54 \times 10^5$ different ordering patterns. Hence, training a sequential FS predictor directly using this number of patterns as states is impractical.

To mitigate this overtraining issue, we define a hierarchical structure where we tie together certain patterns to form intermediate states or equivalence classes each representing a collection of the original patterns. Here, we use the location of the largest element for grouping the patterns in a recursive manner. Certain specific combinations of these equivalence classes can be used to construct sequential FS predictors. With our approach we represent more than $2^{(h/e)^h}$ different FS predictors corresponding to different permutations

978-1-4673-7278-7/15 $31.00 © 2015 IEEE
DOI 10.1109/BigDataCongress.2015.11

and combinations of these equivalence classes. We then apply the EG algorithm [19] to asymptotically achieve the performance of the best FS predictor in the hierarchy with computational complexity only linear in the pattern length. Our approach is generic such that our algorithm can be applied to a wide range of hierarchical equivalence class definitions (other than order preserving patterns or location of the largest element) to asymptotically achieve the performance of the optimal sequential FS predictor with the best state assignment.

The remainder of the paper is as follows. In Section II, we provide the problem description and introduce a method to create FS predictors. In Section III, we then introduce an online algorithm to combine the outputs of all doubly exponential number of different FS predictors with a computational complexity only linear in the hierarchy depth. We illustrate the performance of the proposed algorithm over various experiments in Section IV. The paper concludes with certain remarks in Section V

II. FS Predictors Based on Order Preserving Patterns

In order to produce the output $\hat{d}_t$, we use finite state (FS) predictors. In general, a FS predictor has a prediction function

$$\hat{d}_t = f(s_t), \tag{1}$$

where $s_t \in \mathcal{S}$ is the current state taking values from a finite set $\mathcal{S} = \{1, \ldots, S\}$, and upon the observation of the new data x_{t+1}, the states are traversed according to the next state function

$$s_{t+1} = g(s_t, x_{t+1}), \tag{2}$$

where f and g are arbitrary functions. The accumulated loss of this FS predictor is given by $\sum_{t=1}^{n} (d_t - \hat{d}_t)^2$ over any length n.

We use the relative ordering pattern of the past sequence as our states as shown in Figure 1. In particular, at each time t, we use the last h samples of the sequence history $x_{t-h+1}, \ldots, x_t$ to define relative ordering patterns. As an example, for $h = 3$, we have 6 different possible patterns, i.e., $(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)$, where "3" represents the location of the largest value and "1" represents the location of the smallest value, e.g., the sequence $(x_{t-2}, x_{t-1}, x_t) = (10.2, 13.5, -1)$ corresponds to the pattern or ordering $(2,3,1)$. Given h and this set of ordering patterns, one can arbitrarily assign each pattern to a state so that $s_t \in \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$ for each t. After fixing the state assignments, s_{t+1} is selected after observing x_{t+1}.

Given the state definitions, we can easily construct a sequential algorithm that asymptotically achieves the performance of the optimal batch predictor (which uses the same states) [13], such as using the least mean squares (LMS) or the recursive least squares (RLS) algorithms. However, for finite length sequences, such an approach can only provide satisfactory results if we have enough data to learn the optimal batch predictor at each state, i.e., there should be enough occurrences of each state pattern in the past $x_1, \ldots, x_t$. However, even for a moderate value of h that define meaningful patterns in real life applications involving big data [11], [12], say for $h = 10$, the number of patterns grows as $h! = 10!$. In this sense, to train a sequential predictor for each ordering patterns, we require a substantial amount of past observations, which is not available in most real life applications even for stationary data [16]. This problem is more severe for nonstationary data. One can mitigate this problem by defining "super set" equivalence classes or tying certain states together as widely used in speech recognition applications when there are not enough data to adequately train all the phoneme states [20].

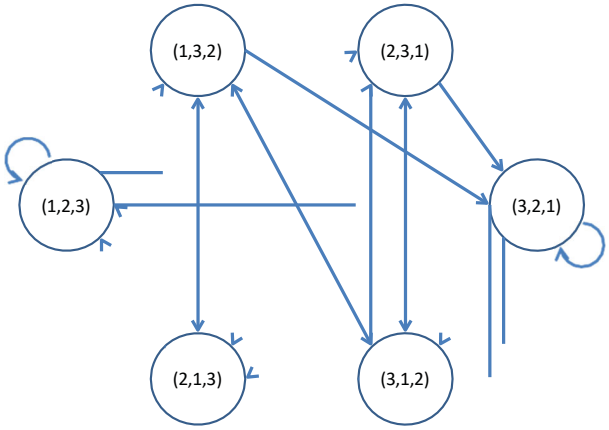

Figure 1: FS Diagram for $h = 3$, where all allowable transitions are drawn.

Although a sequence of length h, i.e., $(x_{t-h+1}, \ldots, x_t)$, can have $h!$ different ordering patterns, most of these patterns share similar characteristics that can be exploited to group (or tie) them together to form intermediate states each representing a collection of the original patterns. In this paper, we use the location of the largest element as the main characteristics in order to group the patterns in an hierarchical manner. However, our methods are generic such that they cover other hierarchical equivalence class definitions as detailed in Section III.

For example in Figure 2, we show how we hierarchically divide all possible patterns into different groups or equivalence classes for $h = 3$. At the first level, i.e., level-1, in the figure, we first group the ordering patterns into h different equivalence classes based on the location of the largest element, e.g., the equivalence class $c_{1,1}$ represents all the ordering patterns that have the largest element at the first location x_t, where these patterns are

and combinations of these equivalence classes. We then apply the EG algorithm [19] to asymptotically achieve the performance of the best FS predictor in the hierarchy with computational complexity only linear in the pattern length. Our approach is generic such that our algorithm can be applied to a wide range of hierarchical equivalence class definitions (other than order preserving patterns or location of the largest element) to asymptotically achieve the performance of the optimal sequential FS predictor with the best state assignment.

The remainder of the paper is as follows. In Section II, we provide the problem description and introduce a method to create FS predictors. In Section III, we then introduce an online algorithm to combine the outputs of all doubly exponential number of different FS predictors with a computational complexity only linear in the hierarchy depth. We illustrate the performance of the proposed algorithm over various experiments in Section IV. The paper concludes with certain remarks in Section V

II. FS Predictors Based on Order Preserving Patterns

In order to produce the output $\hat{d}_t$, we use finite state (FS) predictors. In general, a FS predictor has a prediction function

$$\hat{d}_t = f(s_t), \tag{1}$$

where $s_t \in \mathcal{S}$ is the current state taking values from a finite set $\mathcal{S} = \{1, \ldots, S\}$, and upon the observation of the new data x_{t+1}, the states are traversed according to the next state function

$$s_{t+1} = g(s_t, x_{t+1}), \tag{2}$$

where f and g are arbitrary functions. The accumulated loss of this FS predictor is given by $\sum_{t=1}^{n}(d_t - \hat{d}_t)^2$ over any length n.

We use the relative ordering pattern of the past sequence as our states as shown in Figure 1. In particular, at each time t, we use the last h samples of the sequence history $x_{t-h+1}, \ldots, x_t$ to define relative ordering patterns. As an example, for $h = 3$, we have 6 different possible patterns, i.e., $(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2)$, $(3,2,1)$, where "3" represents the location of the largest value and "1" represents the location of the smallest value, e.g., the sequence $(x_{t-2}, x_{t-1}, x_t) = (10.2, 13.5, -1)$ corresponds to the pattern or ordering $(2,3,1)$. Given h and this set of ordering patterns, one can arbitrarily assign each pattern to a state so that $s_t \in \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$ for each t. After fixing the state assignments, s_{t+1} is selected after observing x_{t+1}.

Given the state definitions, we can easily construct a sequential algorithm that asymptotically achieves the performance of the optimal batch predictor (which uses the same states) [13], such as using the least mean squares (LMS) or the recursive least squares (RLS) algorithms. However, for finite length sequences, such an approach can only provide satisfactory results if we have enough data to learn the optimal batch predictor at each state, i.e., there should be enough occurrences of each state pattern in the past $x_1, \ldots, x_t$. However, even for a moderate value of h that define meaningful patterns in real life applications involving big data [11], [12], say for $h = 10$, the number of patterns grows as $h! = 10!$. In this sense, to train a sequential predictor for each ordering patterns, we require a substantial amount of past observations, which is not available in most real life applications even for stationary data [16]. This problem is more severe for nonstationary data. One can mitigate this problem by defining "super set" equivalence classes or tying certain states together as widely used in speech recognition applications when there are not enough data to adequately train all the phoneme states [20].

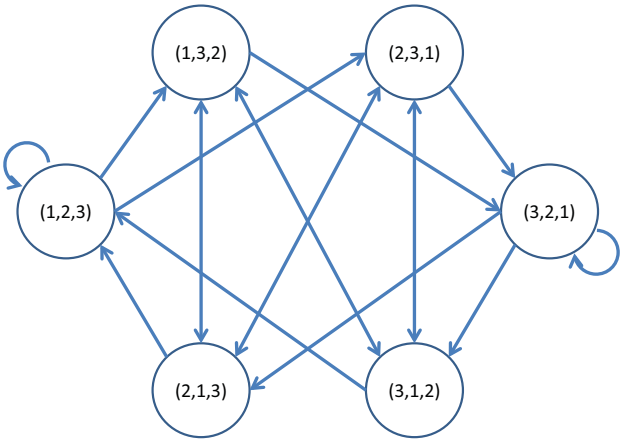

Figure 1: FS Diagram for $h = 3$, where all allowable transitions are drawn.

Although a sequence of length h, i.e., $(x_{t-h+1}, \ldots, x_t)$, can have $h!$ different ordering patterns, most of these patterns share similar characteristics that can be exploited to group (or tie) them together to form intermediate states each representing a collection of the original patterns. In this paper, we use the location of the largest element as the main characteristics in order to group the patterns in an hierarchical manner. However, our methods are generic such that they cover other hierarchical equivalence class definitions as detailed in Section III.

For example in Figure 2, we show how we hierarchically divide all possible patterns into different groups or equivalence classes for $h = 3$. At the first level, i.e., level-1, in the figure, we first group the ordering patterns into h different equivalence classes based on the location of the largest element, e.g., the equivalence class $c_{1,1}$ represents all the ordering patterns that have the largest element at the first location x_t, where these patterns are

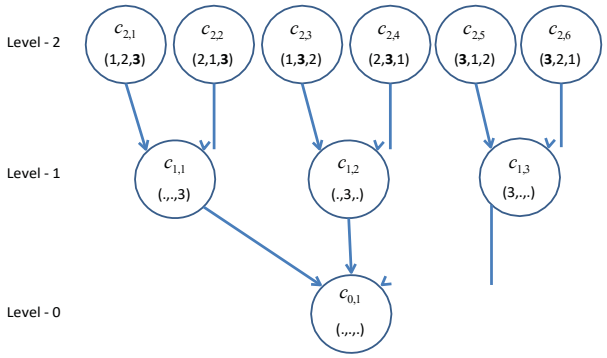

Figure 2: All equivalence classes for the FS diagram with $h = 3$.

$c_{1,1} = (\cdot, \cdot, 3) = \{(1, 2, 3), (2, 1, 3)\}$. At the second level, i.e., level-2, we continue to separate the patterns into finer groups. Since the length of the pattern is $h = 3$, we only have $h - 1 = 2$ levels to consider. Each equivalence class at the same level, e.g., $c_{1,1}, c_{1,2}, \ldots, c_{1,h}$, represents a disjoint grouping of the original patterns $S = \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$. One can completely partition the set of patterns S into hierarchical equivalence classes in this way and each equivalence class in Figure 2 represents the set of patterns that are the union of the set of patterns of their subclasses.

Certain collection of the equivalence classes in Figure 2 completely covers all the original ordering patterns S, such as $\{(\cdot, \cdot, 3), (\cdot, 3, \cdot), (3, \cdot, \cdot)\}$. Hence, each of such configurations can be used to construct a FS predictor, whose states are these equivalence classes. With our representation, for a length h history, we can have $K_h \approx 2^{(h/e)^h}$ different tying configurations each of which completely covers the whole pattern set S (since $K_{h+1} = K_h^{h+1} + 1$).

One of these FS predictors, $\hat{d}_t^{\{k\}}$, $k = 1, \ldots, K_h$, is optimal for the current observations and the optimal FS predictor can change over time. As a comparison, when there is not enough data, the coarser FS predictor having the equivalence classes $\{c_{1,1}, c_{1,2}, c_{1,3}\}$ is expected to learn much faster than the finer FS predictor having the equivalence classes $\{c_{2,1}, c_{2,2}, c_{2,3}, c_{2,4}, c_{2,5}, c_{2,6}\}$. However, one expects the finer model to perform better as the data length increases due to its higher modeling power for stationary data. However, in this paper, instead of committing to one of these models or switching between the models, we next introduce a mixture-of-experts approach to adaptively combine all of these FS predictors, $\hat{d}^{\{k\}}$, $k = 1, \ldots, K_h$, to build a sequential predictor that is universal over all ordering patterns (with computational complexity only linear in the pattern length h). Hence, the computational complexity of our algorithm is highly scalable, which makes our algorithm very suitable for applications involving big data.

III. A Universal Trend Predictor Based on Order Preserving Patterns

Suppose we construct all possible FS predictors $\hat{d}_t^{\{k\}}$, $k = 1, \ldots, K_h$, and run them in parallel to match patterns and predict d_t. We next use the EG algorithm [19] to combine the outputs of all FS predictors to produce the final output

$$\hat{d}_t = \sum_{k=1}^{K_h} w_t^{\{k\}} \hat{d}_t^{\{k\}}, \tag{3}$$

yielding the error $e_t = d_t - \hat{d}_t$, and update the combination weights using

$$w_t^{\{k\}} = \frac{w_{t-1}^{\{k\}} \exp\left(-\mu \hat{d}_{t-1}^{\{k\}} e_{t-1}\right)}{\sum_{r=1}^{K_h} w_{t-1}^{\{r\}} \exp\left(-\mu \hat{d}_{t-1}^{\{r\}} e_{t-1}\right)}, \tag{4}$$

where $\mu > 0$ is a positive constant controlling the learning rate. The weighted mixture algorithm (3) sequentially achieves the performance of the best algorithm in the mixture [19], i.e., when applied to any $x_1, x_2, \ldots$ and $d_1, d_2, \ldots$, this algorithm has the performance

$$\frac{1}{n} \sum_{t=1}^{n} (d_t - \hat{d}_t)^2 \le \min_{k=1,\ldots,K_h} \frac{1}{n} \sum_{t=1}^{n} (d_t - \hat{d}_t^{\{k\}})^2 + O\left(\frac{\ln K_h}{n}\right), \tag{5}$$

for any n without any knowledge of the optimal $\hat{d}_t^{\{k\}}$, the future of the sequences or the data length n. We emphasize that there exist various extensions of the basic update (4) in the context of tracking the best expert or the best linear combination [21], for different loss functions [22], in a stochastic context [23] and in different frameworks [13]. We emphasize that our derivations directly cover these frameworks.

Note that even for a moderate length pattern such as $h = 5$, we have $K_h \approx 2^{(5/e)^5}$. Hence, in this form the algorithm (3) cannot be directly implemented since we need to run K_h FS predictors in parallel and monitor their performances to construct (3). We next introduce a method that implements (3) with complexity only linear in the pattern length h.

For an efficient implementation of (3), we first note that we can write (4) as

$$w_t^{\{k\}} = \frac{\exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right)}{\sum_{r=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{r\}} e_z\right)}, \tag{6}$$

after some algebra, when $w_1^{\{k\}} = 1/K_h$, $\forall k = 1, \ldots, K_h$, which yields a recursive formulation later in the paper. Before providing the recursive formulation, we observe that although there are K_h FS predictors in the mixture (3) (with different weights), the states used by these FS predictors include a relatively small number of equivalence classes shown in Figure 2. To efficiently use this observation, we

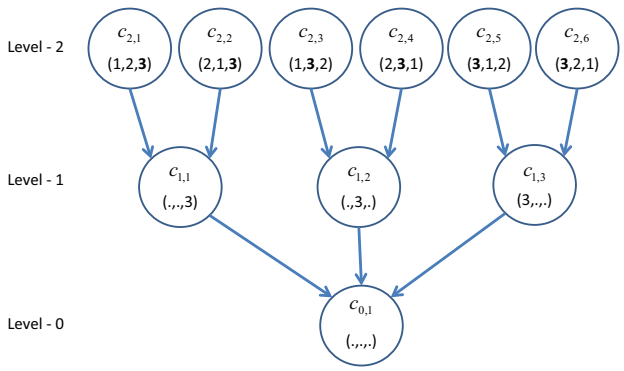

Figure 2: All equivalence classes for the FS diagram with $h = 3$.

$c_{1,1} = (\cdot, \cdot, 3) = \{(1,2,3), (2,1,3)\}$. At the second level, i.e., level-2, we continue to separate the patterns into finer groups. Since the length of the pattern is $h = 3$, we only have $h - 1 = 2$ levels to consider. Each equivalence class at the same level, e.g., $c_{1,1}, c_{1,2}, \ldots, c_{1,h}$, represents a disjoint grouping of the original patterns $\mathcal{S} = \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$. One can completely partition the set of patterns $\mathcal{S}$ into hierarchical equivalence classes in this way and each equivalence class in Figure 2 represents the set of patterns that are the union of the set of patterns of their subclasses.

Certain collection of the equivalence classes in Figure 2 completely covers all the original ordering patterns $\mathcal{S}$, such as $\{(\cdot, \cdot, 3), (\cdot, 3, \cdot), (3, \cdot, \cdot)\}$. Hence, each of such configurations can be used to construct a FS predictor, whose states are these equivalence classes. With our representation, for a length h history, we can have $K_h \approx 2^{(h/e)^h}$ different tying configurations each of which completely covers the whole pattern set $\mathcal{S}$ (since $K_{h+1} = K_h^{h+1} + 1$).

One of these FS predictors, $\hat{d}_t^{\{k\}}$, $k = 1, \ldots, K_h$, is optimal for the current observations and the optimal FS predictor can change over time. As a comparison, when there is not enough data, the coarser FS predictor having the equivalence classes $\{c_{1,1}, c_{1,2}, c_{1,3}\}$ is expected to learn much faster than the finer FS predictor having the equivalence classes $\{c_{2,1}, c_{2,2}, c_{2,3}, c_{2,4}, c_{2,5}, c_{2,6}\}$. However, one expects the finer model to perform better as the data length increases due to its higher modeling power for stationary data. However, in this paper, instead of committing to one of these models or switching between the models, we next introduce a mixture-of-experts approach to adaptively combine all of these FS predictors, $\hat{d}_t^{\{k\}}$, $k = 1, \ldots, K_h$, to build a sequential predictor that is universal over all ordering patterns (with computational complexity only linear in the pattern length h). Hence, the computational complexity of our algorithm is highly scalable, which makes our algorithm very suitable for applications involving big data.

III. A Universal Trend Predictor Based on Order Preserving Patterns

Suppose we construct all possible FS predictors $\hat{d}_t^{\{k\}}$, $k = 1, \ldots, K_h$, and run them in parallel to match patterns and predict d_t. We next use the EG algorithm [19] to combine the outputs of all FS predictors to produce the final output

$$\hat{d}_t \triangleq \sum_{k=1}^{K_h} w_t^{\{k\}} \hat{d}_t^{\{k\}}, \tag{3}$$

yielding the error $e_t \triangleq d_t - \hat{d}_t$, and update the combination weights using

$$w_t^{\{k\}} = \frac{w_{t-1}^{\{k\}} \exp\left(-\mu \hat{d}_{t-1}^{\{k\}} e_{t-1}\right)}{\sum_{r=1}^{K_h} w_{t-1}^{\{r\}} \exp\left(-\mu \hat{d}_{t-1}^{\{r\}} e_{t-1}\right)}, \tag{4}$$

where $\mu > 0$ is a positive constant controlling the learning rate. The weighted mixture algorithm (3) sequentially achieves the performance of the best algorithm in the mixture [19], i.e., when applied to any $x_1, x_2, \ldots$ and $d_1, d_2, \ldots$, this algorithm has the performance

$$\frac{1}{n}\sum_{t=1}^{n}(d_t - \hat{d}_t)^2 \leq \min_{k=1,\ldots,K_h} \frac{1}{n}\sum_{t=1}^{n}\left(d_t - \hat{d}_t^{\{k\}}\right)^2 + O\left(\frac{\ln K_h}{n}\right), \tag{5}$$

for any n without any knowledge of the optimal $\hat{d}_t^{\{k\}}$, the future of the sequences or the data length n. We emphasize that there exist various extensions of the basic update (4) in the context of tracking the best expert or the best linear combination [21], for different loss functions [22], in a stochastic context [23] and in different frameworks [13]. We emphasize that our derivations directly cover these frameworks.

Note that even for a moderate length pattern such as $h = 5$, we have $K_h \approx 2^{(5/e)^5}$. Hence, in this form the algorithm (3) cannot be directly implemented since we need to run K_h FS predictors in parallel and monitor their performances to construct (3). We next introduce a method that implements (3) with complexity only linear in the pattern length h.

For an efficient implementation of (3), we first note that we can write (4) as

$$w_t^{\{k\}} = \frac{\exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right)}{\sum_{r=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{r\}} e_z\right)}, \tag{6}$$

after some algebra, when $w_0^{\{k\}} = 1/K_h$, $\forall k = 1, \ldots, K_h$, which yields a recursive formulation later in the paper. Before providing the recursive formulation, we observe that although there are K_h FS predictors in the mixture (3) (with different weights), the states used by these FS predictors include a relatively small number of equivalence classes shown in Figure 2. To efficiently use this observation, we

assign a predictor $\hat{c}_t^{\{c_{i,j}\}}$ to each equivalence class in Figure 2. Each equivalence class predictor constructs its output based on the past sequence using a sequential learning algorithm such as the LMS or RLS algorithms.

Each FS predictor, $\hat{d}_t^{\{k\}}$, uses a finite set of these equivalence class predictors, e.g., $\hat{d}_t^{\{1\}}$ with $\{c_{1,1}, c_{1,2}, c_{1,3}\}$ uses the equivalence class predictors $\hat{d}_t^{\{c_{1,1}\}}$, $\hat{d}_t^{\{c_{1,2}\}}$, $\hat{d}_t^{\{c_{1,3}\}}$. If we observe at time t the pattern $(x_{t-2}, x_{t-1}, x_t) = (10.2, 13.5, -1)$ with ordering $(2, 3, 1)$, then $\hat{d}_t^{\{1\}}$ uses the state predictor of $\hat{d}_t^{\{c_{1,2}\}}$ to give its final output as $\hat{d}_t^{\{1\}} = \hat{d}_t^{\{c_{1,2}\}}$, since $(2, 3, 1) \in c_{1,2}$. In this sense, although there are K_h different FS predictors, at any time t, each of these FS predictors uses the output of only one of the m_h different equivalence class predictors, $\hat{d}_t^{\{c_{i,j}\}}$, based on the current pattern.

To calculate (3) and (4), we first show *i)* how the total sum in the denominator of (6) can be recursively calculated. Based on this recursion, we then introduce *ii)* a sequential-in-time update of this sum, which yields the numerator of (6). This recursive formulation and the sequential-in-time update of the denominator of (6) are then used to *iii)* calculate the combined weights in a recursive form with a computational cost only linear in the pattern length h. In the following, we rigorously derive these steps.

i) The total loss in the denominator of (6) is defined as

$$L_t \quad \sum_{k=1}^{K_h} L_t^{\{k\}}, \tag{7}$$

where

$$L_t^{\{k\}} \quad \exp\ -\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z \quad . \tag{8}$$

We also define a function of loss for each equivalence class in Figure 2 as follows

$$L_t^{\{c_{i,j}\}} = \exp\ -\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{c_{i,j}\}} e_z I_z^{\{c_{i,j}\}} \quad , \tag{9}$$

where $I_t^{\{c_{i,j}\}}$ is the indicator function, i.e.,

$$I_t^{\{c_{i,j}\}} \quad \begin{matrix} 1 & \text{, if } (x_{t-h+1}, \ldots, x_t) \in c_{i,j} \\ 0 & \text{, otherwise} \end{matrix} . \tag{10}$$

Each $L_t^{\{k\}}$ can be written as a product of $L_t^{\{\cdot\}}$'s of its equivalence class predictors, e.g.,

$$L_t^{\{1\}} = L_t^{\{c_{1,1}\}} L_t^{\{c_{1,2}\}} L_t^{\{c_{1,3}\}} . \tag{11}$$

Based on this observation, we next use a recursive formulation in order to efficiently calculate the sum L_t.

To recursively calculate L_t, starting from the top of Figure 2 and going downwards to the bottom, we recursively define intermediate sums that represent the total accumulated function of loss up to that point as follows

$$T_t^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}} + \sum_{n \in C^{\{c_{i,j}\}}} T_t^{\{n\}}, \tag{12}$$

for each equivalence class $c_{i,j}$, where $C^{\{c_{i,j}\}}$ represents the upper branches of the equivalence class $c_{i,j}$, e.g., for $c_{0,1}$, $C^{\{c_{0,1}\}} = \{c_{1,1}, c_{1,2}, c_{1,3}\}$. To start the recursion (12) from the top of Figure 2, at level $i = h-1$, we set $T_t^{\{c_{h-1,j}\}} = L_t^{\{c_{h-1,j}\}}$. If we open the recursive formulation for $T_t^{\{c_{0,1}\}}$ using the relations (11) and (12), we get after some algebra $L_t = T_t^{\{c_{0,1}\}}$, i.e., the recursive calculation (12) from top to bottom yields L_t at the bottom.

ii) Suppose after we observe $(x_{t-h+1}, \ldots, x_t)$, we produced our prediction $\hat{d}_t$, the true data d_t is revealed and we get $e_t = d_t - \hat{d}_t$. Our task is now to calculate L_{t+1} from L_t. Naturally, a recursive formulation for L_{t+1} also holds as in (12), where we have $T_{t+1}^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}}$ terms instead of $T_t^{\{c_{i,j}\}}$ and $L_t^{\{c_{i,j}\}}$ terms. However, suppose $(x_{t-h}, \ldots, x_t)$ leads to a particular pattern. Then, from time t to $t+1$, due to the indicator function in (9), only the equivalence classes who match the particular pattern of $(x_{t-h+1}, \ldots, x_t)$ are effected by this update. For all other equivalence classes that do not include this particular pattern, we have $T_{t+1}^{\{c_{i,j}\}} = T_t^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}}$. Hence, we start from the top of the figure and update only $T_t^{\{c_{i,j}\}}$ and $L_t^{\{c_{i,j}\}}$ for h different equivalence classes that include the current pattern in order to get $T_{t+1}^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}}$ as follows.

After we set $T_{t+1}^{\{c_{i,j}\}} = T_t^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}}$ for the unaffected equivalence classes, we start from the top of Figure 2. To continue with our example, suppose we have $(x_{t-2}, x_{t-1}, x_t) = (10.2, 13.5, -1)$, yielding $(2, 3, 1)$. Starting from the top of the figure and considering only the equivalence classes that contain the particular pattern at time t, i.e., for $c_{2,3}$ at level-2, we have

$$\begin{aligned} T_{t+1}^{\{c_{2,3}\}} &= L_{t+1}^{\{c_{2,3}\}} \\ &= L_t^{\{c_{2,3}\}} \exp\ -\mu \hat{d}_t^{\{c_{2,3}\}} e_t \quad , \end{aligned} \tag{13}$$

and continuing to the top, for $c_{1,2}$ at level-1, we have

$$T_{t+1}^{\{c_{1,2}\}} = L_t^{\{c_{1,2}\}} \exp\ -\mu \hat{d}_t^{\{c_{1,2}\}} e_t \ + T_{t+1}^{\{c_{2,3}\}} T_{t+1}^{\{c_{2,4}\}}, \tag{14}$$

and for $c_{0,1}$ at level-0, we have

$$T_{t+1}^{\{c_{0,1}\}} = L_t^{\{c_{0,1}\}} \exp\ -\mu \hat{d}_t^{\{c_{0,1}\}} e_t \ + T_{t+1}^{\{c_{1,1}\}} T_{t+1}^{\{c_{1,2}\}} T_{t+1}^{\{c_{1,3}\}} . \tag{15}$$

Hence, after $h = 3$ updates and recursive calculations, we get $L_{t+1} = T_{t+1}^{\{c_{0,1}\}}$. Based on this recursion we next produce (3).

assign a predictor $\hat{d}_t^{\{c_{i,j}\}}$ to each equivalence class in Figure 2. Each equivalence class predictor constructs its output based on the past sequence using a sequential learning algorithm such as the LMS or RLS algorithms.

Each FS predictor, $\hat{d}_t^{\{k\}}$, uses a finite set of these equivalence class predictors, e.g., $\hat{d}_t^{\{1\}}$ with $\{c_{1,1}, c_{1,2}, c_{1,3}\}$ uses the equivalence class predictors $\hat{d}_t^{\{c_{1,1}\}}$, $\hat{d}_t^{\{c_{1,2}\}}$, $\hat{d}_t^{\{c_{1,3}\}}$. If we observe at time t the pattern $(x_{t-2}, x_{t-1}, x_t) = (10.2, 13.5, -1)$ with ordering $(2, 3, 1)$, then $\hat{d}_t^{\{1\}}$ uses the state predictor of $\hat{d}_t^{\{c_{1,2}\}}$ to give its final output as $\hat{d}_t^{\{1\}} = \hat{d}_t^{\{c_{1,2}\}}$, since $(2, 3, 1) \in c_{1,2}$. In this sense, although there are K_h different FS predictors, at any time t, each of these FS predictors uses the output of only one of the m_h different equivalence class predictors, $\hat{d}_t^{\{c_{i,j}\}}$, based on the current pattern.

To calculate (3) and (4), we first show *i)* how the total sum in the denominator of (6) can be recursively calculated. Based on this recursion, we then introduce *ii)* a sequential-in-time update of this sum, which yields the numerator of (6). This recursive formulation and the sequential-in-time update of the denominator of (6) are then used to *iii)* calculate the combined weights in a recursive form with a computational cost only linear in the pattern length h. In the following, we rigorously derive these steps.

i) The total loss in the denominator of (6) is defined as

$$L_t \triangleq \sum_{k=1}^{K_h} L_t^{\{k\}}, \tag{7}$$

where

$$L_t^{\{k\}} \triangleq \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right). \tag{8}$$

We also define a function of loss for each equivalence class in Figure 2 as follows

$$L_t^{\{c_{i,j}\}} = \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{c_{i,j}\}} e_z I_z^{\{c_{i,j}\}}\right), \tag{9}$$

where $I_t^{\{c_{i,j}\}}$ is the indicator function, i.e.,

$$I_t^{\{c_{i,j}\}} \triangleq \begin{cases} 1 & \text{, if } (x_{t-h+1}, \ldots, x_t) \in c_{i,j} \\ 0 & \text{, otherwise} \end{cases}. \tag{10}$$

Each $L_t^{\{k\}}$ can be written as a product of $L_t^{\{\cdot\}}$'s of its equivalence class predictors, e.g.,

$$L_t^{\{1\}} = L_t^{\{c_{1,1}\}} L_t^{\{c_{1,2}\}} L_t^{\{c_{1,3}\}}. \tag{11}$$

Based on this observation, we next use a recursive formulation in order to efficiently calculate the sum L_t.

To recursively calculate L_t, starting from the top of Figure 2 and going downwards to the bottom, we recursively define intermediate sums that represent the total accumulated function of loss up to that point as follows

$$T_t^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}} + \prod_{n \in C^{\{c_{i,j}\}}} T_t^{\{n\}}, \tag{12}$$

for each equivalence class $c_{i,j}$, where $C^{\{c_{i,j}\}}$ represents the upper branches of the equivalence class $c_{i,j}$, e.g., for $c_{0,1}$, $C^{\{c_{0,1}\}} = \{c_{1,1}, c_{1,2}, c_{1,3}\}$. To start the recursion (12) from the top of Figure 2, at level $i = h - 1$, we set $T_t^{\{c_{h-1,j}\}} = L_t^{\{c_{h-1,j}\}}$. If we open the recursive formulation for $T_t^{\{c_{0,1}\}}$ using the relations (11) and (12), we get after some algebra $L_t = T_t^{\{c_{0,1}\}}$, i.e., the recursive calculation (12) from top to bottom yields L_t at the bottom.

ii) Suppose after we observe $(x_{t-h+1}, \ldots, x_t)$, we produced our prediction $\hat{d}_t$, the true data d_t is revealed and we get $e_t = d_t - \hat{d}_t$. Our task is now to calculate L_{t+1} from L_t. Naturally, a recursive formulation for L_{t+1} also holds as in (12), where we have $T_{t+1}^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}}$ terms instead of $T_t^{\{c_{i,j}\}}$ and $L_t^{\{c_{i,j}\}}$ terms. However, suppose $(x_{t-h}, \ldots, x_t)$ leads to a particular pattern. Then, from time t to $t+1$, due to the indicator function in (9), only the equivalence classes who match the particular pattern of $(x_{t-h+1}, \ldots, x_t)$ are effected by this update. For all other equivalence classes that do not include this particular pattern, we have $T_{t+1}^{\{c_{i,j}\}} = T_t^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}}$. Hence, we start from the top of the figure and update only $T_t^{\{c_{i,j}\}}$ and $L_t^{\{c_{i,j}\}}$ for h different equivalence classes that include the current pattern in order to get $T_{t+1}^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}}$ as follows.

After we set $T_{t+1}^{\{c_{i,j}\}} = T_t^{\{c_{i,j}\}}$ and $L_{t+1}^{\{c_{i,j}\}} = L_t^{\{c_{i,j}\}}$ for the unaffected equivalence classes, we start from the top of Figure 2. To continue with our example, suppose we have $(x_{t-2}, x_{t-1}, x_t) = (10.2, 13.5, -1)$, yielding $(2, 3, 1)$. Starting from the top of the figure and considering only the equivalence classes that contain the particular pattern at time t, i.e., for $c_{2,3}$ at level-2, we have

$$\begin{aligned} T_{t+1}^{\{c_{2,3}\}} &= L_{t+1}^{\{c_{2,3}\}} \\ &= L_t^{\{c_{2,3}\}} \exp\left(-\mu \hat{d}_t^{\{c_{2,3}\}} e_t\right), \end{aligned} \tag{13}$$

and continuing to the top, for $c_{1,2}$ at level-1, we have

$$T_{t+1}^{\{c_{1,2}\}} = L_t^{\{c_{1,2}\}} \exp\left(-\mu \hat{d}_t^{\{c_{1,2}\}} e_t\right) + T_{t+1}^{\{c_{2,3}\}} T_{t+1}^{\{c_{2,4}\}}, \tag{14}$$

and for $c_{0,1}$ at level-0, we have

$$T_{t+1}^{\{c_{0,1}\}} = L_t^{\{c_{0,1}\}} \exp\left(-\mu \hat{d}_t^{\{c_{0,1}\}} e_t\right) + T_{t+1}^{\{c_{1,1}\}} T_{t+1}^{\{c_{1,2}\}} T_{t+1}^{\{c_{1,3}\}}. \tag{15}$$

Hence, after $h = 3$ updates and recursive calculations, we get $L_{t+1} = T_{t+1}^{\{c_{0,1}\}}$. Based on this recursion we next produce (3).

Initialization:

$L_0^{\{c_{i,j}\}} = 1$, calculate $T_0^{\{c_{i,j}\}}$.

Prediction:

For $t = 1, \ldots$ do

Find the current state s_t.

Find $\mathbf{C}$, the set of equivalence classes having s_t.

Calculate $\gamma_{t+1}^{\{c_{i,j}\}}$, $\forall c_{i,j} \in \mathbf{C}$.

Output $\hat{d}_t = \gamma_{t+1}^{\{c_{0,1}\}} / T_t^{\{c_{0,1}\}}$.

Find the error $e_t = d_t - \hat{d}_t$.

Update $L_t^{\{c_{i,j}\}}$, $T_t^{\{c_{i,j}\}}$, $\hat{d}_t^{\{c_{i,j}\}}$ $\forall c_{i,j} \in \mathbf{C}$ as in (13)-(15).

End for

Figure 3: The Pseudo-code of the algorithm.

iii) To finally construct $\hat{d}_t$, we observe from (3) and (6) that

$$\begin{aligned}\hat{d}_t &= \sum_{k=1}^{K_h} \mu_t^{\{k\}} \hat{d}_t^{\{k\}} \\ &= \sum_{k=1}^{K_h} \frac{\exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right)}{\sum_{r=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{r\}} e_z\right)} \hat{d}_t^{\{k\}} \\ &= \frac{\sum_{k=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right) \hat{d}_t^{\{k\}}}{L_t}. \end{aligned} \quad (16)$$

We observe that the numerator of (16) is similar to

$$\begin{aligned} L_{t+1} &= \sum_{k=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t} \hat{d}_z^{\{k\}} e_z\right) \\ &= \sum_{k=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right) \exp\left(-\mu \hat{d}_t^{\{k\}} e_t\right), \end{aligned} \quad (17)$$

where only the last exponential terms in (17) are replaced by $\hat{d}_t^{\{k\}}$ terms. Hence, we can use the recursion from L_t to L_{t+1} to efficiently calculate the numerator of (16).

To continue with our example, starting from the top of Figure 2, if we replace the exponential term $\exp\left(-\mu \hat{d}_t^{\{c_{2,3}\}} e_t\right)$ in (13) with $\hat{d}_t^{\{c_{2,3}\}}$, then we get

$$\gamma_{t+1}^{\{c_{2,3}\}} = L_t^{\{c_{2,3}\}} \hat{d}_t^{\{c_{2,3}\}}, \quad (18)$$

similarly from (14), we have

$$\gamma_{t+1}^{\{c_{1,2}\}} = L_t^{\{c_{1,2}\}} \hat{d}_t^{\{c_{1,2}\}} + \gamma_{t+1}^{(c_{2,3})} T_{t+1}^{\{c_{2,4}\}}, \quad (19)$$

and from (15), we get

$$\gamma_{t+1}^{(c_{0,1})} = L_t^{(c_{0,1})} \hat{d}_t^{(c_{0,1})} + T_{t+1}^{(c_{1,1})} \gamma_{t+1}^{\{c_{1,2}\}} T_{t+1}^{(c_{1,3})}, \quad (20)$$

which yields the numerator of (16)

$$\gamma_{t+1}^{(c_{0,1})} = \sum_{k=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right) \hat{d}_t^{\{k\}}, \quad (21)$$

after h recursive calculations.

This concludes the derivation of the algorithm and the pseudocode of the algorithm can be found in Figure 3.

IV. Simulations

In this section, we illustrate the merits of the proposed algorithm with numerical examples.

First, we consider the FS model in Figure 1, where $h = 3$, with transition probabilities

$$P = \begin{bmatrix} 0.4 & 0 & 0.3 & 0.3 & 0 & 0 \\ 0.4 & 0 & 0.3 & 0.3 & 0 & 0 \\ 0 & 0.3 & 0 & 0 & 0.3 & 0.4 \\ 0 & 0.3 & 0 & 0 & 0.3 & 0.4 \\ 0.4 & 0 & 0.3 & 0.3 & 0 & 0 \\ 0 & 0.3 & 0 & 0 & 0.3 & 0.4 \end{bmatrix}, \quad (22)$$

where P_{ij}, denoting the i-th row and j-th column of the matrix P, represents the transition probability from state $c_{2,i}$ to state $c_{2,j}$, according to the state definitions in Figure 2, i.e., $c_{2,1} = (1, 2, 3)$, $c_{2,2} = (2, 1, 3)$, and so on. The transition probabilities in (22) are chosen such that a non-trivial learning task would occur, where all states are consistently traversed. According to the transition probabilities in (22), we arbitrarily generated a sequence of length 2500, where the aim is to predict the trend of the sequence such that $d_t = 1$ if $x_{t+1} \geq x_t$ and $d_t = -1$, otherwise, i.e., we try to predict whether the sequence increases or decreases at time $t+1$ compared to x_t.

In Figure 4, the accumulated square error performances (normalized with time) of the proposed algorithms are compared, where "Universal" represents the universal predictor introduced in this paper, "Finest" represents the finest predictor for $h = 3$, i.e., the predictor using all $S = \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$ as its states, "Coarsest" represents the coarsest predictor, i.e., the predictor with only one state $\{\cdot, \cdot, \cdot)\}$, and "Batch" represents the optimal batch predictor that knows the transition probabilities in (22) even before the processing starts. Note that while the predictors "Universal", "Finest", and "Coarsest" learns the individual state predictors and weights in an online manner, the "Batch" predictor already knows the optimal parameters, i.e., does not perform any learning. As expected from (5), the performance of the "Universal" predictor is as well as the "Coarsest" predictor when there is not sufficient amount of data to train finer equivalence classes. Furthermore, although as the data length increases, the performance of the "Coarsest" predictor deteriorates with respect to predictors having finer equivalence classes, the "Universal" predictor still performs as well as the "Finest" predictor even after a significant amount of observations.

We emphasize that as the pattern order h increases or the underlying data is highly nonstationary, the convergence performance of the "Universal" predictor will significantly outperform the performance of the "Finest" predictor since the "Finest" predictor may not be able to observe enough training sequences to achieve a satisfactory performance. This result is also apparent in Figure 4, where over short data sequences the performance of the "Finest" predictor

Initialization:
$L_0^{\{c_{i,j}\}} = 1$, calculate $T_0^{\{c_{i,j}\}}$.
Prediction:
For $t = 1, \ldots$ do
Find the current state s_t.
Find $\mathcal{C}$, the set of equivalence classes having s_t.
Calculate $\widetilde{T}_{t+1}^{\{c_{i,j}\}}$, $\forall c_{i,j} \in \mathcal{C}$.
Output $\hat{d}_t = \widetilde{T}_{t+1}^{\{c_{0,1}\}} / T_t^{\{c_{0,1}\}}$.
Find the error $e_t = d_t - \hat{d}_t$.
Update $L_t^{\{c_{i,j}\}}$, $T_t^{\{c_{i,j}\}}$, $\hat{d}_t^{\{c_{i,j}\}}$ $\forall c_{i,j} \in \mathcal{C}$ as in (13)-(15).
End for

Figure 3: The Pseudo-code of the algorithm.

iii) To finally construct $\hat{d}_t$, we observe from (3) and (6) that

$$\begin{aligned}\hat{d}_t &= \sum_{k=1}^{K_h} \mu_t^{\{k\}} \hat{d}_t^{\{k\}} \\ &= \sum_{k=1}^{K_h} \frac{\exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right)}{\sum_{r=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{r\}} e_z\right)} \hat{d}_t^{\{k\}} \\ &= \frac{\sum_{k=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right) \hat{d}_t^{\{k\}}}{L_t}. \end{aligned} \quad (16)$$

We observe that the numerator of (16) is similar to

$$\begin{aligned} L_{t+1} &= \sum_{k=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t} \hat{d}_z^{\{k\}} e_z\right) \\ &= \sum_{k=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right) \exp\left(-\mu \hat{d}_t^{\{k\}} e_t\right), \end{aligned} \quad (17)$$

where only the last exponential terms in (17) are replaced by $\hat{d}_t^{\{k\}}$ terms. Hence, we can use the recursion from L_t to L_{t+1} to efficiently calculate the numerator of (16).

To continue with our example, starting from the top of Figure 2, if we replace the exponential term $\exp\left(-\mu \hat{d}_t^{\{c_{2,3}\}} e_t\right)$ in (13) with $\hat{d}_t^{\{c_{2,3}\}}$, then we get

$$\widetilde{T}_{t+1}^{\{c_{2,3}\}} = L_t^{\{c_{2,3}\}} \hat{d}_t^{\{c_{2,3}\}}, \quad (18)$$

similarly from (14), we have

$$\widetilde{T}_{t+1}^{\{c_{1,2}\}} = L_t^{\{c_{1,2}\}} \hat{d}_t^{\{c_{1,2}\}} + \widetilde{T}_{t+1}^{(c_{2,3})} T_{t+1}^{\{c_{2,4}\}}, \quad (19)$$

and from (15), we get

$$\widetilde{T}_{t+1}^{(c_{0,1})} = L_t^{(c_{0,1})} \hat{d}_t^{(c_{0,1})} + T_{t+1}^{(c_{1,1})} \widetilde{T}_{t+1}^{\{c_{1,2}\}} T_{t+1}^{(c_{1,3})}, \quad (20)$$

which yields the numerator of (16)

$$\widetilde{T}_{t+1}^{(c_{0,1})} = \sum_{k=1}^{K_h} \exp\left(-\mu \sum_{z=1}^{t-1} \hat{d}_z^{\{k\}} e_z\right) \hat{d}_t^{\{k\}}, \quad (21)$$

after h recursive calculations.

This concludes the derivation of the algorithm and the pseudocode of the algorithm can be found in Figure 3.

IV. Simulations

In this section, we illustrate the merits of the proposed algorithm with numerical examples.

First, we consider the FS model in Figure 1, where $h = 3$, with transition probabilities

$$P = \begin{bmatrix} 0.4 & 0 & 0.3 & 0.3 & 0 & 0 \\ 0.4 & 0 & 0.3 & 0.3 & 0 & 0 \\ 0 & 0.3 & 0 & 0 & 0.3 & 0.4 \\ 0 & 0.3 & 0 & 0 & 0.3 & 0.4 \\ 0.4 & 0 & 0.3 & 0.3 & 0 & 0 \\ 0 & 0.3 & 0 & 0 & 0.3 & 0.4 \end{bmatrix}, \quad (22)$$

where P_{ij}, denoting the i-th row and j-th column of the matrix P, represents the transition probability from state $c_{2,i}$ to state $c_{2,j}$, according to the state definitions in Figure 2, i.e., $c_{2,1} = (1, 2, 3)$, $c_{2,2} = (2, 1, 3)$, and so on. The transition probabilities in (22) are chosen such that a non-trivial learning task would occur, where all states are consistently traversed. According to the transition probabilities in (22), we arbitrarily generated a sequence of length 2500, where the aim is to predict the trend of the sequence such that $d_t = 1$ if $x_{t+1} \geq x_t$ and $d_t = \quad 1$, otherwise, i.e., we try to predict whether the sequence increases or decreases at time $t + 1$ compared to x_t.

In Figure 4, the accumulated square error performances (normalized with time) of the proposed algorithms are compared, where "Universal" represents the universal predictor introduced in this paper, "Finest" represents the finest predictor for $h = 3$, i.e., the predictor using all $\mathcal{S} = \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$ as its states, "Coarsest" represents the coarsest predictor, i.e., the predictor with only one state $\{\cdot, \cdot, \cdot)\}$, and "Batch" represents the optimal batch predictor that knows the transition probabilities in (22) even before the processing starts. Note that while the predictors "Universal", "Finest", and "Coarsest" learns the individual state predictors and weights in an online manner, the "Batch" predictor already knows the optimal parameters, i.e., does not perform any learning.

As expected from (5), the performance of the "Universal" predictor is as well as the "Coarsest" predictor when there is not sufficient amount of data to train finer equivalence classes. Furthermore, although as the data length increases, the performance of the "Coarsest" predictor deteriorates with respect to predictors having finer equivalence classes, the "Universal" predictor still performs as well as the "Finest" predictor even after a significant amount of observations.

We emphasize that as the pattern order h increases or the underlying data is highly nonstationary, the convergence performance of the "Universal" predictor will significantly outperform the performance of the "Finest" predictor since the "Finest" predictor may not be able to observe enough training sequences to achieve a satisfactory performance. This result is also apparent in Figure 4, where over short data sequences the performance of the "Finest" predictor

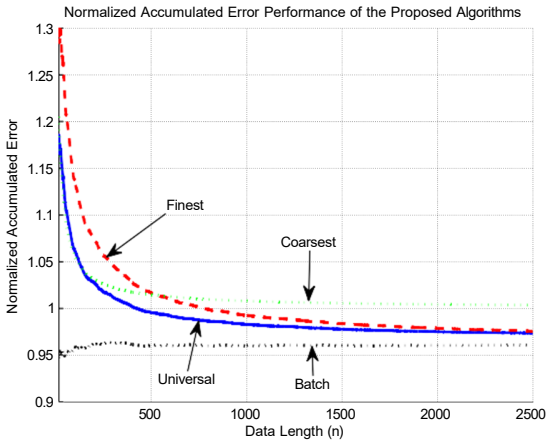

Figure 4: Normalized accumulated squared error performance of the proposed algorithms for the FS model in (22) averaged over 25 trials.

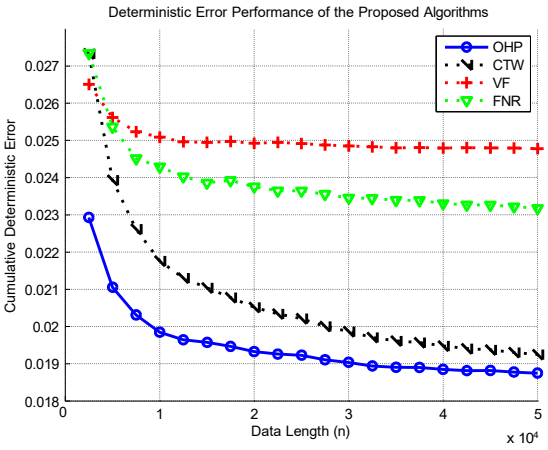

Figure 5: Normalized accumulated squared error performance of the proposed algorithms for the Mackey-Glass time series generated by the fourth order Runge-Kutta method using (23).

is worse compared to the "Universal" and the "Coarsest" predictor. Hence, the universal algorithm outperforms the constituent FS predictors by exploiting the time-dependent nature of the best choice among constituent FS predictors that are defined on the hierarchical structure.

We next consider the performance of our algorithm for Mackey-Glass time series. The Mackey-Glass time series are generated according to the following time delay differential equation

$$\frac{dx}{dt} = \beta \frac{x_\tau}{1 + x_\tau^n} - \gamma x, \tag{23}$$

where x_τ represents the value of the variable x at time $t-\tau$ and we set $\gamma = 1$, $\beta = 2$, $\tau = 2$, and $n = 9.65$ starting from an initial condition of $x = 0.5$ for $t < 0$ in order to generate the well-known chaotic response of the Mackey-Glass sequence. To generate the time series, we use the fourth order Runge-Kutta method. The generated time series are then normalized between $[-1, 1]$ for a fair comparison between the proposed algorithms.

In Figure 5, we compare the performances of the online hierarchical predictor (OHP) proposed in this paper, the context tree weighting (CTW) algorithm presented in [17], the Volterra filter (VF) [24], and the Fourier nonliner regressor (FNR) presented in [25]. Note that the computational complexities of the VF and FNR algorithms are quadratic in the order of the filters, whereas the computational complexities of the OHP and CTW algorithms are linear in the depth of the hierarchy. Therefore, for a fair performance comparison among these algorithms, we use the second order VF and FNR algorithms and the hierarchy depth is set to 4 for the OHP and CTW algorithms. We use RLS algorithm to train the VF and FNR algorithms as well as the equivalence class predictors of the OHP and CTW algorithms, where equivalence classes are defined as the partitions of the space of the sequence history.

Figure 5 illustrates that the OHP and CTW algorithms achieve a better convergence performance compared to the VF and FNR algorithms. This shows the modeling accuracy of the proposed methods. On the other hand, the OHP algorithm converges significantly faster with respect to the CTW algorithm. This follows from the incorporation of the EG algorithm into our framework, whereas the CTW algorithm uses universal weights, which are derived according to information theoretic setups instead of directly minimizing the accumulated squared error. As can be seen from Figure 5, even for a remarkably long sequence of data, the performance of the OHP algorithm is superior to the performance of the CTW algorithm.

We then consider the performance of our algorithm for the data generated by the Lorenz attractor [26]. This data is generated by the following three ordinary differential equations

$$\frac{dx}{dt} = \sigma(y - x) \tag{24}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{25}$$

$$\frac{dz}{dt} = xy - \beta z, \tag{26}$$

where we set $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$ to generate the chaotic behavior. In this experiment, x_t is selected as the desired data and the two dimensional region represented by (y_t, z_t) is selected as the regressor space. We use the same experiment setup as in the previous experiment and compare the performance of our algorithm with respect to the same algorithms.

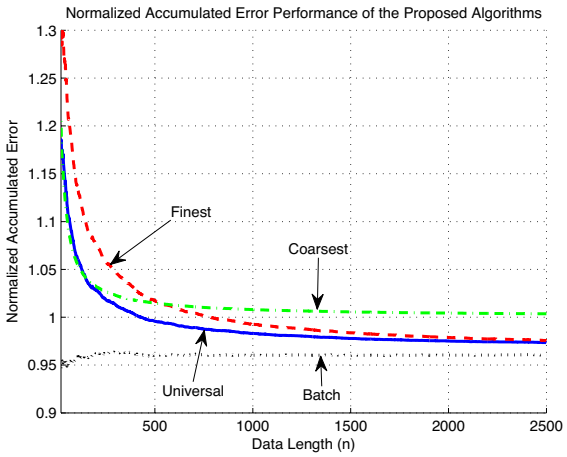

Figure 4: Normalized accumulated squared error performance of the proposed algorithms for the FS model in (22) averaged over 25 trials.

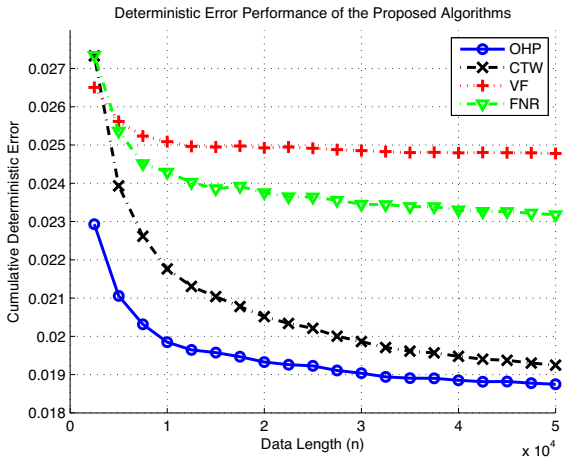

Figure 5: Normalized accumulated squared error performance of the proposed algorithms for the Mackey-Glass time series generated by the fourth order Runge-Kutta method using (23).

is worse compared to the "Universal" and the "Coarsest" predictor. Hence, the universal algorithm outperforms the constituent FS predictors by exploiting the time-dependent nature of the best choice among constituent FS predictors that are defined on the hierarchical structure.

We next consider the performance of our algorithm for Mackey-Glass time series. The Mackey-Glass time series are generated according to the following time delay differential equation

$$\frac{dx}{dt} = \beta \frac{x_\tau}{1 + x_\tau^n} - \gamma x, \tag{23}$$

where x_τ represents the value of the variable x at time $t-\tau$ and we set $\gamma = 1$, $\beta = 2$, $\tau = 2$, and $n = 9.65$ starting from an initial condition of $x = 0.5$ for $t < 0$ in order to generate the well-known chaotic response of the Mackey-Glass sequence. To generate the time series, we use the fourth order Runge-Kutta method. The generated time series are then normalized between $[-1, 1]$ for a fair comparison between the proposed algorithms.

In Figure 5, we compare the performances of the online hierarchical predictor (OHP) proposed in this paper, the context tree weighting (CTW) algorithm presented in [17], the Volterra filter (VF) [24], and the Fourier nonliner regressor (FNR) presented in [25]. Note that the computational complexities of the VF and FNR algorithms are quadratic in the order of the filters, whereas the computational complexities of the OHP and CTW algorithms are linear in the depth of the hierarchy. Therefore, for a fair performance comparison among these algorithms, we use the second order VF and FNR algorithms and the hierarchy depth is set to 4 for the OHP and CTW algorithms. We use RLS algorithm to train the VF and FNR algorithms as well as the equivalence class predictors of the OHP and CTW algorithms, where equivalence classes are defined as the partitions of the space of the sequence history.

Figure 5 illustrates that the OHP and CTW algorithms achieve a better convergence performance compared to the VF and FNR algorithms. This shows the modeling accuracy of the proposed methods. On the other hand, the OHP algorithm converges significantly faster with respect to the CTW algorithm. This follows from the incorporation of the EG algorithm into our framework, whereas the CTW algorithm uses universal weights, which are derived according to information theoretic setups instead of directly minimizing the accumulated squared error. As can be seen from Figure 5, even for a remarkably long sequence of data, the performance of the OHP algorithm is superior to the performance of the CTW algorithm.

We then consider the performance of our algorithm for the data generated by the Lorenz attractor [26]. This data is generated by the following three ordinary differential equations

$$\frac{dx}{dt} = \sigma(y - x) \tag{24}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{25}$$

$$\frac{dz}{dt} = xy - \beta z, \tag{26}$$

where we set $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$ to generate the chaotic behavior. In this experiment, x_t is selected as the desired data and the two dimensional region represented by (y_t, z_t) is selected as the regressor space. We use the same experiment setup as in the previous experiment and compare the performance of our algorithm with respect to the same algorithms.

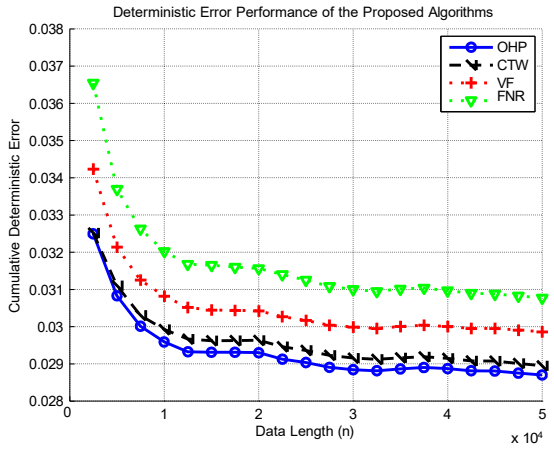

Figure 6: Normalized accumulated squared error performance of the proposed algorithms for the time series generated by the Lorenz attractor given in (24)-(26), averaged over 20 trials.

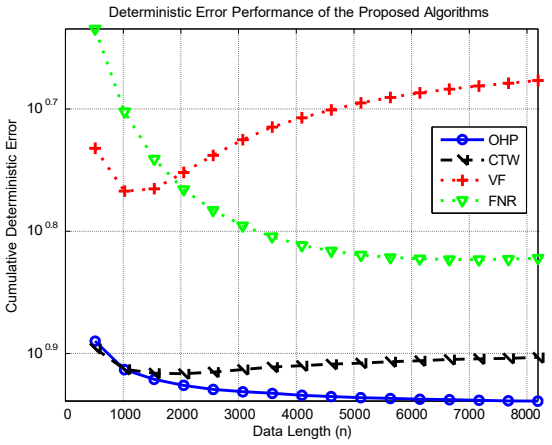

Figure 7: Normalized accumulated squared error performance of the proposed algorithms for the pumadyn data set, averaged over 100 trials.

In Figure 6, we provide the normalized accumulated errors of the proposed algorithms for the Lorenz attractor generated by (24)-(26) averaged over 20 trials (each corresponding to a random initialization). Figure 6 illustrates that the proposed algorithm significantly outperforms the competitor algorithms as it attains a smaller average accumulated error. This follows from the adaptation capability of our algorithm to dynamic changes in the underlying model.

As the last experiment, we consider the prediction of a benchmark data set, namely the pumadyn data set. Among its variants, we have used the one that is highly nonlinear and moderate noisy. The task in this data set is to predict the angular acceleration of one of the robot arm's links using inputs such as the angular positions, velocities and torques of the robot arm. We note that each of these parameters are normalized between $[-1, 1]$ for a fair performance comparison between the competitor algorithms. In Fig. 7, we present the normalized accumulated errors of the proposed algorithms for this data set averaged over 100 independent trials, where in each trial we randomly permuted the data sequence. As can be seen from the figure, the proposed OHP algorithm captures the salient characteristics of the underlying data better than its well-known alternatives in the literature. This experiment illustrates that the proposed algorithm can be efficiently used in real life applications.

V. Conclusion

In this paper, we study sequential FS predictors for real valued sequences, where we use the relative order- ing patterns of the sequence history to construct states. Instead of directly using the relative ordering patterns of the sequence history, which can result a prohibitively large number of states for even moderate length patterns, we define hierarchical equivalence classes by recursively tying certain patterns to avoid over training problems. With this hierarchical equivalence class definitions, we construct a huge number of FS predictors, one of which is optimal for the underlying task at hand in an individual sequence manner (and the optimal predictor can change in time). By using the EG algorithm, we show that we can sequentially achieve the performance of the best sequential FS predictor that can be defined on this hierarchical structure, i.e., the best sequential FS predictor out of $2^{(h/e)^h}$ possible FS predictors, with computational complexity only linear in the length of the pattern h. Our results are generic such that they can be directly used for a wide range of hierarchical equivalence class definitions (e.g., instead of the location of the largest element in the context, one can use other methods to tie relative ordering patterns) or hold for a wide range of loss functions [22].

References

[1] D. Ruta, "Automated trading with machine learning on big data," in *2014 IEEE International Congress on Big Data (BigData Congress)*, June 2014, pp. 824–830.

[2] A. Baheti and D. Toshniwal, "Trend analysis of time series data using data mining techniques," in *2014 IEEE International Congress on Big Data (BigData Congress)*, June 2014, pp. 430–437.

[3] J. Huang, Z. Kalbarczyk, and D. M. Nicol, "Knowledge discovery from big data for intrusion detection using LDA," in *2014 IEEE International Congress on Big Data (BigData Congress)*, June 2014, pp. 760–761.

[4] Y. Song, A. Sailer, and H. Shaikh, "Hierarchical online problem classification for IT support services," *IEEE Transactions*

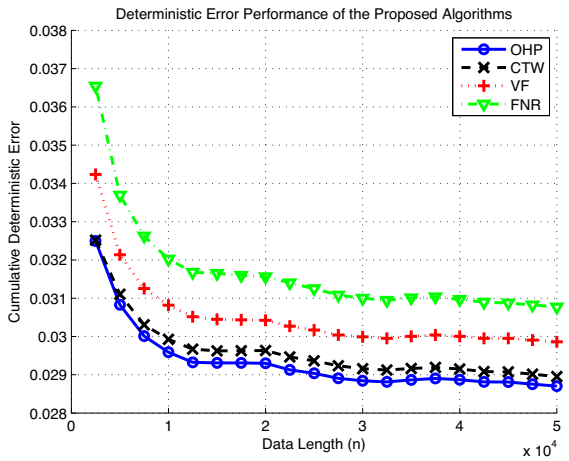

Figure 6: Normalized accumulated squared error performance of the proposed algorithms for the time series generated by the Lorenz attractor given in (24)-(26), averaged over 20 trials.

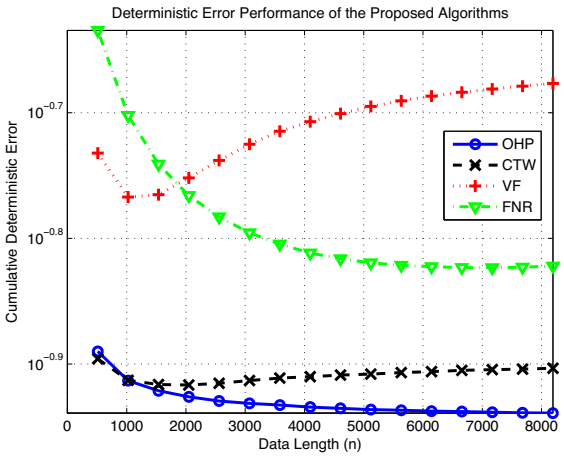

Figure 7: Normalized accumulated squared error performance of the proposed algorithms for the pumadyn data set, averaged over 100 trials.

In Figure 6, we provide the normalized accumulated errors of the proposed algorithms for the Lorenz attractor generated by (24)-(26) averaged over 20 trials (each corresponding to a random initialization). Figure 6 illustrates that the proposed algorithm significantly outperforms the competitor algorithms as it attains a smaller average accumulated error. This follows from the adaptation capability of our algorithm to dynamic changes in the underlying model.

As the last experiment, we consider the prediction of a benchmark data set, namely the pumadyn data set. Among its variants, we have used the one that is highly nonlinear and moderate noisy. The task in this data set is to predict the angular acceleration of one of the robot arm's links using inputs such as the angular positions, velocities and torques of the robot arm. We note that each of these parameters are normalized between $[-1, 1]$ for a fair performance comparison between the competitor algorithms. In Fig. 7, we present the normalized accumulated errors of the proposed algorithms for this data set averaged over 100 independent trials, where in each trial we randomly permuted the data sequence. As can be seen from the figure, the proposed OHP algorithm captures the salient characteristics of the underlying data better than its well-known alternatives in the literature. This experiment illustrates that the proposed algorithm can be efficiently used in real life applications.

V. Conclusion

In this paper, we study sequential FS predictors for real valued sequences, where we use the relative ordering patterns of the sequence history to construct states. Instead of directly using the relative ordering patterns of the sequence history, which can result a prohibitively large number of states for even moderate length patterns, we define hierarchical equivalence classes by recursively tying certain patterns to avoid over training problems. With this hierarchical equivalence class definitions, we construct a huge number of FS predictors, one of which is optimal for the underlying task at hand in an individual sequence manner (and the optimal predictor can change in time). By using the EG algorithm, we show that we can sequentially achieve the performance of the best sequential FS predictor that can be defined on this hierarchical structure, i.e., the best sequential FS predictor out of $2^{(h/e)^h}$ possible FS predictors, with computational complexity only linear in the length of the pattern h. Our results are generic such that they can be directly used for a wide range of hierarchical equivalence class definitions (e.g., instead of the location of the largest element in the context, one can use other methods to tie relative ordering patterns) or hold for a wide range of loss functions [22].

References

[1] D. Ruta, "Automated trading with machine learning on big data," in *2014 IEEE International Congress on Big Data (BigData Congress)*, June 2014, pp. 824–830.

[2] A. Baheti and D. Toshniwal, "Trend analysis of time series data using data mining techniques," in *2014 IEEE International Congress on Big Data (BigData Congress)*, June 2014, pp. 430–437.

[3] J. Huang, Z. Kalbarczyk, and D. M. Nicol, "Knowledge discovery from big data for intrusion detection using LDA," in *2014 IEEE International Congress on Big Data (BigData Congress)*, June 2014, pp. 760–761.

[4] Y. Song, A. Sailer, and H. Shaikh, "Hierarchical online problem classification for IT support services," *IEEE Transactions*

on Services Computing, vol. 5, no. 3, pp. 345–357, Third 2012.

[5] Y. Achbany, I. J. Jureta, S. Faulkner, and F. Fouss, "Continually learning optimal allocations of services to tasks," *IEEE Transactions on Services Computing*, vol. 1, no. 3, pp. 141–154, July 2008.

[6] G. Cassar, P. Barnaghi, and K. Moessner, "Probabilistic matchmaking methods for automated service discovery," *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 654–666, Oct 2014.

[7] Z. Obrenovic and D. Gasevic, "End-user service computing: Spreadsheets as a service composition tool," *IEEE Transactions on Services Computing*, vol. 1, no. 4, pp. 229–242, Oct 2008.

[8] H. K. Kim, J. K. Kim, and Y. U. Ryu, "Personalized recommendation over a customer network for ubiquitous shopping," *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 140–151, April 2009.

[9] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-aware autonomic resource allocation in multitier virtualized environments," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 2–19, Jan 2012.

[10] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 497–511, Fourth 2012.

[11] J. Kim, P. Eades, R. Fleischer, S.-H. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, and T. Tokuyama, "Order-preserving matching," *Theoretical Computer Science*, vol. 525, no. 0, pp. 68–79, 2014.

[12] M. Kubica, T. Kulczyski, J. Radoszewski, W. Rytter, and T. Wale, "A linear time algorithm for consecutive permutation pattern matching," *Information Processing Letters*, vol. 113, no. 12, pp. 430 – 433, 2013.

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

[14] N. D. Vanli and S. S. Kozat, "A unified approach to universal prediction: Generalized upper and lower bounds," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 3, pp. 646–651, March 2015.

[15] 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.

[16] N. D. Vanli and S. S. Kozat, "A comprehensive approach to universal piecewise nonlinear regression based on trees," *IEEE Transactions on Signal Processing*, vol. 62, no. 20, pp. 5471–5486, Oct 2014.

[17] 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.

[18] M. Feder, N. Merhav, and M. Gutman, "Universal prediction of individual sequences," *IEEE Trans. on Info. Theory*, vol. 38, pp. 1258–1270, 1992.

[19] J. Kivinen and M. K. Warmuth, "Exponentiated gradient versus gradient descent for linear predictors," *Journal of Information and Computation*, vol. 132, no. 1, pp. 1–62, 1997.

[20] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*. Englewood Cliffs, NJ: Prentice-Hall, 1978.

[21] M. Herbser and M. K. Warmuth, "Tracking the best expert," in *Proceedings of International Conference on Machine Learning*, 1995, pp. 286–294.

[22] D. Haussler, J. Kivinen, and M. K. Warmuth, "Sequential prediction of individual sequences under general loss functions," *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 1906–1925, 1998.

[23] 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, no. 8, pp. 4050–4063, August 2010.

[24] V. J. Mathews, "Adaptive polynomial filters," *Signal Processing Magazine, IEEE*, vol. 8, no. 3, pp. 10–26, 1991.

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

[26] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20, no. 2, pp. 130–141, 1963.