Efficient Online Sequence Prediction with Side Information

Han Xiao
Institute of Informatics
Technische Universität München, Germany
Email: xiaoh@in.tum.de

Claudia Eckert
Institute of Informatics
Technische Universität München, Germany
Email: claudia.eckert@in.tum.de

***Abstract*—Sequence prediction is a key task in machine learning and data mining. It involves predicting the next symbol in a sequence given its previous symbols. Our motivating application is predicting the execution path of a process on an operating system in real-time. In this case, each symbol in the sequence represents a system call accompanied with arguments and a return value. We propose a novel online algorithm for predicting the next system call by leveraging both context and side information. The online update of our algorithm is efficient in terms of time cost and memory consumption. Experiments on real-world data sets showed that our method outperforms state-of-the-art online sequence prediction methods in both accuracy and efficiency, and incorporation of side information does significantly improve the predictive accuracy.**

I. Introduction

Online sequence prediction is the problem of observing a sequence of symbols one at a time and predicting the next symbol before it is revealed. This technique has been successfully applied in a large variety of disciplines, such as stock market analysis, natural language processing and DNA sequencing. The problem of sequence prediction has received considerable attention throughout the years in information theory, machine learning and data mining. Typically, the *Markov property* is assumed when modeling a sequence. That is, a finite history of the past, i.e. the *context*, can be useful in predicting the future. The length of the context is called the *order* of Markov models. Previous work shows ample evidence of the fact that making such an assumption is often reasonable in a practical sense [1], [2]. For instance in natural language processing, it is often well-enough to describe text by a fixed order Markov models (e.g. bigram, trigram), though the next word is not necessarily related to its previous words.

Our motivating application is modeling the execution path of a process on a desktop/mobile system in real-time. Each process produces an ordered sequence of system calls which request different services from the operating system. An illustrative example is depicted in Fig. 1.

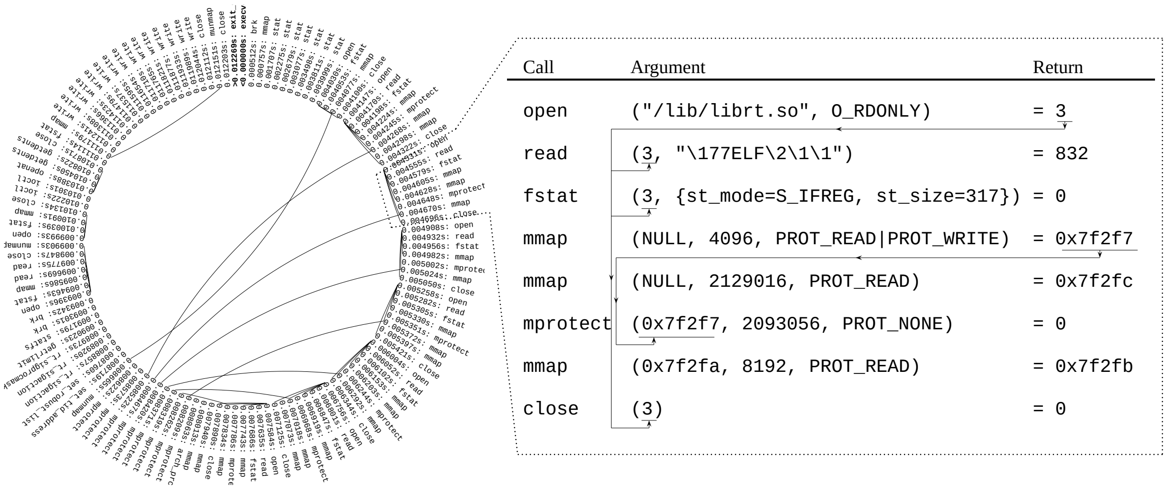
Three remarks are in order. First, some system calls have a long range dependency. For instance, after creating a file the process may produce hundreds of system calls before it finally closes the file. In this case, the dependency between `creat` and `close` can not be observed from a short context of `close`. Although one can increase the order of Markov models to capture information from a long distant context, it is often difficult in practice due to the requirement of vast amounts of training data and more sophisticated smoothing algorithms [3]. In general, the length of context needed to make an accurate prediction is not constant, but rather depends on the recently executed system calls. Second, the information from the arguments and return values (e.g. file descriptor, memory address and signal) may be also indicative in predicting the next system call. Considering a process repetitively reads data from the file `1` and writes data to the file `2`. A resulting system call sequence may look like

```
open(1), read(1), close(1), open(2),
write(2), close(2), open(1), ...
```

Assume that we have observed the above sequence with seven system calls; the goal is to predict the next system call. Without using the knowledge of the arguments, a bigram model based merely on the name of adjacent system call will predict `read` and `write` with even chance. However, as the file `2` has been closed, the correct prediction should be `read`. Although one can solve this problem by extending Markov models with more sophisticated graphical models, incorporating side information is in general not straightforward for probabilistic Markov models. Third, a process may exhibit different behaviors at various points during its lifetime, depending on user's input and the status of the system. In other word, the sequence is usually not stationary and no prior assumption on its distribution should be made. This suggests the necessity of an online model that can be continuously updated, preserving information from a long distant context while giving more emphasis to recent data, so that the stationarity is not required.

We focus on the problem of predicting the next system call given an observed sequence. The solution of this problem can be extremely useful in a wide range of applications, such as anomaly detection [4], [5], buffer cache management in operating system [6], power management in smartphones [7] and sandbox systems [8]. We leverage both context and side information of each system call and model a sequence in an online fashion. The proposed algorithm performs prediction in real-time and can quickly update the model when a prediction error is made.

The rest of the paper is organized as follows. Section II briefly reviews previous work on sequence prediction. Subsequently, our novel contribution is highlighted. Section III describes the problem formulation. We next cast sequence prediction as a linear separation problem in Section IV. The proposed method is presented in Section V. Experimental results are demonstrated in Section VI. Section VII concludes the paper and points out some future directions.



Call	Argument	Return
open	("/lib/librt.so", O_RDONLY)	= 3
read	(3, "\177ELF\2\1\1")	= 832
fstat	(3, {st_mode=S_IFREG, st_size=317})	= 0
mmap	(NULL, 4096, PROT_READ|PROT_WRITE)	= 0x7f2f7
mmap	(NULL, 2129016, PROT_READ)	= 0x7f2fc
mprotect	(0x7f2f7, 2093056, PROT_NONE)	= 0
mmap	(0x7f2fa, 8192, PROT_READ)	= 0x7f2fb
close	(3)	= 0

Fig. 1. Left panel shows a circular plot of a system call trace when running `ls` on Linux, which was collected using `strace`. System calls are plotted clockwise, starting with `execve` and ending with `exit` on top. A time stamp is labeled in front of each system call. A curve connects two system calls if the return value of the former was used as an argument of the latter. On the right panel, a sample segment of this sequence is detailed, with argument defined within the parentheses. For the sake of clarity, some long arguments (e.g. string) are omitted. The dependencies between the return value and argument are highlighted with arrow lines.

II. Related Work

The problem of sequence prediction has a fairly long history and has received much attention from the field of game theory [9], [10], [11], information theory [12], [13], [14], [15], and machine learning [16], [17], [18], [19], [20], [21]. One of the most useful tools is context trees, which store informative histories and the probability of the next symbol given these [15], [22]. Context trees use only a few recently observed symbols for prediction. The number of symbols that are used depends on the specific context in which the prediction is made. The motivation for exploring context tree strategies stems from their simplicity and their success in lossless data compression applications [23]. Another family of approach based on Bayesian nonparametric models has generated considerable recent research interest [24], [19], [20]. It is assumed that the distribution of the current symbol is determined by some random process (e.g. Dirichlet process, Pitman-Yor process) governed by its context. The hierarchy is defined recursively to the first symbol in the sequence, on which a global base distribution is defined. These models give state-of-the-art performance in language modeling, however, inference in such models is not straightforward. It often relies on repeated random sampling (e.g. Markov chain Monte Carlo), which can be time-consuming in practice.

Another related line of work is online learning, which takes place in a sequence of consecutive rounds. On each round, the learner is given a question and is required to provide an answer to this question. The performance of an online learning algorithm is measured by the cumulative loss suffered by the prediction along the run on a sequence of question-answer pairs. The Perceptron algorithm [25], [26], [27] is perhaps the first and simplest online learning algorithm designed for answering yes/no questions. Adaptations of the Perceptron for multiclass categorization tasks include [28], [29]. As the Perceptron algorithm is essentially a gradient descent (first-order) method, recent years have seen a surge of studies on the second-order online learning [30], [31], [32]. For example, the confidence-weighted algorithm [32] maintains a Gaussian distribtuion over some linear classifier hypotheses and employs it to control the online update of parameters. Several work has followed this idea and showed that parameters' confidence information can significantly improve online learning performance [32], [33], [34], [35].

The system call sequence was mainly studied by computer security researchers in the early days [36], [37], [4]. They used patterns in the sequence to identify misuses and intrusions in systems. To contain the attack preemptively, plan recognition was developed, aiming at recognizing and predicting goals based on observed system call sequences [38], [39]. Recently, the problem of system call prediction attracted much attention due to its importance in many applications. For example, in a sandbox the amount of time that a process must suspend for a security check can be eliminated when the current system call is correctly predicted, yielding a more efficient sandbox implementation [8]. On mobile devices it has been demonstrated that system calls prediction can be used to design user-oriented prefetching techniques [6] and reduce power consumption [7]. However, most of these studies are over-simplistic in the sense that they focused only on the names of system calls and overlooked the arguments and return values. One possible reason is the difficulty in representing this side information, which requires a different modeling technique, such as rule learning [40], [41]. Hence, it can not be incorporated into a sequence prediction model in a straightforward manner.

In this paper we introduce a novel online algorithm for predicting system calls in a sequence. Our algorithm combines the ideas from both context trees [18], [42] and second-order online learning algorithms [32], [33], [34], [35]. Unlike previous work on system call prediction that only uses context information, we also consider side information such as arguments, return values and structures into learning and prediction. The side information can be straightforwardly incorporated into our model, giving a further boost to the accuracy of prediction. Furthermore, we propose several techniques to improve the efficiency (in terms of both time and memory) of our algorithm on long sequences, yielding a good scalabilty on big data.

III. Problem Formulation

We denote the alphabet of the observed symbols as $\Sigma := \{1, \dots, K\}$. Let Σ^* be the set of all finite length sequences over the alphabet Σ. Specifically, the empty sequence ϵ is included in Σ^*. We focus on the online learning framework, where learning is performed in rounds. Let $x^{[t]} \in \Sigma$ be the t^{th} symbol in a sequence. Denote $\mathbf{x}^{[1:t-1]} \in \Sigma^*$ be the *context* of $x^{[t]}$, i.e. $\mathbf{x}^{[1:t-1]} := x^{[1]}, \dots, x^{[t-1]}$. For completeness, let $\mathbf{x}^{[t:t-1]} := \epsilon$. On round t, the algorithm first predicts $\hat{x}^{[t]} \in \Sigma$ according to its current prediction rule and the context $\mathbf{x}^{[t:t-1]}$. After that, the true symbol $x^{[t]}$ is revealed and the algorithm suffers a loss which reflects the degree to which its prediction was wrong. The algorithm then has the option to modify its prediction rule, with the explicit goal of improving the accuracy of its predictions for the rounds to come.

Assume that any symbol in the sequence is determined by its context, the problem of sequence prediction can be formulated as finding a function $f : \Sigma^* \to \Sigma$. To predict the t^{th} symbol one can simply set $\hat{x}^{[t]} := f(\mathbf{x}^{[1:t-1]})$. We generalize this definition and allow the algorithm to output predictions from a real-valued set Y. Specifically, let $Y := \mathbb{R}^K$ and $f : \Sigma^* \to Y$, where a prediction $\mathbf{y} \in Y$ is interpreted as a degree of confidence for each of the symbols in Σ. Consequently, the mapping from a score vector $\mathbf{y}$ to an actual symbol in Σ is via $\hat{x} := \arg\max_{k\in\Sigma} y_k$. On round t, the loss of f is measured by a zero-one loss function $\ell_{\mathbb{1}}\left(f; (\mathbf{x}^{[1:t-1]}, x^{[t]})\right)$. That is, $\ell_{\mathbb{1}}$ is zero if $\hat{x}^{[t]} = x^{[t]}$. Therefore, our ultimate goal is to incrementally learn a function f which minimizes

$$\frac{1}{T}\sum_{t=1}^{T} \ell_{\mathbb{1}}\left(f; (\mathbf{x}^{[1:t-1]}, x^{[t]})\right),$$

where T is the length of the sequence.

IV. Sequence Prediction as Linear Separation

Having described a general scheme for sequence prediction, we now focus on determining the form of f to obtain a concrete algorithm. In what follows we cast the sequence prediction problem as the problem of linear separation in a Hilbert space, which is a popular topic in machine learning. We shall see that by doing so one can harness powerful machine learning tools such as the Perceptron algorithm [26], [27] and online convex programming [43] to our purpose.

As it was suggested in the Section I, the number of previous symbols needed to make an accurate prediction is usually not constant, but rather depends on the identity of those symbols. With this consideration in mind, we define a *suffix-closed* set $V \subset \Sigma^*$ such that for every $\mathbf{s} \in V$, every suffix of $\mathbf{s}$ (including ϵ) is also contained in V. To allow the algorithm to look as far back as needed, we can set V to be large enough. Specifically, let $\mathcal{H}$ be the Hilbert space of square integrable functions $\psi : V \to \mathbb{R}$ endowed with the inner product

$$\langle \zeta, \psi \rangle = \sum_{\mathbf{s}\in V} \zeta(\mathbf{s})\psi(\mathbf{s}),$$

and the induced norm $\|\zeta\| = \sqrt{\langle \zeta, \zeta \rangle}$. Note that if we can bound $|V|$ by a constant, then the Hilbert space $\mathcal{H}$ is isomorphic to the $|V|$-dimensional vector space, i.e. $\mathbb{R}^{|V|}$.

On round t the context $\mathbf{x}^{[1:t-1]}$ is observed, we map this sequence to the function $\psi \in \mathcal{H}$ as follows

$$\psi(\mathbf{s}^{[1:i]}) := \begin{cases} 1 & \text{if } \mathbf{s}^{[1:i]} = \epsilon \\ e^{-\rho i} & \text{if } \mathbf{s}^{[1:i]} \in \operatorname{suf}(\mathbf{x}^{[1:t-1]}) \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

where $\operatorname{suf}(\mathbf{x}^{[1:t-1]})$ denotes the set of all suffixes of $\mathbf{x}^{[1:t-1]}$. The decay factor $\rho > 0$ is a predefined hyperparameter and mitigates the effect of long contexts on the function ψ. It is noticed from Eq. (1) that all suffixes of $\mathbf{x}^{[1:t-1]}$ are mapped to non-zero values; the value tends to decrease as the length of suffix increases. This idea expresses the assumption that symbols appearing earlier in a sequence have the least importance in modeling the current symbol. As we shall see in Section V-C, this assumption can be infringed to some extent by incorporating side information into our model.

We have mapped sequences to functions in $\mathcal{H}$. The next step is to create separating hyperplanes in $\mathcal{H}$ for prediction. We employ a *multi-class context tree*. A multi-class context tree is a K-ary tree, each node of which represents one of the sequences in V. Specifically, the root of the tree represents the empty sequence ϵ. The node that represents the sequence $\mathbf{x}^{[i:j]}$ is the child of the node representing the sequence $\mathbf{x}^{[i+1:j]}$. An observed sequence thus defines a path from the root of the tree to one of its nodes. Note that this path can either terminate at an inner node or at a leaf. We associate each node with a K-dimensional vector. In other words, a multiclass context tree can be represented as a function $\tau : V \to \mathbb{R}^K$. An illustrative example is given in Fig. 2. In particular, if we only look at the k^{th} element of the vector on every node and denote the corresponding context tree as $\tau_k : V \to \mathbb{R}$, then it is easy to verify that τ_k is embedded in $\mathcal{H}$.

To construct the context tree on rounds, we initialize $\tau^{[1]}$ to be a tree of a single (the root) node which assigns a weight of zero to the empty sequence, i.e. $V^{[1]} := \{\epsilon\}$. After receiving $\mathbf{x}^{[1:t-1]}$, a trivial solution is adding all sequences in the set $\operatorname{suf}(\mathbf{x}^{[1:t-1]})$ to $V^{[t]}$ and associate each of which with an undetermined vector in $\mathbb{R}^K$. The method for determining the value of these vectors will be presented in Section V-A. For a long sequence adding all suffixes to the tree can impose serious computational problems, as the required memory for storing the tree grows quadratically with t. We shall resolve this issue in Section V-B.

Returning to our sequence prediction problem, let $\mathbf{x}^{[1:t-1]}$ be the sequence of observed symbols on round t, and let $\psi^{[t]}$ be its corresponding function in $\mathcal{H}$ defined by Eq. (1). Denote $\tau_k^{[t]}$ as the current context tree subject to the class k. By following the description in Section III, the prediction problem can be formulated as

$$\hat{x}^{[t]} := \arg\max_{k\in\Sigma} \underbrace{\left\langle \psi^{[t]}, \tau_k^{[t]} \right\rangle}_{f(\mathbf{x}^{[1:t-1]})}. \quad (2)$$

Geometrically, we can consider $\mathcal{H}$ as a space partitioned by K hyperplanes, whose normal are given by $\tau_1, \dots, \tau_k$, respectively. The t^{th} symbol is then predicted by picking the hyperplane that gives the maximum (signed) distance to the vector $\psi^{[t]}$.

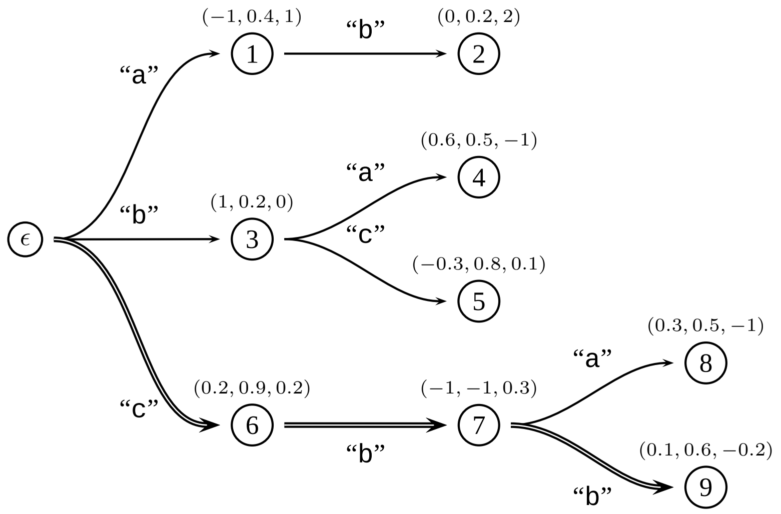


Fig. 2. An example of a multi-class context tree, where $K = 3$ and $V = \{\epsilon, \mathsf{a}, \mathsf{ba}, \mathsf{b}, \mathsf{ab}, \mathsf{cb}, \mathsf{c}, \mathsf{bc}, \mathsf{abc}, \mathsf{bbc}\}$. The label on each node represents the index. Notice how the index matches the element in V. The context associated with each node is indicated on the edges of the tree along the path from the root ϵ to that node. The vector associated with each node is provided above each node. This context tree can be parameterized as a 10×3 matrix, with the first column $(0, 0, 0)^\top$ corresponds to the empty sequence ϵ. Considering the context tree as a function, given an input sequence "aabbc", the output from this context tree is $(0.1, 0.6, -0.2)$, whose path is plotted with double lines.

V. Online Learning Algorithm

It can be seen in Section IV, our predictor is fully specified by a multi-class context tree τ, which can be represented by $\tau_1, \ldots, \tau_K$. Given a fixed V, we can parameterize τ_k by a vector $\mathbf{w}_k \in \mathbb{R}^{|V|}$. Denote $\mathbf{W} := [\mathbf{w}_1, \ldots, \mathbf{w}_K]$, in which rows correspond to vectors associated with each node as depicted in Fig. 2. The size of $\mathbf{W}$ is thus $|V| \times K$. Note that a context tree is now fully specified by its weight vector $\mathbf{W}$ and structure V. That is, every $\{\mathbf{W}, V\}$ represents a unique τ and vice versa. Therefore, the problem of learning an accurate predictor can be reduced to the problem of determining $\mathbf{W}$ and V. Denote $\boldsymbol{\psi}^{[t]} \in \mathbb{R}^{|V|}$ a vector corresponding to the sequence $\mathbf{x}^{[1:t-1]}$. To construct $\boldsymbol{\psi}^{[t]}$ we simply follow Eq. (1) and only assign values to the sequences in $\mathbf{x}^{[1:t-1]} \cap V$. Elements of $\boldsymbol{\psi}^{[t]}$ are indexed in the same order as $\mathbf{w}_k^{[t]}$. Thus, the score vector $\mathbf{y}$ described in Section III amounts to $(\mathbf{W}^{[t]})^\top \boldsymbol{\psi}^{[t]}$.

This section describes our online learning algorithm in four parts. We first describe the method to learn $\mathbf{W}$, subsequently, we present an approach for constructing V in a memory-efficient way. Extension for incorporating side information is described towards the end. Finally, several implementation issues are highlighted.

A. Learning Weight Vectors

We first show how the update of $\mathbf{W}$ can be performed in rounds. Our method is closely related to the family of confidence-weighted linear classifiers [32], [33], [34], [35]. Following the idea of previous work, we maintain a Gaussian distribution for every column of $\mathbf{W}$ with a mean vector $\boldsymbol{\mu}_k \in \mathbb{R}^{|V|}$ and a diagonal covariance matrix $\boldsymbol{\Lambda}_k \in \mathbb{R}^{|V|\times|V|}$, i.e. $\mathbf{w}_k \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)$. Notice that by restricting $\boldsymbol{\Lambda}_k$ to a diagonal matrix, the weights become independent [1]. This is not true in real-world, yet it is necessary due to the large value of $|V|$. For the sake of efficiency, the predicted symbol is simply approximated by $\arg\max_{k\in\Sigma} \boldsymbol{\mu}_k \cdot \boldsymbol{\psi}^{[t]}$ instead of using weight vectors sampled from $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)$. In other words, the information captured by $\boldsymbol{\Lambda}_k$ does not influence the decision. This is analogous to Bayes point machines [44].

On each round, we update the model by minimizing the Kullback-Leibler divergence between new distribution and the old one while ensuring that the probability of correct prediction on t^{th} symbol is not smaller than the confidence hyperparameter $\eta \in [0, 1]$. After revealing the true symbol $r := x^{[t]}$, we need to update $(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)$ to the solution of the following optimization problem

$$\left(\boldsymbol{\mu}_k^{[t+1]}, \boldsymbol{\Lambda}_k^{[t+1]}\right) = \arg\min_{\boldsymbol{\mu},\boldsymbol{\Lambda}} \quad \mathrm{D_{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Lambda}) \,\|\, \mathcal{N}\left(\boldsymbol{\mu}_k^{[t]}, \boldsymbol{\Lambda}_k^{[t]}\right)\right)$$
$$\text{s.t.} \qquad \Pr_{\mathbf{w}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Lambda})}\left[\mathbf{w}_r \cdot \boldsymbol{\psi}^{[t]} \geq \mathbf{w} \cdot \boldsymbol{\psi}^{[t]}\right] \geq \eta. \quad (3)$$

Notice that the optimization problem in Eq. (3) needs to be solved $K - 1$ times for every $k \in \Sigma \setminus r$ on each round, which can be computationally expensive. We therefore provide a simplified algorithm, where only two updates is required on each round. The intuition was to ensure that the true symbol is more likely to be predicted than the symbol that is its closest competitor. Specifically, let s be the highest ranked wrong symbol on round t. That is,

$$s := \arg\max_{k\in\Sigma\setminus r} \boldsymbol{\mu}_k^{[t]} \cdot \boldsymbol{\psi}^{[t]}. \quad (4)$$

In each round only $(\boldsymbol{\mu}_r, \boldsymbol{\Lambda}_r)$ and $(\boldsymbol{\mu}_s, \boldsymbol{\Lambda}_s)$ are updated as

[1]One may consider $\mathbf{W}$ as a random variable from a matrix normal distribution, i.e. $\mathbf{W} \sim \mathcal{MN}(\mathbf{M}, \mathbf{U}, \mathbf{V})$, where $\mathbf{U}$ and $\mathbf{V}$ represents the correlation among-row and among-column, respectively. However, under the assumption of independent weights and independent symbols, $\mathbf{U}$ and $\mathbf{V}$ are simply diagonal matrices. This results an equivalent formulation to ours.

follows

$$\left(\boldsymbol{\mu}_r^{[t+1]}, \boldsymbol{\Lambda}_r^{[t+1]}\right) = \arg\min_{\boldsymbol{\mu},\boldsymbol{\Lambda}} \quad \mathrm{D_{KL}}\left(\mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Lambda}\right) \,\|\, \mathcal{N}\left(\boldsymbol{\mu}_r^{[t]}, \boldsymbol{\Lambda}_r^{[t]}\right)\right)$$

$$\text{s.t.} \qquad \Pr_{\mathbf{w}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Lambda})}\left[\mathbf{w}\cdot\boldsymbol{\psi}^{[t]} \geq \mathbf{w}_s\cdot\boldsymbol{\psi}^{[t]}\right] \geq \eta. \tag{5}$$

$$\left(\boldsymbol{\mu}_s^{[t+1]}, \boldsymbol{\Lambda}_s^{[t+1]}\right) = \arg\min_{\boldsymbol{\mu},\boldsymbol{\Lambda}} \quad \mathrm{D_{KL}}\left(\mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Lambda}\right) \,\|\, \mathcal{N}\left(\boldsymbol{\mu}_s^{[t]}, \boldsymbol{\Lambda}_s^{[t]}\right)\right)$$

$$\text{s.t.} \qquad \Pr_{\mathbf{w}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Lambda})}\left[\mathbf{w}_r\cdot\boldsymbol{\psi}^{[t]} \geq \mathbf{w}\cdot\boldsymbol{\psi}^{[t]}\right] \geq \eta. \tag{6}$$

Notice how the constraint of Eq. (5) and Eq. (6) differs from each other. We follow the derivation in [33] and obtain the closed-form update as

$$\boldsymbol{\mu}_r^{[t+1]} = \boldsymbol{\mu}_r^{[t]} + \alpha\boldsymbol{\Lambda}_r^{[t]}\boldsymbol{\psi}^{[t]} \tag{7}$$

$$\boldsymbol{\mu}_s^{[t+1]} = \boldsymbol{\mu}_s^{[t]} - \alpha\boldsymbol{\Lambda}_s^{[t]}\boldsymbol{\psi}^{[t]} \tag{8}$$

$$\boldsymbol{\Lambda}_r^{[t+1]} = \left(\left(\boldsymbol{\Lambda}_r^{[t]}\right)^{-1} + 2\alpha\phi\,\mathrm{diag}^2\left(\boldsymbol{\psi}^{[t]}\right)\right)^{-1} \tag{9}$$

$$\boldsymbol{\Lambda}_s^{[t+1]} = \left(\left(\boldsymbol{\Lambda}_s^{[t]}\right)^{-1} + 2\alpha\phi\,\mathrm{diag}^2\left(\boldsymbol{\psi}^{[t]}\right)\right)^{-1}, \tag{10}$$

where $\mathrm{diag}^2\left(\boldsymbol{\psi}^{[t]}\right)$ is a diagonal matrix made from the squares of the elements of $\boldsymbol{\psi}^{[t]}$ on the diagonal. The inverse of diagonal matrix can be computed element-wise. The coefficient α is calculated as follows

$$\alpha = \frac{-(1+2\phi m) + \sqrt{(1+2\phi m)^2 - 8\phi(m-\phi v)}}{4\phi v},$$

where

$$m = \left(\boldsymbol{\mu}_r^{[t]} - \boldsymbol{\mu}_s^{[t]}\right)\cdot\boldsymbol{\psi}^{[t]} \tag{11}$$

$$v = \left(\boldsymbol{\mu}_r^{[t]}\right)^{\top}\boldsymbol{\Lambda}_r\boldsymbol{\mu}_r^{[t]} - \left(\boldsymbol{\mu}_s^{[t]}\right)^{\top}\boldsymbol{\Lambda}_s\boldsymbol{\mu}_s^{[t]} \tag{12}$$

$$\phi = \Phi^{-1}(\eta), \tag{13}$$

and $\Phi^{-1}(\cdot)$ is the inverse of the normal cumulative distribution function.

For initialization we set $\boldsymbol{\mu}_k^{[1]} := \mathbf{0}$ and $\boldsymbol{\Lambda}_k^{[1]} := \mathbf{I}$ for all k, where $\mathbf{I}$ is the identity matrix. It is noticed from Eq. (7) and Eq. (8) that during online learning the mean weight vector is updated in a similar fashion as in the Perceptron. The confidence of all observed suffixes is increased by shrinking the corresponding value on the diagonal of $\boldsymbol{\Lambda}_k$ (see Eq. (9) and Eq. (10)), which leads to the update of weight vector in the next round more focusing on low confidence features.

B. Memory-Efficient Update of Suffix Set

Having described the method for learning the weight vectors of the context tree, we now focus on determining its structure, i.e. V. Instead of adding all suffixes of the context to V on each round, we introduce three strategies for constructing V in a memory-efficient way.

First of all, we only update V if the probability constraint

$$\Pr_{\substack{\mathbf{w}_r\sim\mathcal{N}(\boldsymbol{\mu}_r,\boldsymbol{\Lambda}_r)\\ \mathbf{w}_s\sim\mathcal{N}(\boldsymbol{\mu}_s,\boldsymbol{\Lambda}_s)}}\left[\mathbf{w}_r\cdot\boldsymbol{\psi}^{[t]} \geq \mathbf{w}_s\cdot\boldsymbol{\psi}^{[t]}\right] \geq \eta \tag{14}$$

is violated. Note that Eq. (14) can be rewritten as

$$(\boldsymbol{\mu}_r - \boldsymbol{\mu}_s)\cdot\boldsymbol{\psi}^{[t]} \geq \phi\sqrt{\left(\boldsymbol{\psi}^{[t]}\right)^{\top}(\boldsymbol{\Lambda}_r + \boldsymbol{\Lambda}_s)\boldsymbol{\psi}^{[t]}},$$

where $\phi = \Phi^{-1}(\eta)$. Further, we introduce a loss function as

$$\ell_{\phi}\left(\{(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)\}_{k=1}^{K}; (\mathbf{x}^{[1:t-1]}, x^{[t]})\right) :=$$

$$\max\left(0, \phi\sqrt{\left(\boldsymbol{\psi}^{[t]}\right)^{\top}(\boldsymbol{\Lambda}_r + \boldsymbol{\Lambda}_s)\boldsymbol{\psi}^{[t]}} - (\boldsymbol{\mu}_r - \boldsymbol{\mu}_s)\cdot\boldsymbol{\psi}^{[t]}\right). \tag{15}$$

It is easy to verify that satisfying the probability constraint Eq. (14) is equivalent to satisfying $\ell_{\phi} = 0$. In this case, we simply set $V^{[t+1]}$ to be equal to $V^{[t]}$. Otherwise we add all sequences in $\mathrm{suf}(\mathbf{x}^{[1:t-1]})$ to $V^{[t]}$. Note that ρ and ϕ can be tuned as a trade-off between the passiveness and aggressiveness of the update.

Second, when a sequence is extremely long, adding all suffixes of a long context can impose serious memory growth problem. Hence, it is not a practical solution. To limit the maximum depth of the context tree, we prune the context $\mathbf{x}^{[1:t-1]}$ to a certain length $\kappa^{[t]}$ before adding its suffixes to V, where $\kappa^{[t]}$ is given by

$$\kappa^{[t]} = \min\left(\left\lfloor\frac{1}{\rho}\log\ell_{\mathbb{1}}(t)\right\rfloor, t-1\right),$$

with $\ell_{\mathbb{1}}(t)$ denoting the number of prediction mistakes made by the algorithm so far. The intuition behind is to limit the depth of the context tree by the number of prediction mistakes, which is inspired by [18]. As a consequence, one can straightforwardly translate the mistake bound of confidence weighted classifier (Theorem 4 in [33]) into a bound on the growth-rate of the resulting context tree [18], [42].

Finally, we limit the size of V by removing the elements giving smallest $\sum_k \mu_{k,i}^2$ when $|V|$ exceeds the maximum allowed size Q, where $\mu_{k,i}$ is the i^{th} element of $\boldsymbol{\mu}_k$ and $i \in [1, Q]$. This criterion has been shown effective in recursive feature elimination [45] and has a good theoretical support [46], [47]. Alternatively, one can also use the $\sum_k 1/\lambda_{k,i}$ or $\sum_k |\mu_{k,i}|/\lambda_{k,i}$ as the criterion, where $\lambda_{k,v}$ is the v^{th} element on the diagonal of $\boldsymbol{\Lambda}_k$. By employing the above three strategies the context tree grows at a much slower pace and the algorithm can utilize memory more conservatively. Finally, the pseudo-code of our algorithm is summarized in Algorithm 1, which is called EOSP in the sequel for short.

C. Incorporation of Side Information

Thus far we augment only context information from the sequence. As we highlighted in Section I side information of system calls can support the prediction of the next symbol. Apart from that, in language modeling grammars (e.g. part-of-speech tags), topics, styles are helpful to predict the next word [48], [49]. Comparing to the n-gram models and Bayesian nonparametrics models [24], [19], a key advantage of our approach is its simplicity of leveraging side information. Specifically, if side information on round t can be given in the form of a vector $\mathbf{b}^{[t]} \in \mathbb{R}^B$, we can directly incorporate it into the prediction via a linear combination as follows

$$\hat{x}^{[t]} := \arg\max_{k\in\Sigma}\boldsymbol{\mu}_k^{[t]}\cdot\boldsymbol{\psi}^{[t]} + \boldsymbol{\gamma}_k^{[t]}\cdot\mathbf{b}^{[t]}.$$

This corresponds to replacing $\boldsymbol{\psi}^{[t]}$ in Algorithm 1 as a $(Q+B)$-dimensional vector $[\boldsymbol{\psi}^{[t]}, \mathbf{b}^{[t]}]$. The dimension of the mean vector and confidence matrix associated with each symbol is

Algorithm 1: Efficient online sequence prediction (EOSP).

```
Input     : Damping factor: $\rho > 0$; confidence
            parameter: $\eta \in [0,1]$; maximum allowed size
            of $V$: $Q > 0$.
Output    : Mean vectors and confidences matrices:
            $\{(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)\}_{k=1}^K$; set: $V$.
Initialize: $\forall k \in \Sigma$, $(\boldsymbol{\mu}_k^{[1]}, \boldsymbol{\Lambda}_k^{[1]}) = (\mathbf{0}, \mathbf{I})$, $\phi = \Phi^{-1}(\eta)$,
            $V^{[1]} = \{\epsilon\}$;
1  for $t = 1, 2, \dots$ do
2      Construct $\boldsymbol{\psi}^{[t]}$ from $\mathbf{x}^{[1:t-1]}$;      /* Eq. (1) */
3      Rank all symbols by $\boldsymbol{\mu}_k^{[t]} \cdot \boldsymbol{\psi}^{[t]}$;
4      Receive the true symbol $r$;
5      Compute $s$;                                /* Eq. (4) */
6      Suffer loss $\ell_\phi$;                    /* Eq. (15) */
7      if $\ell_\phi > 0$ then
8          while $|V^{[t]}| > Q - \kappa^{[t]}$ do
9              $i = \arg\min_{j=1,\dots,Q} \sum_{k\in\Sigma} \mu_{k,j}^2$;
10             $\forall k \in \Sigma$, $\mu_{k,i} = 0$;
11             Remove the $i^{\text{th}}$ sequence from $V^{[t]}$;
12         $V^{[t+1]} = V^{[t]} \cup \{\mathbf{x}^{[t-i:t-1]} \mid 1 \le i \le \kappa^{[t]}\}$;
13         Set $(\boldsymbol{\mu}_r^{[t+1]}, \boldsymbol{\Lambda}_r^{[t+1]})$ and $(\boldsymbol{\mu}_s^{[t+1]}, \boldsymbol{\Lambda}_s^{[t+1]})$;
           /* Eqs. (7) to (10) */
```

extended accordingly. Note that an ineffective representation of side information can adversely affect the prediction performance as well, hence there has to be some mechanism for selecting features that really contribute to prediction. In our algorithm, this can be done by constantly setting $\mu_{k,i+b}$ to zero if $\sum_k |\mu_{k,i+b}|/\lambda_{k,i+b}$ is too small. In addition, one can also initialize $\boldsymbol{\Lambda}_k := \begin{pmatrix} \mathbf{I}_{|V|\times|V|} & \\ & \gamma\mathbf{I}_{B\times B} \end{pmatrix}$ with $0 < \gamma < 1$ to balance the learning rate of the weights on the context and side information. Specifically, when $\gamma = 1$ the side information shares the same learning rate with context information; when $\gamma = 0$ the side information does not contribute the learning and prediction at all.

The side information used in this work is summarized in Table I. The idea of using these attributes is mainly based on our experiences and observations. For instance, we observed that system calls with similar functionality tend to occur together, which could be due to some sub-task of the process. Thus, if a particular group of system calls is frequently observed in the recent context, then the next system call is very likely from the same group. In present work, system calls are grouped manually by their documentation, which is partially based on [50]. It is also possible to automatically group system calls by using topic models [51]. Another observation is that a block of system calls repeats themselves especially when some of them return an error. This was probably attributed to the exception handling (e.g. restart mechanism) of a process. Thus, a simple statistic of the error codes is maintained and considered as one of the evidences for predicting the next system call. In practice, the side information listed in Table I can be easily extracted from the context with negligible computational cost.

D. *Efficient Implementation*

It can be observed from Eqs. (7) to (10) that most of the entries of $\boldsymbol{\psi}$, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Lambda}_k$ are zero, which implies a possibility to improve the efficiency by storing them in a compact way. In the implementation, we store $\boldsymbol{\mu}_k$, $\boldsymbol{\Lambda}_k$ and $\boldsymbol{\psi}$ in sparse vectors. The algorithms of addition and dot product for sparse vectors can be found in [52]. Moreover, as the updates of $(\boldsymbol{\mu}_r, \boldsymbol{\Lambda}_r)$ and $(\boldsymbol{\mu}_s, \boldsymbol{\Lambda}_s)$ are independent to each other, line 13 Algorithm 1 can be implemented in a parallel manner. Furthermore, the operations on V can be implemented efficiently using a data structure called *suffix trie*. Finally, removing one element at a time (line 8 to 11 Algorithm 1) is time consuming and in practice we remove as much as half of Q when $|V^{[t]}| > Q - \kappa^{[t]}$.

VI. Experimental Results

Two sets of experiments were carried out to validate our algorithm. First, we compared the accuracy and efficiency of EOSP with state-of-the-art sequence prediction methods. Second, we investigated several factors that affect the performance of EOSP in order to gain more insights of it.

The experiments were conducted on three groups of data. The first set of data is from BSM (Basic Security Module) data portion of 1998 DARPA intrusion detection evaluation data set created by MIT Lincoln Labs[2]. We used a subset of training data, which contained four-hour BSM audit data of all processes running on a Solaris machine. Each system call was recorded with its corresponding arguments and return value. The second group of data was obtained from University of New Mexico [4], in which system call traces of several process were generated in either "synthetic" or "live" manner[3]. Our experiment was conducted on their "live" normal data, where traces of programs were collected during normal usage of real users. Unlike DARPA data set, a trace in UNM data set is just a list of system call names; no arguments and return values are available. Therefore, for UNM data set only the functional group in Table I was available as side information. The third data set was collected by ourselves[4]. By using `strace` and a prepared script, we collected system call sequences with their corresponding arguments and return values from all executable programs on an Ubuntu system. The program options were chosen solely for the purpose of exercising the program, and not to meet any real user's requests. From these three sources we selected a total of 8 data sets, and their characteristics are summarized in Table II.

TABLE II. Characteristics of data sets used in the experiment.

Data set	# calls	# seq.	Min. len.	Max. len.	Avg. len.
darpa	243	200	2	3,074	57
lpr1	182	2,766	82	59,565	1,080
lpr2	182	1,232	74	39,306	449
sendmail1	190	8,000	8	173,664	669
sendmail2	190	8,000	8	149,616	648
stide1	164	8,000	225	146,695	1,055
stide2	164	8,000	108	174,401	1,255
ubuntu	458	1,218	2	53,247	952

[2]http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/
[3]http://www.cs.unm.edu/~immsec/systemcalls
[4]URL is masked for blind review.

TABLE I. SIDE INFORMATION USED IN OUR ALGORITHM FOR SYSTEM CALL PREDICTION.

Feature set	Size	Description
File descriptor	2	The number of opened files and the number of closed files, respectively.
File type	9	Each element represents the number of opened files of a particular type, such as `RDONLY`, `WRONLY`, `APPEND`, etc.
Functional group	9	Each element represents the number of occurrences of system calls associated with a group given a context. The groups were built in advance by categorizing similar system calls together, resulting 9 groups in total. For instance, the "file" group includes `creat`, `open`, `close`, `read`, etc. The "process" group includes `fork`, `wait`, `exec`, etc. The "signal" group includes `signal`, `kill`, `alarm`, etc.
Access location	12	Each element represents the number of accesses to a particular directory, such as `/usr/bin`, `/usr/lib`, `/usr/tmp`, etc.
Error code	124	Each element represents the number of caught errors of each code, such as `ENOENT`, `EAGAIN`, `EBGDF`, etc.
POSIX signal	28	Each element represents the number of sent signals of each type, such as `SIGSEGV`, `SIGABRT`, `SIGBUS` etc.
String character	256	Each element represents the frequency value of a string character. A `char` is considered as an 8-bit value, resulting 256 possible characters. We only count characters in the string that is not file path.

Four sequence prediction methods were employed in the experiment. They were interpolated Kneser-Ney (IKN) [3], online prediction suffix tree (PST) [18], sequence memoizer (SM) [20], and learning experts (LEX) [21]. We restricted the maximum length of context to 50 for all algorithms except for SM, which was designed for modeling context with infinite length. Specifically, we used a 50-gram IKN in our experiment. For LEX the number of experts was set to one and $d \coloneqq 50$, resulting an individual sequence predictor trained with the log loss. The maximum depth of the context tree for PST was set to 50. These four methods were compared with the proposed EOSP, and the algorithm with side information denoting as EOSP$_\text{s}$. The confidence parameter η was 0.8; the damping factor ρ was 0.1; the maximum length of the context was 50 and the maximum size of V was $20,000$.

A. Comparison of Predictive Performance

The comparison of predictive performance between different methods is summarized in Table III, where the online error rate and perplexity were used as evaluation metrics. The online error rate of an algorithm on a given input sequence is defined to be the number of prediction mistakes the algorithm makes on that sequence normalized by the length of the sequence. The perplexity reflects an algorithm's performance when taking its probabilistic output into account. For EOSP we just normalized the score vector to obtain the prediction probability $\Pr[\hat{x}^{[t]} \,|\, \mathbf{x}^{1:t-1}]$. The reported results were averaged over all sequences in each data set respectively.

It is evident from the results that, EOSP and EOSP$_\text{s}$ gave a considerably better prediction than other baseline algorithms. In particular, EOSP$_\text{s}$ achieved the best performance on the majority data sets (seven out of eight in terms of perplexity), which indicates the effectiveness of incorporating side information into the model. On five out of six UNM data sets, we observed an improvement by just incorporating the functional group information. On `darpa` and `ubuntu` data sets where side information are fully available, a striking improvement of EOSP$_\text{s}$ over EOSP was observed. In general, we found SM is a strong competitor in terms of online error rate. However, EOSP and EOSP$_\text{s}$ still outperformed SM with appreciable lower perplexity on all data sets. This suggests a potentially valuable property for our method, e.g. for combining it with other probabilistic model in a big system. Moreover, SM is much slower than EOSP on long sequence, as we shall see in the next experiment.

TABLE III. EXPERIMENTAL RESULTS ON DIFFERENT DATA SETS. SMALLER VALUE INDICATES BETTER PERFORMANCE.

(a) Online error rate (%) of different algorithms.

Data set	EOSP	EOSP$_\text{s}$	IKN	PST	SM	LEX
`darpa`	50.11	**48.17**	52.14	49.25	49.75	51.11
`lpr1`	41.63	41.53	41.09	46.24	**40.88**	42.27
`lpr2`	47.44	**47.03**	47.61	48.52	47.24	51.15
`sendmail1`	33.47	34.26	35.62	33.65	**33.06**	36.81
`sendmail2`	33.11	33.91	33.52	34.17	**32.19**	38.96
`stide1`	8.34	**8.29**	8.54	8.59	8.41	9.06
`stide2`	**7.75**	7.75	8.09	7.95	7.78	8.51
`ubuntu`	40.90	**36.13**	38.90	39.23	75.26	52.72

(b) Online perplexity of different algorithms.

Data set	EOSP	EOSP$_\text{s}$	IKN	PST	SM	LEX
`darpa`	48.98	**40.23**	78.34	98.97	63.07	82.36
`lpr1`	9.82	**9.23**	16.14	17.05	14.75	11.71
`lpr2`	12.94	**11.08**	21.43	16.34	19.94	22.19
`sendmail1`	8.31	**8.17**	11.48	30.34	9.23	11.90
`sendmail2`	8.33	**7.96**	11.50	30.38	9.17	12.46
`stide1`	1.42	**1.41**	2.08	3.42	1.92	4.06
`stide2`	**1.39**	1.41	1.98	3.23	1.67	4.51
`ubuntu`	33.13	**31.65**	42.81	35.35	68.25	35.62

B. Comparison of Efficiency

The comparison of computation speed and memory consumption for all algorithms is shown in Fig. 3 and Fig. 4, respectively. We concatenated all traces in `sendmail1` to obtain a long sequence, and tested different methods on this sequence with increasing length. The setup of each method was same as in the last experiment. For the sake of fair comparison, all algorithms were implemented in C/C++. We only plotted the curve for EOSP as EOSP$_\text{s}$ took almost same amount of time and memory in our experiment. As can be seen in Fig. 3, EOSP showed a substantial reduction of time comparing to other baseline algorithms. Moreover, the time cost of EOSP only increased at a very low pace with respect to the length of the sequence. As we expected, LEX and SM were extremely slow especially on long sequences, since on each round they require gradient descent and Gibbs sampling, respectively. On contrary, in EOSP one only need to compute dot products of sparse vectors on each round, which can be done efficiently. On the other hand, though the memory consumption of EOSP is higher than other baselines at the beginning, it remained almost constant with increasing length of the sequence. Methods such as IKN and LEX, however, consume more and more memory as the sequence becomes longer. This demonstrates the effectiveness of our update strategies described in Section V-B.

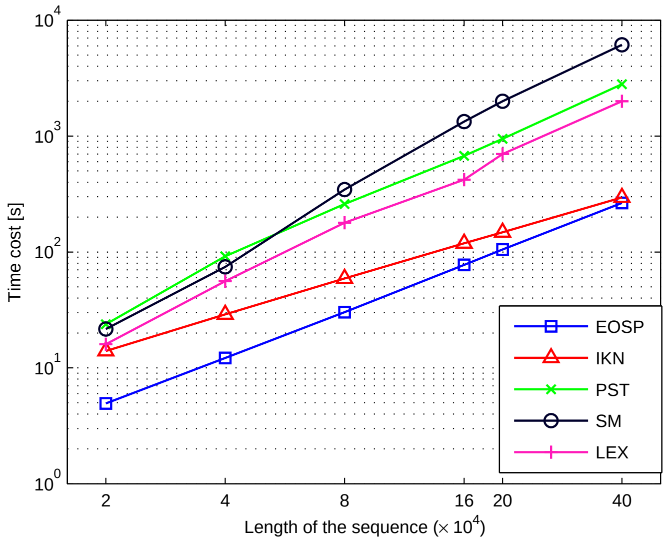


Fig. 3. Time cost in second (averaged over 10 runs) of different algorithms. Both axes are in logarithmic scale.

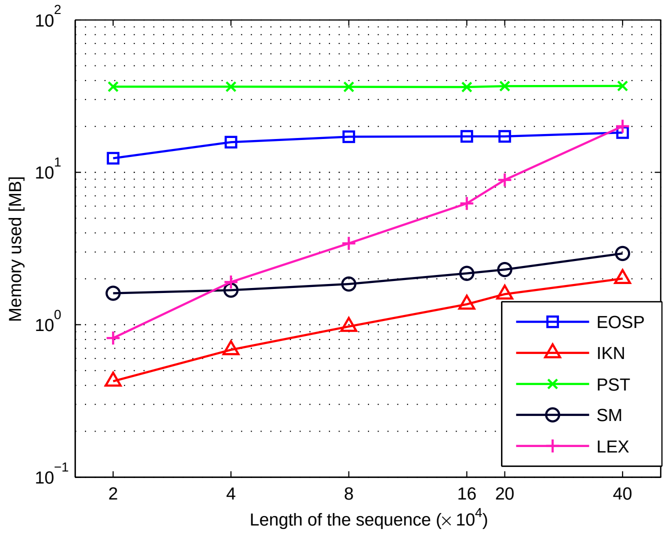


Fig. 4. Memory consumption (averaged over 10 runs) of different algorithms. Both axes are in logarithmic scale.

C. Exploration of Model Parameters

In order for EOSP to be a practical tool in real-world applications, it is necessary to make decisions about the details of its specification. Our exploration focused on three parameters that mainly govern the performance of EOSP. Namely, the confidence parameter η, the maximum length of the context, and the maximum size of V. We focused only on EOSP and ignored all side information in this set of experiments.

The performance of EOSP with respect to different settings of confidence parameter η is summarized in Table IV. We fixed the maximum length of the context to 50 and maximum size of V to $20,000$. On the majority of data sets, the online error rate hit the bottom when η is around 0.9. However, the lowest perplexity was often observed when $\eta := 0.8$; the perplexity slightly increased after that. In general, bigger value of η allows the algorithm to perform a more confident update on each round, which generally leads to higher predictive accuracy when the data is noise-less.

TABLE IV. PERFORMANCE OF EOSP W.R.T. DIFFERENT SETTINGS OF CONFIDENCE PARAMETER η. SMALLER VALUE INDICATES BETTER PERFORMANCE.

(a) Online error rate (%) of EOSP

Data set	0.6	0.7	0.8	0.9
`darpa`	50.41	50.52	**50.11**	50.39
`lpr1`	41.61	41.55	41.63	**41.43**
`lpr2`	47.21	47.02	47.44	**46.56**
`sendmail1`	34.25	34.12	33.47	**33.12**
`sendmail2`	33.32	**33.10**	33.11	33.74
`stide1`	8.21	8.25	8.34	**8.13**
`stide2`	7.79	7.79	7.75	**7.64**
`ubuntu`	44.77	44.76	**40.90**	44.60

(b) Perplexity of EOSP

Data set	0.6	0.7	0.8	0.9
`darpa`	**48.82**	49.05	48.98	49.03
`lpr1`	9.74	9.73	9.82	**9.72**
`lpr2`	12.62	12.57	12.94	**12.33**
`sendmail1`	8.67	8.64	**8.31**	8.44
`sendmail2`	8.69	8.67	**8.33**	8.47
`stide1`	1.43	1.44	**1.42**	1.42
`stide2`	1.40	1.41	**1.39**	1.39
`ubuntu`	32.93	33.02	32.93	**32.29**

Table V summarizes the results of EOSP with respect to different maximum length of the context, where $\eta := 0.8$ and $Q := 20,000$. Although one may expect an improvement of the predictive accuracy by allowing the algorithm to look back long distant context, we found that the optimal length of the context varies with data sets. On `darpa`, `lpr1` and `stide1`, for example, the context length of 40 was sufficient for a good prediction; increasing this length did not improve the prediction. On `ubuntu`, the online error rate decreased with increasing context length up to 100. In general, we found that the prediction of EOSP is not adversely affected by the overlength context, though its efficiency can be degraded due to more memory consumption.

TABLE V. PERFORMANCE OF EOSP W.R.T. DIFFERENT MAXIMUM LENGTH OF CONTEXT.

(a) Online error rate (%) of EOSP

Data set	20	40	60	80	100
`darpa`	50.21	**50.10**	50.10	50.10	50.10
`lpr1`	41.66	**41.63**	41.63	41.65	41.65
`lpr2`	**47.45**	47.45	47.45	47.45	47.45
`sendmail1`	35.70	33.48	**33.47**	33.47	33.47
`sendmail2`	33.67	33.60	**33.11**	33.11	33.11
`stide1`	8.45	**8.27**	8.27	8.27	8.27
`stide2`	8.02	**7.75**	7.75	7.75	7.75
`ubuntu`	41.84	41.23	40.90	40.75	**40.69**

(b) Perplexity of EOSP

Data set	20	40	60	80	100
`darpa`	**48.97**	48.97	48.98	48.98	48.98
`lpr1`	**9.78**	9.82	9.82	9.82	9.82
`lpr2`	12.94	**12.93**	12.94	12.94	12.94
`sendmail1`	8.36	**8.31**	8.31	8.31	8.31
`sendmail2`	8.38	**8.32**	8.32	8.33	8.33
`stide1`	1.43	**1.42**	1.42	1.42	1.42
`stide2`	1.40	**1.39**	1.40	1.40	1.40
`ubuntu`	33.15	33.13	33.12	33.11	**33.10**

Finally, to study the performance with respect to different sizes of V, we fixed $\eta := 0.8$ and the maximum length of context to 50. Results are summarized in Table VI. It was found that on the majority of data sets predictive performance can be improved by allowing V to contain more suffixes, which can be expected. However, on darpa data set having at most $4,000$ suffixes in V was sufficient for obtaining a good result; increasing the upper bound of $|V|$ did not improve the performance but raised the memory consumption. This is probably due to that most sequences in darpa are short (with average length of 57) and hence there are not many combinations for frequently occurred subsequences. In general, if the patterns in a sequence are simple (especially with some periodicity), then one can set a small size for V.

TABLE VI. PERFORMANCE OF EOSP W.R.T. DIFFERENT MAXIMUM SIZE ($\times 10^3$) OF V.

(a) Online error rate (%) of EOSP.

Data set	1	2	4	8	16
darpa	50.39	50.21	**50.13**	50.13	50.13
lpr1	42.45	42.18	41.83	41.68	**41.65**
lpr2	48.21	48.00	47.56	**47.45**	47.45
sendmail1	38.01	36.57	35.92	35.24	**34.63**
sendmail2	34.98	33.55	32.88	32.60	**32.43**
stide1	9.23	8.99	8.62	8.32	**8.27**
stide2	8.74	8.57	8.28	7.90	**7.85**
ubuntu	42.73	42.20	41.97	41.51	**41.21**

(b) Perplexity of EOSP.

Data set	1	2	4	8	16
darpa	**48.97**	48.98	48.98	48.98	48.98
lpr1	10.37	10.15	9.93	**9.82**	9.82
lpr2	13.54	13.44	13.01	**12.94**	12.94
sendmail1	9.46	8.84	8.47	**8.31**	8.31
sendmail2	9.49	8.88	8.49	**8.33**	8.33
stide1	1.50	1.47	1.45	**1.43**	1.43
stide2	1.46	1.45	1.42	**1.40**	1.40
ubuntu	33.53	33.30	33.19	33.15	**33.11**

VII. CONCLUSIONS

Motivated by the problem of system call prediction, this work has proposed a novel method for predicting the next symbol in a sequence. Unlike previous methods our algorithm does not rely on a fixed length context during learning and can be easily incorporated with side information. The algorithm maintains a set of distributions over parameters. On each round, the distributions are updated to satisfy a probabilistic constraint. The update can be computed in closed-form and implemented using sparse vectors. Moreover, we proposed several strategies to reduce the memory consumption, allowing a good scalability on long sequences. An improvement of accuracy and efficiency over existing methods has been demonstrated in the experiments.

Our method can serve as a backbone in a wide range of real-time applications, such as intrusion detection and power consumption modeling on mobile devices. Comparing to previous methods in this area, our algorithm allows one to incorporate the domain knowledge as side information to improve prediction. Besides, our framework can be also adopted to perform other tasks, such as language modeling and structure prediction. An important question for future studies is to explore theoretical properties of our algorithm, such as the convergence rate under different noise settings. In particular, it would be interesting to develop a robust algorithm for predicting sequence with adversarial noise.

ACKNOWLEDGMENT

This work is funded by the HIVE project (FKZ: 16BY1200D) of the German Federal Ministry of Education and Research. We would like to thank Yurong Tao for help with experiments, and anonymous reviewers for helpful suggestions.

REFERENCES

[1] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *Information Theory, IEEE Transactions on*, vol. 24, no. 5, pp. 530–536, 1978.

[2] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[3] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1996, pp. 310–318.

[4] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 1999, pp. 133–145.

[5] E. Eskin, W. Lee, and S. J. Stolfo, "Modeling system calls for intrusion detection with dynamic window sizes," in *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, vol. 1. IEEE, 2001, pp. 165–175.

[6] P. Fricke, F. Jungermann, K. Morik, N. Piatkowski, O. Spinczyk, M. Stolpe, and J. Streicher, "Towards adjusting mobile devices to users behaviour," in *Analysis of Social Media and Ubiquitous Data*. Springer, 2011, pp. 99–118.

[7] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 153–168.

[8] Y. Oyama, K. Onoue, and A. Yonezawa, "Speculative security checks in sandboxing systems," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 8–pp.

[9] H. Robbins, "Asymptotically subminimax solutions of compound statistical decision problems," in *Herbert Robbins Selected Papers*. Springer, 1985, pp. 7–24.

[10] D. Blackwell, "An analog of the minimax theorem for vector payoffs," *Pacific Journal of Mathematics*, vol. 6, no. 1, pp. 1–8, 1956.

[11] J. Hannan, "Approximation to bayes risk in repeated play," *Contributions to the Theory of Games*, vol. 3, pp. 97–139, 1957.

[12] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[13] T. M. Cover and A. Shenhar, "Compound bayes predictors for sequences with apparent markov structure," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 7, no. 6, pp. 421–424, 1977.

[14] M. Feder, N. Merhav, and M. Gutman, "Universal prediction of individual sequences," *Information Theory, IEEE Transactions on*, vol. 38, no. 4, pp. 1258–1270, 1992.

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

[16] 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.

[17] F. C. Pereira and Y. Singer, "An efficient extension to mixture techniques for prediction and decision trees," *Machine Learning*, vol. 36, no. 3, pp. 183–199, 1999.

[18] O. Dekel, S. Shalev-Shwartz, and Y. Singer, "Individual sequence prediction using memory-efficient context trees," *Information Theory, IEEE Transactions on*, vol. 55, no. 11, pp. 5251–5262, 2009.

[19] F. Wood, C. Archambeau, J. Gasthaus, L. James, and Y. W. Teh, “A stochastic memoizer for sequence data,” in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1129–1136.

[20] F. Wood, J. Gasthaus, C. Archambeau, L. James, and Y. W. Teh, “The sequence memoizer,” *Communications of the ACM*, vol. 54, no. 2, pp. 91–98, 2011.

[21] E. Eban, A. Birnbaum, S. Shalev-Shwartz, and A. Globerson, “Learning the experts for online sequence prediction,” in *ICML*, 2012.

[22] R. Begleiter, R. El-Yaniv, and G. Yona, “On prediction using variable order markov models,” *J. Artif. Intell. Res. (JAIR)*, vol. 22, pp. 385–421, 2004.

[23] A. Martin, G. Seroussi, and M. J. Weinberger, “Linear time universal coding and time reversal of tree sources via fsm closure,” *Information Theory, IEEE Transactions on*, vol. 50, no. 7, pp. 1442–1468, 2004.

[24] Y. W. Teh, “A hierarchical bayesian language model based on pitman-yor processes,” in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2006, pp. 985–992.

[25] T. Motzkin and I. Schoenberg, “The relaxation method for linear inequalities,” *Canadian Journal of Mathematics*, vol. 6, no. 3, pp. 393–404, 1954.

[26] F. Ronsenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychological review*, vol. 65, pp. 386–408, 1958.

[27] A. B. Novikoff, “On convergence proofs for perceptrons,” DTIC Document, Tech. Rep., 1963.

[28] R. O. Duda, P. E. Hart *et al.*, *Pattern classification and scene analysis*. Wiley New York, 1973, vol. 3.

[29] K. Crammer and Y. Singer, “Ultraconservative online algorithms for multiclass problems,” *The Journal of Machine Learning Research*, vol. 3, pp. 951–991, 2003.

[30] N. Cesa-Bianchi, A. Conconi, and C. Gentile, “A second-order perceptron algorithm,” *SIAM Journal on Computing*, vol. 34, no. 3, pp. 640–668, 2005.

[31] C. Brotto, C. Gentile, and F. Vitale, “On higher-order perceptron algorithms,” *Advances in Neural Information Processing Systems*, vol. 19, 2007.

[32] M. Dredze, K. Crammer, and F. Pereira, “Confidence-weighted linear classification,” in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 264–271.

[33] K. Crammer, M. D. Fern, and O. Pereira, “Exact convex confidence-weighted learning,” in *In Advances in Neural Information Processing Systems 22*. Citeseer, 2008.

[34] K. Crammer, A. Kulesza, M. Dredze *et al.*, “Adaptive regularization of weight vectors,” *Advances in Neural Information Processing Systems*, vol. 22, pp. 414–422, 2009.

[35] J. Wang, P. Zhao, and S. C. Hoi, “Exact soft confidence-weighted learning,” *arXiv preprint arXiv:1206.4612*, 2012.

[36] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, “A sense of self for unix processes,” in *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1996, pp. 120–128.

[37] W. Lee, S. J. Stolfo, and P. K. Chan, “Learning patterns from unix process execution traces for intrusion detection,” in *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, 1997, pp. 50–56.

[38] C. W. Geib and R. P. Goldman, “Plan recognition in intrusion detection systems,” in *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX’01. Proceedings*, vol. 1. IEEE, 2001, pp. 46–55.

[39] L. Feng, X. Guan, S. Guo, Y. Gao, and P. Liu, “Predicting the intrusion intentions by observing system call sequences,” *Computers & Security*, vol. 23, no. 3, pp. 241–252, 2004.

[40] A. Chaturvedi, S. Bhatkar, and R. Sekar, “Improving attack detection in host-based ids by learning properties of system call arguments,” in *In Proceedings of the IEEE Symposium on Security and Privacy*. Citeseer, 2005.

[41] G. Tandon and P. Chan, “Learning rules from system call arguments and sequences for anomaly detection,” in *ICDM Workshop on Data Mining for Computer Security (DMSEC)*, 2003, pp. 20–29.

[42] N. Karampatziakis and D. Kozen, “Learning prediction suffix trees with winnow,” in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 489–496.

[43] S. Shalev-shwartz and Y. Singer, “Convex repeated games and fenchel duality,” in *Advances in Neural Information Processing Systems 19*. MIT Press, 2006.

[44] R. Herbrich, T. Graepel, and C. Campbell, “Bayes point machines,” *The Journal of Machine Learning Research*, vol. 1, pp. 245–279, 2001.

[45] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene selection for cancer classification using support vector machines,” *Machine learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[46] X.-w. Chen, X. Zeng, and D. van Alphen, “Multi-class feature selection for texture classification,” *Pattern Recognition Letters*, vol. 27, no. 14, pp. 1685–1691, 2006.

[47] O. Chapelle and S. S. Keerthi, “Multi-class feature selection with support vector machines,” in *Proceedings of the American statistical association*, 2008.

[48] T. L. Griffiths, M. Steyvers, D. M. Blei, and J. B. Tenenbaum, “Integrating topics and syntax,” *Advances in neural information processing systems*, vol. 17, pp. 537–544, 2005.

[49] X. Wang, A. McCallum, and X. Wei, “Topical n-grams: Phrase and topic discovery, with an application to information retrieval,” in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*. IEEE, 2007, pp. 697–702.

[50] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system concepts*. J. Wiley & Sons, 2009.

[51] H. Xiao and T. Stibor, “A supervised topic transition model for detecting malicious system call sequences,” in *Proceedings of the 2011 workshop on Knowledge discovery, modeling and simulation*. ACM, 2011, pp. 23–30.

[52] T. A. Davis, *Direct methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2006, vol. 2.