Fast Second-Order Online Kernel Learning Through Incremental Matrix Sketching and Decomposition

Dongxie Wen[1] , **Xiao Zhang**[1,*] , **Zhewei Wei**[1,*] , **Chenping Hou**[2] , **Shuai Li**[3] and **Weinan Zhang**[3]
[1]Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
[2]National University of Defense Technology
[3]Shanghai Jiao Tong University
{2019202221, zhangx89, zhewei}@ruc.edu.cn, hcpnudt@hotmail.com, {shuaili8, wnzhang}@sjtu.edu.cn

Abstract

Second-order Online Kernel Learning (OKL) has attracted considerable research interest due to its promising predictive performance in streaming environments. However, existing second-order OKL approaches suffer from at least quadratic time complexity with respect to the pre-set budget, rendering them unsuitable for large-scale datasets. Moreover, the singular value decomposition required to obtain explicit feature mapping is computationally expensive due to the complete decomposition process. To address these issues, we propose FORKS, a fast incremental matrix sketching and decomposition approach tailored for second-order OKL. FORKS constructs an incremental maintenance paradigm for second-order kernelized gradient descent, which includes incremental matrix sketching for kernel approximation and incremental matrix decomposition for explicit feature mapping construction. Theoretical analysis demonstrates that FORKS achieves a logarithmic regret guarantee on par with other second-order approaches while maintaining a linear time complexity w.r.t. the budget, significantly enhancing efficiency over existing methods. We validate the performance of our method through extensive experiments conducted on real-world datasets, demonstrating its superior scalability and robustness against adversarial attacks.

1 Introduction

The objective of online learning is to efficiently and effectively update hypotheses in a data stream environment, where the processes of training and testing are intermixed [Shalev-Shwartz, 2012]. A popular online learning algorithm is Online Gradient Descent (OGD) [Zinkevich, 2003], which aims to minimize the loss function by iteratively adjusting the parameters in the direction of the negative gradient of the function. To address this limitation, Online Kernel Learning (OKL) maps the input space to a high-dimensional reproducing kernel Hilbert space (RKHS), effectively handling nonlinear learning tasks [Kivinen *et al.*, 2001; Singh *et al.*, 2012; Hu *et al.*, 2015; Lu *et al.*, 2016b; CAO *et al.*, 2017; Sahoo *et al.*, 2019].

From an optimization perspective, OKL can be classified into first-order and second-order methods. First-order methods use a fixed learning rate for gradient descent, resulting in $O(\sqrt{T})$ regret for any arbitrary sequence of convex losses [Cavallanti *et al.*, 2007; Orabona *et al.*, 2008; Hoi *et al.*, 2012; Lu *et al.*, 2016b; Zhang and Liao, 2019], where T denotes the number of rounds. Although some works set a more aggressive learning rate to improve the regret bound to $O(\log T)$, it requires the assumption that the loss function exhibits strong convexity, which is unrealistic for most loss functions [Zhu and Xu, 2015]. In contrast, second-order algorithms utilize the Hessian matrix to adjust the learning rate dynamically, achieving logarithmic regret without requiring strong convexity in all directions [Hazan *et al.*, 2007; Zhdanov and Kalnishkan, 2013; Calandriello *et al.*, 2017a; Calandriello *et al.*, 2017b; Zhang *et al.*, 2023].

Despite the improved regret, exact second-order OKL requires $O(t^2)$ space and time complexity at round t due to the need to store the entire kernel matrix. Previous studies have primarily focused on approximating the kernel matrix using online sampling techniques [Calandriello *et al.*, 2017a; Calandriello *et al.*, 2017b; CHEN *et al.*, 2022]. However, sampling-based algorithms face two significant issues. First, without prior knowledge of the effective dimension of streaming data, sampling-based methods exhibit at least quadratic time complexity with respect to the budget, which is the maximum size of the static subspace. Second, these methods only address the problem of kernel matrix approximation in online kernel learning and cannot efficiently obtain explicit feature mapping. More precisely, due to the varying size of the subspace, complete singular value decomposition is required to calculate the feature mapping, which is computationally expensive.

In this paper, we propose a fast incremental matrix sketching approach for second-order online kernel learning, along with an efficient decomposition method designed for incremental updates, effectively addressing the two aforementioned challenges both theoretically and experimentally. Our contributions can be summarized as follows:

- We propose FORKS, a fast and effective second-order online kernel learning method that can be generalized to both regression and classification tasks. FORKS

*Xiao Zhang and Zhewei Wei are the corresponding authors.

maintains incremental matrix sketching using efficient low-rank modifications and constructs an effective time-varying explicit feature mapping. We provide a detailed theoretical analysis to illustrate the advantages of FORKS, including having linear time complexity w.r.t. the budget, and enjoying a logarithmic regret bound.

- We propose TISVD, a novel incremental singular value decomposition adapting to matrix decomposition problems in online learning environments. We theoretically compare the time complexity between TISVD and the original truncated low-rank SVD, confirming that FORKS with TISVD is computationally more efficient without compromising prediction performance.
- We conduct extensive experiments to demonstrate the superior performance of FORKS on both adversarial and real-world datasets, while also enhancing efficiency. Furthermore, we validate FORKS's robustness and scalability using a large-scale streaming recommendation dataset.

2 Notations and Preliminaries

Let $[n] = \{1, 2, \ldots, n\}$, upper-case bold letters (e.g., $\boldsymbol{A}$) represent matrix and lower-case bold letters (e.g., $\boldsymbol{a}$) represent vectors. We denote by $\boldsymbol{A}_{i*}$ and $\boldsymbol{A}_{*j}$ the i-th row and j-th column of matrix $\boldsymbol{A}$, $\boldsymbol{A}^\dagger$ the Moore-Penrose pseudoinverse of $\boldsymbol{A}$, $\|\boldsymbol{A}\|_2$ and $\|\boldsymbol{A}\|_F$ the spectral and Frobenius norms of $\boldsymbol{A}$. Let $\mathcal{S} = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^T \subseteq (\mathcal{X} \times \mathcal{Y})^T$ be the data stream of T instances, where $\boldsymbol{x}_t \in \mathbb{R}^d$. We use $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ to represent the SVD of $\boldsymbol{A}$, where $\boldsymbol{U}, \boldsymbol{V}$ denote the left and right matrices of singular vectors and $\boldsymbol{\Sigma} = \text{diag}[\lambda_1, ..., \lambda_n]$ is the diagonal matrix of singular values.

2.1 Online Kernel Learning

In this section, we introduce the problem of online kernel learning. Let the kernel function be denoted by $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, with the corresponding kernel matrix $\boldsymbol{K} = (\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j))$, where $\mathcal{X}$ represents the input space. Let $\mathcal{H}_\kappa$ denote the reproducing kernel Hilbert space (RKHS) induced by κ, and let the feature mapping $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{H}_\kappa$ correspond to this RKHS. In this context, the kernel function can be expressed as the inner product $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\phi}(\boldsymbol{x}_j)$.

Consider a data stream $\mathcal{S}$ and a convex loss function ℓ. At round t, we define the hypothesis as $f_t = \boldsymbol{w}_t^\top \boldsymbol{\phi}(\boldsymbol{x}_t)$, where $\boldsymbol{x}_t$ is the incoming sample. Upon receiving a new example $\boldsymbol{x}_t$, the hypothesis predicts the label $\hat{y}_t$ using f_t. The hypothesis incurs a loss $\ell_t(f_t(\boldsymbol{x}_t)) := \ell(f_t(\boldsymbol{x}_t), y_t)$, where y_t is the true label, and subsequently updates its model parameters. The objective of an online learning algorithm is to bound the cumulative regret, which is defined as:

$$\text{Reg}_T(f^*) = \sum_{t=1}^{T} \left[\ell_t(f_t) - \ell_t(f^*)\right],$$

where f^* is the optimal hypothesis, determined in hindsight.

2.2 Matrix Sketching

Given a matrix $\boldsymbol{M} \in \mathbb{R}^{a \times b}$, the sketch of $\boldsymbol{M}$ is defined as $\boldsymbol{M}\boldsymbol{S} \in \mathbb{R}^{a \times s}$, where $\boldsymbol{S} \in \mathbb{R}^{b \times s}$ is a sketch matrix. In this paper, we introduce the Sparse Johnson-Lindenstrauss Transform and Column-sampling matrix as the sketch matrix [Charikar *et al.*, 2002; Kane and Nelson, 2014].

Sparse Johnson-Lindenstrauss Transform (SJLT). SJLT is a randomized sketching technique based on hash functions. SJLT consists of D submatrices, written as $\boldsymbol{S} = [\boldsymbol{S}_1, \ldots, \boldsymbol{S}_D] \in \mathbb{R}^{b \times s_p}$. Each submatrix $\boldsymbol{S}_k \in \mathbb{R}^{b \times (s_p/D)}$ is defined by two sets of hash functions:

$$h_k : \{1, \ldots, b\} \to \left\{1, \ldots, \frac{s_p}{D}\right\},$$
$$g_k : \{1, \ldots, b\} \to \left\{\frac{-1}{\sqrt{D}}, \frac{1}{\sqrt{D}}\right\},$$

where $k \in \{1, \ldots, D\}$. For each submatrix, the element $[\boldsymbol{S}_k]_{i,j}$ equals $g_k(i)$ if $j = h_k(i)$, and equals 0 otherwise.

Column-sampling matrix. We denote the column-sampling matrix by $\boldsymbol{S}_m \in \mathbb{R}^{b \times s_m}$, the columns of $\boldsymbol{S}_m$ is obtained by uniformly sampling column vectors of $\boldsymbol{I}_{b \times b}$.

3 FORKS: The Proposed Algorithm

In this section, we propose a novel and efficient second-order OKL method that incorporates incremental sketching and decomposition. While previous work has primarily focused on the incremental maintenance of the approximated kernel matrix, our approach is the first to address the incremental construction and maintenance of the feature mapping.

3.1 Algorithm Overview

The second-order OKL process can be described in three key stages: kernel matrix approximation, feature mapping, and second-order online learning. Figure 1 illustrates the overall architecture of our method. Rather than storing the entire kernel matrix, we employ matrix sketching techniques to approximate it with a fixed, constant size. Furthermore, the novel truncated incremental singular value decomposition method is utilized to generate the time-varying feature mapping. Notably, both the kernel matrix and feature mapping can be updated incrementally, ensuring efficient maintenance and adaptability to changes in user preferences. Additionally, we use second-order updates to predict user behavior effectively. A non-trivial proof establishes that our method guarantees logarithmic regret while preserving computational efficiency.

3.2 Matrix Sketching for Kernel Matrix Approximation

In this section, we apply the matrix sketching technique to approximate the kernel matrix. Although this reduction has been employed in first-order methods [Zhang and Liao, 2018; Zhang and Liao, 2019], the regret bound for second-order online kernel learning, which operates under the directional curvature condition, remains unknown in the literature.

In the offline setting, given the kernel matrix $\boldsymbol{K}^{(t)} \in \mathbb{R}^{t \times t}$, the prototype model [Williams and Seeger, 2000] computes the approximate kernel matrix $\hat{\boldsymbol{K}}^{(t)} = \boldsymbol{C}_m^{(t)} \boldsymbol{U}_{\text{fast}} (\boldsymbol{C}_m^{(t)})^\top$ by solving the following optimization problem at round t:

$$\boldsymbol{U}_{\text{fast}} = \arg\min_{\boldsymbol{U}} \left\| \boldsymbol{C}_m^{(t)} \boldsymbol{U} \left(\boldsymbol{C}_m^{(t)}\right)^\top - \boldsymbol{K}^{(t)} \right\|_F^2, \qquad (1)$$

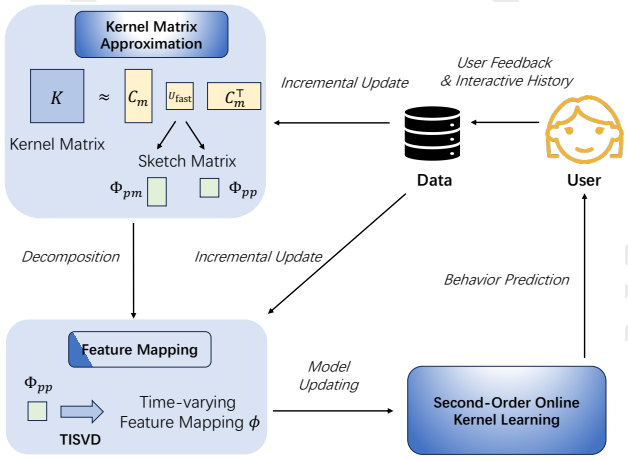


Figure 1: The illustration of the proposed method FORKS.

where $\boldsymbol{C}_m^{(t)}$ represents the sketch matrix at round t, defined as $\boldsymbol{C}_m^{(t)} = \boldsymbol{K}^{(t)}\boldsymbol{S}_m^{(t)} \in \mathbb{R}^{t\times s_m}$, with $\boldsymbol{S}_m^{(t)} \in \mathbb{R}^{t\times s_m}$ being the column-sampling matrix used to reduce the size of the approximate kernel matrix.

Note that solving Eq. (1) can impose substantial computational demands. To address this problem, the sketch-based method [Wang *et al.*, 2016] is proposed to reduce the computational complexity. Specifically, let $\boldsymbol{S}_p^{(t)} \in \mathbb{R}^{t\times s_p}$ represent the randomized sketch matrix based on the SJLT (details in Section 2.2). The sketched kernel matrix approximation problem at round t is then formulated as follows:

$$\begin{aligned}\boldsymbol{U}_{\text{fast}} = \arg\min_{\boldsymbol{U}} \Big\| \left(\boldsymbol{S}_p^{(t)}\right)^\top \boldsymbol{C}_m^{(t)} \boldsymbol{U} \left(\boldsymbol{C}_m^{(t)}\right)^\top \boldsymbol{S}_p^{(t)} - \\ \left(\boldsymbol{S}_p^{(t)}\right)^\top \boldsymbol{K}^{(t)} \boldsymbol{S}_p^{(t)} \Big\|_F^2 \\ = \left(\boldsymbol{\Phi}_{pm}^{(t)}\right)^\dagger \boldsymbol{\Phi}_{pp}^{(t)} \left(\boldsymbol{\Phi}_{pm}^{(t)\top}\right)^\dagger,\end{aligned} \tag{2}$$

where

$$\begin{aligned}\boldsymbol{\Phi}_{pm}^{(t)} &= \boldsymbol{S}_p^{(t)\top} \boldsymbol{C}_m^{(t)} \in \mathbb{R}^{s_p\times s_m}, \\ \boldsymbol{\Phi}_{pp}^{(t)} &= \boldsymbol{S}_p^{(t)\top} \boldsymbol{K}^{(t)} \boldsymbol{S}_p^{(t)} \in \mathbb{R}^{s_p\times s_p}.\end{aligned} \tag{3}$$

Instead of storing the entire kernel matrix, we can maintain smaller sketches for approximation. In the online setting, these sketches can be incrementally updated with the arrival of new data, $\boldsymbol{x}_{t+1}$. Specifically, the updates are given by:

$$\begin{aligned}\boldsymbol{\Phi}_{pm}^{(t+1)} &= \boldsymbol{\Phi}_{pm}^{(t)} + \boldsymbol{\Delta}_{pm}^{(t+1)}, \\ \boldsymbol{\Phi}_{pp}^{(t+1)} &= \boldsymbol{\Phi}_{pp}^{(t)} + \boldsymbol{\Delta}_{pp}^{(t+1)},\end{aligned} \tag{4}$$

where $\boldsymbol{\Delta}_{pm}^{(t+1)}$ and $\boldsymbol{\Delta}_{pp}^{(t+1)}$ are composed of three rank-1 matrices. Due to space constraints, the detailed construction is provided in Appendix B.1.

Next, we construct the time-varying explicit feature mapping. For simplicity, we use rank-k SVD, though it is inefficient due to the need for a full matrix decomposition. The elements of the kernel matrix are equal to the inner product of the corresponding points after feature mapping, i.e. $\boldsymbol{K}_{i,j} = \boldsymbol{\phi}(x_i)^\top \boldsymbol{\phi}(x_j)$. Once we build the approximate kernel matrix by Eq. (2), we can obtain a time-varying feature mapping through the rank-k SVD. Specifically, if

$$\boldsymbol{\Phi}_{pp}^{(t+1)} \approx \boldsymbol{V}^{(t+1)} \boldsymbol{\Sigma}^{(t+1)} \boldsymbol{V}^{(t+1)\top} \in \mathbb{R}^{s_p\times s_p}, \tag{5}$$

where $\boldsymbol{V}^{(t+1)} \in \mathbb{R}^{s_p\times k}$, $\boldsymbol{\Sigma}^{(t+1)} \in \mathbb{R}^{k\times k}$ and rank $k \le s_p$, we can update the time-varying explicit feature mapping $\boldsymbol{\phi}_{t+2} \in \mathbb{R}^k$ at round $t+1$ by

$$\boldsymbol{\phi}_{t+2}(\cdot) = \left([\kappa(\cdot, \tilde{\boldsymbol{x}}_1), \kappa(\cdot, \tilde{\boldsymbol{x}}_2), ..., \kappa(\cdot, \tilde{\boldsymbol{x}}_{s_m})]\boldsymbol{Z}_{t+1}\right)^\top,$$

where $\{\tilde{\boldsymbol{x}}_i\}_{i=1}^{s_m}$ are the sampled columns by $\boldsymbol{S}_m^{(t+1)}$ and $\boldsymbol{Z}_{t+1} = (\boldsymbol{\Phi}_{pm}^{(t+1)})^\dagger \boldsymbol{V}^{(t+1)} (\boldsymbol{\Sigma}^{(t+1)})^{\frac{1}{2}}$.

However, directly applying rank-k SVD to $\boldsymbol{\Phi}_{pp}$ is inefficient in online learning scenarios. Specifically, the standard rank-k SVD incurs a time complexity of $O(s_p^3)$ at each update, rendering it impractical in scenarios with frequent updates.

3.3 Novel Incremental Matrix Decomposition Method for Feature Mapping

In this section, we propose **TISVD** (Truncated Incremental Singular Value Decomposition), a novel incremental SVD method designed for decomposing sketches. TISVD achieves linear time and space complexity with respect to the sketch size s_p, effectively avoiding the time-consuming operation of performing a complete SVD.

We begin by presenting the construction of TISVD, which is well-suited for decomposing matrices with low-rank update properties. Without loss of generality, we denote a matrix at round t as $\boldsymbol{M}^{(t)} = \boldsymbol{U}^{(t)}\boldsymbol{\Sigma}^{(t)}\boldsymbol{V}^{(t)\top}$. In the $(t+1)$-th round, $\boldsymbol{M}^{(t)}$ is updated by low-rank matrices as follows:

$$\begin{aligned}\boldsymbol{M}^{(t+1)} &= \boldsymbol{M}^{(t)} + \boldsymbol{\Delta}_1\boldsymbol{\Delta}_2^\top \\ &= \boldsymbol{U}^{(t+1)}\boldsymbol{\Sigma}^{(t+1)} \left(\boldsymbol{V}^{(t+1)}\right)^\top,\end{aligned} \tag{6}$$

where $\boldsymbol{\Delta}_1, \boldsymbol{\Delta}_2 \in \mathbb{R}^{s_p\times c}$ of rank $r \le c \ll s_p$.

Our objective is to directly update the singular matrices $\boldsymbol{U}^{(t)}$, $\boldsymbol{\Sigma}^{(t)}$ and $\boldsymbol{V}^{(t)}$ using low-rank update matrices $\boldsymbol{\Delta}_1$ and $\boldsymbol{\Delta}_2$, resulting in $\boldsymbol{U}^{(t+1)}$, $\boldsymbol{\Sigma}^{(t+1)}$ and $\boldsymbol{V}^{(t+1)}$. First, we formulate orthogonal matrices through orthogonal projection and vertical projection. Let $\boldsymbol{P}, \boldsymbol{Q}$ denote the orthogonal basis of the column space of the following matrices:

$$\left(\boldsymbol{I} - \boldsymbol{U}^{(t)}\boldsymbol{U}^{(t)\top}\right)\boldsymbol{\Delta}_1 \quad \text{and} \quad \left(\boldsymbol{I} - \boldsymbol{V}^{(t)}\boldsymbol{V}^{(t)\top}\right)\boldsymbol{\Delta}_2,$$

respectively.

Set $\boldsymbol{R}_1 \doteq \boldsymbol{P}^\top\left(\boldsymbol{I} - \boldsymbol{U}^{(t)}\boldsymbol{U}^{(t)\top}\right)\boldsymbol{\Delta}_1$ and $\boldsymbol{R}_2 \doteq \boldsymbol{Q}^\top\left(\boldsymbol{I} - \boldsymbol{V}^{(t)}\boldsymbol{V}^{(t)\top}\right)\boldsymbol{\Delta}_2$, we can transform Eq. (6) into

$$\boldsymbol{M}^{(t+1)} = \begin{bmatrix}\boldsymbol{U}^{(t)} & \boldsymbol{P}\end{bmatrix} \boldsymbol{H} \begin{bmatrix}\boldsymbol{V}^{(t)} & \boldsymbol{Q}\end{bmatrix}^\top, \tag{7}$$

where

$$\boldsymbol{H} = \begin{bmatrix}\boldsymbol{\Sigma}^{(t)} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix} + \begin{bmatrix}\boldsymbol{U}^{(t)\top}\boldsymbol{\Delta}_1 \\ \boldsymbol{R}_1\end{bmatrix} \begin{bmatrix}\boldsymbol{V}^{(t)\top}\boldsymbol{\Delta}_2 \\ \boldsymbol{R}_2\end{bmatrix}^\top. \tag{8}$$

Subsequently, as the size of $\boldsymbol{H}$ is smaller than $\boldsymbol{M}^{(t+1)}$, an efficient computation of $\tilde{\boldsymbol{U}}_k$, $\tilde{\boldsymbol{V}}_k$, and $\tilde{\boldsymbol{\Sigma}}_k$ can be obtained by

performing a truncated rank-k SVD on $\boldsymbol{H}$. Since the matrices on the left and right sides are column orthogonal, we finally obtain $\boldsymbol{U}^{(t+1)}$, $\boldsymbol{V}^{(t+1)}$, and $\boldsymbol{\Sigma}^{(t+1)}$ at round $t+1$:

$$\begin{aligned}\boldsymbol{U}^{(t+1)} &= \begin{bmatrix}\boldsymbol{U}^{(t)} & \boldsymbol{P}\end{bmatrix} \tilde{\boldsymbol{U}}_k,\\ \boldsymbol{V}^{(t+1)} &= \begin{bmatrix}\boldsymbol{V}^{(t)} & \boldsymbol{Q}\end{bmatrix} \tilde{\boldsymbol{V}}_k,\\ \boldsymbol{\Sigma}^{(t+1)} &= \tilde{\boldsymbol{\Sigma}}_k.\end{aligned} \tag{9}$$

We summarize the algorithm and provide the pseudo-code for TISVD in Appendix B.2. Note that the sketch $\boldsymbol{\Phi}_{pp}^{(t+1)}$ can be updated using low-rank matrices, as shown in Eq. (4). By replacing the standard rank-k SVD with TISVD, we establish an efficient mechanism for the incremental maintenance of singular matrices. More precisely, in Eq. (5), we update $\boldsymbol{V}^{(t+1)}$ and $\boldsymbol{\Sigma}^{(t+1)}$ using their previous counterparts, $\boldsymbol{V}^{(t)}$ and $\boldsymbol{\Sigma}^{(t)}$, along with the low-rank update $\boldsymbol{\Delta}_{pp}^{(t+1)}$ at round $t+1$.

Remark 1 (Complexity)**.** *The complexity of TISVD is primarily composed of the truncated rank-k SVD on $\boldsymbol{H}$ and the matrix multiplication in Eq.* (9)*. The update cost of the truncated rank-k SVD is $O(k^3)$, while the matrix multiplication in Eq.* (9) *incurs a complexity of $O(s_p k)$. Consequently, compared to standard rank-k SVD, TISVD provides significant improvements by reducing the time complexity from $O(s_p^3)$ to $O(s_p k + k^3)$ and the space complexity from $O(s_p^2)$ to $O(s_p k + k^2)$. More discussion is provided in Appendix B.3.*

3.4 Efficient Second-Order Online Kernel Learning

Since the efficient time-varying explicit feature mapping $\phi_t(\cdot)$ has been constructed, we can formulate the approximate hypothesis $f_t(\boldsymbol{x}_t)$ at round t as follows

$$f_t(\boldsymbol{x}_t) = \boldsymbol{w}_t^\top \boldsymbol{\phi}_t(\boldsymbol{x}_t),$$

where $\boldsymbol{w}_t$ is the weight vector. On the basis of the hypothesis, we propose a two-stage second-order online kernel learning method, named **FORKS** (Fast Second-Order Online Kernel Learning Using Incremental Sketching).

In the first stage, we simply collect the items with nonzero losses to the buffer SV and perform the Kernelized Online Gradient Descent (KOGD) [Kivinen *et al.*, 2001]. When the size of the buffer reaches a fixed budget B, we calculate $\boldsymbol{K}_t$ and initialize sketch matrices $\boldsymbol{\Phi}_{pp}^{(t)}, \boldsymbol{\Phi}_{pm}^{(t)}$ in Eq. (3).

In the second stage, we adopt a periodic updating strategy for sketches. More precisely, we update $\boldsymbol{\Phi}_{pp}^{(t)}, \boldsymbol{\Phi}_{pm}^{(t)}$ by Eq. (4) once for every ρ examples, where $\rho \in [T-B]$ is defined as update cycle. Furthermore, we incrementally update the feature mapping $\phi_t(\cdot)$ by TISVD.

In addition to updating the feature mapping $\phi_t(\cdot)$, we perform second-order updates on the $\boldsymbol{w}_t$. Specifically, we update the hypothesis using Online Newton Step (ONS) [Hazan *et al.*, 2007] for some parameters $\alpha > 0$ and $\sigma_i, \eta_i \geq 0$:

$$\begin{aligned}\boldsymbol{v}_{t+1} &= \boldsymbol{w}_t - \boldsymbol{A}_t^{-1}\boldsymbol{g}_t,\\ \boldsymbol{w}_{t+1} &= \boldsymbol{v}_{t+1} - \frac{h\left(\boldsymbol{\phi}_{t+1}^\top \boldsymbol{v}_{t+1}\right)}{\boldsymbol{\phi}_{t+1}^\top \boldsymbol{A}_t^{-1}\boldsymbol{\phi}_{t+1}} \boldsymbol{A}_t^{-1}\boldsymbol{\phi}_{t+1},\end{aligned} \tag{10}$$

where $\boldsymbol{g}_t = \nabla_{\boldsymbol{w}_t}\ell_t(\hat{y}_t)$, $\boldsymbol{A}_t = \alpha\boldsymbol{I} + \sum_{i=0}^{t}(\sigma_i+\eta_i)\boldsymbol{g}_i\boldsymbol{g}_i^\top$ and $h(z) = \operatorname{sign}(z)\max(|z| - C, 0)$. The second-order updates not only consider the gradient information but also utilize the curvature information of the loss function, leading to faster convergence rates.

Algorithm 1 FORKS

Input: Data stream $\{(\boldsymbol{x}_t, y_t)\}_{t=1}^T$, sketch size s_p, sample size s_m, rank k, budget B, update cycle ρ, regularizer α
Output: Predicted label $\{\hat{y}_t\}_{t=1}^T$

for $t \leftarrow 1, \ldots, T$ **do**
  Receive $\boldsymbol{x}_t$
  while SV is not full **do**
    Update hypothesis by KOGD
    Add $\boldsymbol{x}_t$ to SV whenever the loss is nonzero
  end while
  if SV is full **then**
    Initialize $\boldsymbol{\Phi}_{pp}^{(t)}, \boldsymbol{\Phi}_{pm}^{(t)}$ as in Eq. (3)
    Compute the mapping ϕ_{t+1}
  else if in the update round **then**
    Update ϕ_{t+1} by TISVD (Algorithm 2)
    Set $\boldsymbol{A}_t \leftarrow \alpha\boldsymbol{I}, \quad \boldsymbol{w}_t \leftarrow \boldsymbol{0}$
  else
    $\boldsymbol{\Phi}_{pp}^{(t)} \leftarrow \boldsymbol{\Phi}_{pp}^{(t-1)}, \boldsymbol{\Phi}_{pm}^{(t)} \leftarrow \boldsymbol{\Phi}_{pm}^{(t-1)}, \phi_{t+1} \leftarrow \phi_t$
  end if
  Predict $\hat{y}_t = \operatorname{sgn}\left(\boldsymbol{\phi}_t^\top \boldsymbol{w}_t\right)$
  # Execute a second-order gradient descent
  Obtain $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{w}_t}\ell_t\left(\hat{y}_t\right)$
  Update $\boldsymbol{A}_{t+1} \leftarrow \boldsymbol{A}_t + (\sigma_i+\eta_i)\boldsymbol{g}_t\boldsymbol{g}_t^\top$
  Compute $\boldsymbol{w}_{t+1}, \boldsymbol{v}_{t+1}$ using Eq. (10)
end for

At the start of a new update epoch, we incorporate a *reset* step before applying the gradient descent in the new embedded space. We update the feature mapping ϕ_t but reset $\boldsymbol{A}_t$ and $\boldsymbol{w}_t$. This step is taken to ensure that our starting point cannot be influenced by the adversary. By leveraging efficient second-order updates, we can effectively converge to the optimal hypothesis within the current subspace. Furthermore, the reset of the descent procedure when transitioning between subspaces ensures a stable starting point and maintains a bounded regret throughout the entire process. We summarize the above stages into Algorithm 1.

The incremental update of the feature mapping, combined with the reset mechanism, ensures that we can effectively capture changes in user preferences over time. More specifically, our algorithm updates the feature vector to reflect the user's current state, while resetting the Hessian matrix mitigates the influence of outdated data on the learner.

3.5 Complexity Analysis of FORKS

Given the budget of B, FORKS consists of three parts: (1) the first stage using KOGD, (2) the updating round in the second stage, and (3) the regular round in the second stage. At the first stage ($|SV| \leq B$), FORKS has constant time $O(B)$ and space complexities $O(B)$ per round.

The main computational complexity of FORKS during the update round stems from the matrix decomposition and inversion procedures. These processes are necessary for updating the feature mapping and performing second-order updates, respectively. TISVD reduces the time complexity of $\boldsymbol{\Phi}_{pp}$

decomposition from $O(s_p^3)$ to $O(s_pk + k^3)$, where s_p is the sketch size of $\boldsymbol{S}_p$ and k is the rank in TISVD. A naive implementation of the second-order update requires $O(k^3)$ per-step time and has a space complexity of $O(k^2)$ necessary to store the Hessian $\boldsymbol{A}_t$. By taking advantage of the fact that $\boldsymbol{A}_t$ is rank-1 updated, we can reduce the per-step cost to $O(k^2)$.

We denote the update cycle as ρ and $\mu = B + \lfloor (T-B)/\rho \rfloor$. To summarize, the time complexity of FORKS at each updating round is

$$O\left(\mu + s_p k^2 + s_m s_p k + k^3\right),$$

and the space complexity is $O(\mu + s_p k + s_m s_p + k^2)$, where s_m is the sketch size of $\boldsymbol{S}_m$.

The primary time consumption for the online learning algorithm occurs during regular rounds. At each regular round, the time complexity of FORKS is $O(s_m k + k^2)$. Given that $s_m < s_p < B$, our algorithm achieves a time complexity of $O(Bk+k^2)$ per step. The current state-of-the-art second-order online kernel learning method, PROS-N-KONS, presents a time complexity of $O(B^2)$ per step in its budget version [Calandriello *et al.*, 2017a]. Moreover, due to the changing size of the subspace from online sampling, PROS-N-KONS must perform a complete SVD at each step. In contrast, FORKS introduces substantial advancements by reducing the time complexity from $O(B^2)$ to $O(Bk + k^2)$.

4 Regret Analysis

In this section, we provide the regret analysis for the proposed second-order online kernel learning algorithm. We begin by making the following assumptions about the loss functions.

Assumption 1 (Lipschitz Continuity)**.** *ℓ is Lipschitz continuous with the Lipschitz constant L_{Lip}, i.e., $\|\nabla \ell(\boldsymbol{w})\|_2 \leq L_{\mathrm{Lip}}$.*

Assumption 2 (Directional Curvature)**.** *Let $L_{\mathrm{Cur}} \geq 0$. Then, for any vectors $\boldsymbol{w}_1, \boldsymbol{w}_2$ and the convex function ℓ, denote that $\Delta = \ell(\boldsymbol{w}_1) - \ell(\boldsymbol{w}_2)$, we have*

$$\Delta \geq \langle \nabla \ell(\boldsymbol{w}_2), \boldsymbol{w}_1 - \boldsymbol{w}_2 \rangle + \frac{L_{\mathrm{Cur}}}{2} \langle \nabla \ell(\boldsymbol{w}_2), \boldsymbol{w}_1 - \boldsymbol{w}_2 \rangle^2.$$

In practical scenarios, the assumption of *strong convexity* may not always hold as it imposes constraints on the convexity of losses in all directions. A more feasible approach is to relax this assumption by demanding strong convexity only in the gradient direction, which is a weaker condition as indicated by the two assumptions above. For example, exp-concave losses like squared loss and squared hinge loss satisfy Assumption 2.

Assumption 3 (Matrix Product Preserving)**.** *Let $\boldsymbol{S}_p \in \mathbb{R}^{T \times s_p}$ be a sketch matrix, $\boldsymbol{U}_{\mathrm{m}} \in \mathbb{R}^{T \times s_{\mathrm{m}}}$ be a matrix with orthonormal columns, $\boldsymbol{U}_{\mathrm{m}}^{\perp} \in \mathbb{R}^{T \times (T - s_{\mathrm{m}})}$ be another matrix satisfying $\boldsymbol{U}_{\mathrm{m}} \boldsymbol{U}_{\mathrm{m}}^{\top} + \boldsymbol{U}_{\mathrm{m}}^{\perp} (\boldsymbol{U}_{\mathrm{m}}^{\perp})^{\top} = \boldsymbol{I}_T$ and $\boldsymbol{U}_{\mathrm{m}}^{\top} \boldsymbol{U}_{\mathrm{m}}^{\perp} = \boldsymbol{O}$, and δ_i $(i = 1, 2)$ be the failure probabilities defined as follows:*

$$\Pr\left\{ \frac{\|\boldsymbol{B}_i \boldsymbol{A}_i - \boldsymbol{B}_i \boldsymbol{S}_p \boldsymbol{S}_p^{\top} \boldsymbol{A}_i\|_F^2}{2 \|\boldsymbol{B}_i\|_F^2 \|\boldsymbol{A}_i\|_F^2} > \frac{1}{\delta_i s_p} \right\} \leq \delta_i, \quad i = 1, 2,$$

where $\boldsymbol{A}_1 = \boldsymbol{U}_{\mathrm{m}}$, $\boldsymbol{B}_1 = \boldsymbol{I}_T$, $\boldsymbol{A}_2 = \boldsymbol{U}_{\mathrm{m}}^{\perp} (\boldsymbol{U}_{\mathrm{m}}{}^{\perp})^{\top} \boldsymbol{K}$, $\boldsymbol{B}_2 = \boldsymbol{U}_{\mathrm{m}}^{\top}$, $\boldsymbol{K} \in \mathbb{R}^{T \times T}$ is a kernel matrix.

The conditions stated in Assumption 3 can be satisfied by SJLT matrix [Woodruff, 2014]. Given the loss $\ell_t(\boldsymbol{w}_t) := \ell_t(f_t) = \ell(f_t(\boldsymbol{x}_t), y_t), \forall t \in [T]$ satisfies Assumption 1 and Assumption 2, we bound the following regret:

$$\mathrm{Reg}_T(f^*) = \sum_{t=1}^{T} \left[\ell_t(\boldsymbol{w}_t) - \ell_t(f^*) \right],$$

where $f^* = \arg\min_{f \in \mathcal{H}_\kappa} \sum_{t=1}^{T} \ell_t(f)$ denotes the optimal hypothesis in hindsight in the original RKHS.

Let $\boldsymbol{K} \in \mathbb{R}^{T \times T}$ be a kernel matrix with $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq 1$ for all i, j. Let $\rho = \lfloor \theta(T - B) \rfloor$ denote the update cycle, where $\theta \in (0, 1)$, and let k represent the rank in TISVD. Additionally, define C_{Coh} as the coherence of the intersection matrix of $\boldsymbol{K}$, which is constructed using $B + \lfloor (T-B)/\rho \rfloor$ examples. Let $\boldsymbol{v}_i$ be the i-th singular vector of $\boldsymbol{K}$, the coherence is given by:

$$C_{\mathrm{Coh}} = \left(\frac{B + 1/\theta}{\mathrm{rank}(\boldsymbol{K})} \right) \max_i \|\boldsymbol{v}_i\|_2^2.$$

Note that C_{Coh} is independent of T when $\rho = \lfloor \theta(T - B) \rfloor$. We demonstrate the regret upper bound of FORKS as follows

Theorem 1 (Regret Bound of FORKS)**.** *Let $\delta_0, \epsilon_0 \in (0, 1)$. Set the number of submatrices in SJLT as $D = \Theta(\log^3(s_m))$. Suppose the parameters for updating $\boldsymbol{A}_t$ in FORKS satisfy $\eta_i = 0$ and $\sigma_i \geq L_{\mathrm{Cur}} > 0$. Assume the eigenvalues of $\boldsymbol{K}$ decay polynomially with decay rate $\beta > 1$, and that the SJLT $\boldsymbol{S}_p$ satisfies Assumption 3 with failure probabilities $\delta_1, \delta_2 \in (0, 1)$. If the sketch sizes of $\boldsymbol{S}_p$ and $\boldsymbol{S}_m$ satisfy*

$$s_p = \Omega\left(s_m \, \mathrm{polylog}(s_m \delta_0^{-1}) / \epsilon_0^2 \right), \; s_m = \Omega(C_{\mathrm{Coh}} k \log k).$$

Then, with probability at least $1 - \delta$,

$$\begin{aligned} \mathrm{Reg}_T(f^*) \leq \frac{\alpha D_{\boldsymbol{w}}^2}{2} + \frac{k}{2L_{\mathrm{Cur}}} O(\log T) + \frac{\lambda}{2} \|f^*\|_{\mathcal{H}_\kappa}^2 + \\ \frac{\sqrt{1+\epsilon}}{\lambda} O\left(\sqrt{B}\right) + \frac{(k+1)^{-\beta}}{2\lambda\theta} + \\ \frac{1}{\lambda(\beta - 1)} \left(\frac{3}{2} - \frac{B + \lfloor (T-B)/\rho \rfloor}{T} \right). \end{aligned}$$

where $\delta = \delta_0 + \delta_1 + \delta_2$, $D_{\boldsymbol{w}}$ denotes the diameter of the weight vector space of the hypothesis on the incremental sketches, and ϵ is defined as:

$$\sqrt{\epsilon} = 2\gamma \sqrt{\frac{T}{\delta_1 \delta_2}} + \sqrt{\frac{2\gamma}{\delta_2}} \left(\epsilon_0^2 + 2\epsilon_0 + 2 \right),$$

with $\gamma = s_m / s_p$.

proof sketch. Let $\boldsymbol{w}^*$ denote the optimal hypothesis on the incremental sketches in hindsight, we decompose the instantaneous regret $\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}^*)$ into two terms as follows

$$\underbrace{\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}_t^*)}_{\text{Term 1: Optimization Error}} + \underbrace{\ell_t(\boldsymbol{w}_t^*) - \ell_t(\boldsymbol{w}^*)}_{\text{Term 2: Estimation Error}}.$$

The optimization error stems from the optimization step of second-order online gradient descent, while the estimation error arises from the incremental matrix sketching and the truncated singular values in TISVD. We provide upper bounds for the errors and present the detailed proof in Appendix C.1. □

Remark 2 (Assumption of polynomial decay)**.** *The assumption of polynomial eigenvalue decay for the kernel matrix is widely applicable and holds for various types of kernels, including shift-invariant, finite-rank, and convolution kernels [Liu and Liao, 2015; Belkin, 2018]. This decay property guarantees that the accumulated truncated singular values of TISVD can be upper bounded by the number of update rounds.*

Remark 3 (Convex case)**.** *Note that when $L_{\text{Cur}} = 0$, Assumption 2 essentially enforces convexity. In the worst case when $L_{\text{Cur}} = 0$, the regret bound in the convex case degenerates to $O(\sqrt{T})$. The detailed proof is included in Appendix C.2.*

By configuring the update cycle as $\rho = \lfloor \theta(T - B) \rfloor$, where $\theta \in (0, 1)$, and setting the sketch size ratio to $\gamma = O(\log(T)/\sqrt{T})$, we derive an upper bound on regret of $O(T^{\frac{1}{4}} \log T)$ for second-order online kernel learning. In contrast, the regret bound for first-order online kernel learning is $O(\sqrt{T})$, while requiring a budget of $B = O(T)$ [Lu *et al.*, 2016a], making it less favorable than FORKS.

Under the assumption that the eigenvalues of the kernel matrix decay polynomially with rate β, the regret bound for the existing second-order online kernel learning algorithm PROS-N-KONS [Calandriello *et al.*, 2017a] is $O(T^{\frac{1}{\beta}} \log^2 T)$, which is highly sensitive to the eigenvalue decay rate β. In practical streaming scenarios, however, ensuring a sufficiently rapid decay of the kernel matrix is often unrealistic. When the properties of the streaming matrix are less favorable, such as in the case where $\beta < 4$, the regret of PROS-N-KONS can be worse than that of FORKS.

5 Experiments

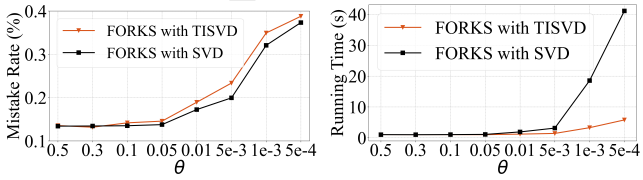
In this section, we conduct experiments to evaluate the performance of FORKS on several datasets. The details of datasets and experimental setup are presented in Appendix D.1, D.2.

5.1 Experiments Under a Fixed Budget

In this section, we demonstrate the performance of FORKS under a fixed budget, employing six widely recognized classification benchmarks. We compare FORKS with the existing budgeted-based online learning algorithms, including first-order algorithms RBP [Cavallanti *et al.*, 2007], BPA-S [Wang and Vucetic, 2010], BOGD [Hoi *et al.*, 2012], FOGD, NOGD [Lu *et al.*, 2016a], Projectron [Orabona *et al.*, 2008], SkeGD [Zhang and Liao, 2019] and second-order algorithm PROS-N-KONS [Calandriello *et al.*, 2017a]. All algorithms are trained using hinge loss, and their performance is measured by the average online mistake rate.

For all the algorithms, we set a fixed budget $B = 50$ for small datasets ($N \leq 10000$) and $B = 100$ for large datasets. Furthermore, we set buffer size $\tilde{B} = 2B, \gamma = 0.2$, $s_p = B$, $s_m = \gamma s_p$, $\theta = 0.3$, and update cycle $\rho = \lfloor \theta N \rfloor$ in SkeGD and FORKS if not specially specified. For algorithms with rank-k approximation, we uniformly set $k = 0.1B$. Besides, we use the same experimental settings for FOGD (feature dimension = 4B). The results are presented in Table 1. Our FORKS shows the best performance on `svmguide3`, `codrna`, `w7a`, and the suboptimal performance on other datasets. The update time of FORKS is comparable to that of the majority of first-order algorithms, including NOGD and SkeGD. Besides, FORKS is significantly more efficient than the existing second-order method PROS-N-KONS in large-scale datasets such as `codrna` and `w7a`.

Then, we conduct experiments to evaluate how TISVD affects the performance of the algorithm. We use the same experimental setup in `codrna` and vary the update rate θ from 0.5 to 0.0005. Figure 2 demonstrates that TISVD maintains efficient decomposition speed without excessively reducing performance. Considering that frequent updates can potentially result in an elevated loss, it is essential to carefully choose an optimal update cycle that strikes a balance between achieving superior accuracy and maintaining efficiency.



(a) average mistake rate (b) average running time

Figure 2: The average mistake rates and average running time w.r.t. TISVD on `codrna`.

5.2 Experiments Under Adversarial Environment

To empirically validate the algorithms under an adversarial environment, we build adversarial datasets using the benchmark `codrna` and `german`. We compare FORKS with first-order algorithms BOGD [Hoi *et al.*, 2012], SkeGD [Zhang and Liao, 2019], NOGD [Lu *et al.*, 2016a] and second-order algorithm PROS-N-KONS [Calandriello *et al.*, 2017a] under the same budget $B = 200$. Besides, we set $\gamma = 0.2$, $s_p = 0.75B$, $s_m = \gamma s_p, k = 0.1B$ and update cycle $\rho = \lfloor 0.005(N - B) \rfloor$ in SkeGD and FORKS. Inspired by the adversarial settings in [Calandriello *et al.*, 2017a; Wang *et al.*, 2018; Zhang and Liao, 2019], we generate an online learning game with b blocks. At each block, we extract an instance from the dataset and repeat it for r rounds. In addition, the labels are flipped in each even block by multiplying them with -1. We set $b = 500$, $r = 10$ for `codrna-1` and `german-1`.

Experimental results are presented in Table 2. It is observed that in the adversarial environment, the performance of all methods significantly decreases with the increase of adversarial changes except for FORKS. This is due to the fact that FORKS accurately captures the concept drifting through the incremental update of the sketch matrix and the execution of rapid second-order gradient descent. Moreover, FORKS maintains its efficiency comparable to first-order algorithms, thereby ensuring that improved performance is achieved without sacrificing computational time.

5.3 Experiments on Large-Scale Datasets

In this experiment, we evaluate the efficiency and effectiveness of FORKS on streaming recommendation. We use `KuaiRec`, which is a real-world dataset collected from the recommendation logs of the video-sharing mobile app Kuaishou [Gao *et al.*, 2022]. We conduct experiments on the dense matrix of

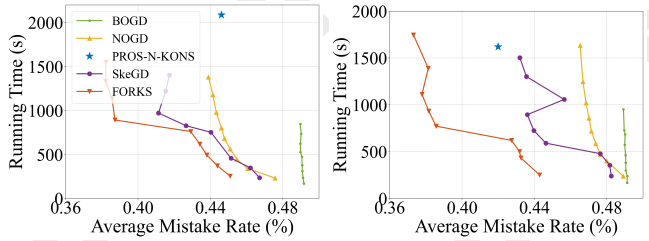
Algorithm	german		svmguide3		spambase	
	Mistake rate	Time	Mistake rate	Time	Mistake rate	Time
RBP	38.830 ± 0.152	0.003	29.698 ± 1.644	0.003	35.461 ± 0.842	0.025
BPA-S	35.235 ± 0.944	0.004	29.027 ± 0.732	0.004	34.394 ± 2.545	0.039
Projectron	36.875 ± 1.403	0.003	25.060 ± 0.373	0.003	32.659 ± 0.914	0.031
BOGD	33.705 ± 1.446	0.007	29.904 ± 1.653	0.006	32.859 ± 0.478	0.049
FOGD	30.915 ± 0.845	0.025	30.024 ± 0.787	0.022	**25.651 ± 0.349**	0.175
NOGD	26.715 ± 0.552	0.014	19.964 ± 0.077	0.008	31.003 ± 0.751	0.077
SkeGD	**25.170 ± 0.391**	0.009	19.976 ± 0.105	0.007	32.413 ± 1.886	0.067
PROS-N-KONS	31.235 ± 0.939	1.017	24.529 ± 0.561	0.015	32.227 ± 0.678	6.638
FORKS (Ours)	26.425 ± 0.562	0.008	**19.710 ± 0.557**	0.009	30.662 ± 0.670	0.070

Algorithm	codrna		w7a		ijcnn1	
	Mistake rate	Time	Mistake rate	Time	Mistake rate	Time
RBP	22.644 ± 0.262	0.210	5.963 ± 0.722	0.945	21.024 ± 0.578	0.633
BPA-S	17.029 ± 0.303	0.313	3.001 ± 0.045	1.145	11.114 ± 0.064	0.747
Projectron	19.257 ± 4.688	0.341	3.174 ± 0.014	0.965	9.478 ± 0.001	0.621
BOGD	17.305 ± 0.146	0.507	3.548 ± 0.164	0.970	11.559 ± 0.174	0.724
FOGD	13.103 ± 0.105	1.480	2.893 ± 0.053	2.548	9.674 ± 0.105	3.125
NOGD	17.915 ± 3.315	0.869	2.579 ± 0.007	2.004	**9.379 ± 0.001**	1.457
SkeGD	13.274 ± 0.262	0.779	2.706 ± 0.335	2.093	11.898 ± 1.440	2.216
PROS-N-KONS	13.387 ± 0.289	114.983	3.016 ± 0.007	92.377	9.455 ± 0.001	5.000
FORKS (Ours)	**12.795 ± 0.360**	0.918	**2.561 ± 0.038**	2.240	9.381 ± 0.001	2.480

Table 1: Comparisons among first-order algorithms RBP, BPA-S, BOGD, Projectron, NOGD, SkeGD, FOGD and second-order algorithms PROS-N-KONS, FORKS w.r.t. the mistake rates (%) and the running time (s). The best result is highlighted in **bold** font, and the second best result is underlined.

Algorithm	codrna-1		german-1	
	Mistake rate	Time	Mistake rate	Time
BOGD	26.066 ± 1.435	0.029	32.131 ± 1.079	0.042
NOGD	29.780 ± 1.257	0.024	28.103 ± 1.247	0.040
SkeGD	24.649 ± 5.087	0.269	11.026 ± 4.018	0.113
PROS-N-KONS	21.299 ± 1.364	3.323	17.174 ± 1.437	0.477
FORKS (Ours)	**6.752 ± 1.647**	0.023	**5.142 ± 0.215**	0.035

Table 2: Comparisons among BOGD, NOGD, PROS-N-KONS, SkeGD and our FORKS w.r.t. the mistake rates (%) and the running time (s). The best result is highlighted in **bold** font.



(a) Hinge loss (b) Squared hinge loss

Figure 3: The tradeoff between running time and the average mistake rate on KuaiRec. As PROS-N-KONS utilizes an adaptive budget, thereby being depicted as a single point in the figures.

KuaiRec, which consists of $4,494,578$ instances with associated timestamps, making it an ideal benchmark for evaluating large-scale online learning tasks. We test the performance of the algorithm used in Section 5.3 under different budgets B ranging from 100 to 500. To avoid excessive training time, we use a budgeted version of PROS-N-KONS that stops updating the dictionary at a maximum budget of $B_{\max} = 100$. Since the buffer size of PROS-N-KONS is data-dependent, we repeat the training process 20 times to compute the average error rate and the average time for comparison. In addition to the hinge loss, we use squared hinge loss to evaluate the performance.

Figure 3 (a) shows the tradeoff between running time and the average mistake rate in the experiment using hinge loss. Figure 3 (b) shows the tradeoff between running time and the average mistake rate in the experiment using squared hinge loss. We observe that FORKS consistently achieves superior learning performance while maintaining comparable time costs to the other first-order algorithms, regardless of the loss function's shape. In particular, for squared hinge loss, both PROS-N-KONS and FORKS significantly outperform first-order models, highlighting the advantages of second-order methods under exp-concave losses. We also observe that FORKS demonstrates considerably higher efficiency than the second-order algorithm PROS-N-KONS. Under squared hinge loss, to achieve a comparable online error rate, FORKS requires only approximately 500 seconds, while PROS-N-KONS takes over 1500 seconds, resulting in a threefold speedup.

6 Conclusion

This paper introduces FORKS, a fast second-order online kernel learning approach. By leveraging incremental matrix sketching and decomposition techniques, FORKS efficiently addresses the computational challenges inherent in kernel feature mapping and hypothesis updates. The proposed method achieves a logarithmic regret bound while maintaining linear time complexity relative to the budget, significantly improving the efficiency of existing second-order methods. Extensive experimental evaluations on various datasets demonstrate the superior scalability and robustness of our approach.

Acknowledgments

This research was supported in part by National Science and Technology Major Project (2022ZD0114802), by National Natural Science Foundation of China (No. 92470128, No. U2241212, No. 62376275), by Beijing Outstanding Young Scientist Program No.BJJWZYJH012019100020098, by Huawei-Renmin University joint program on Information Retrieval. We also wish to acknowledge the support provided by the fund for building world-class universities (disciplines) of Renmin University of China, by Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education, by Intelligent Social Governance Interdisciplinary Platform, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, Public Policy and Decision-making Research Lab, and Public Computing Cloud, Renmin University of China. The work was partially done at Beijing Key Laboratory of Research on Large Models and Intelligent Governance, MOE Key Lab of Data Engineering and Knowledge Engineering, Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE, and Pazhou Laboratory (Huangpu), Guangzhou, Guangdong 510555, China.

References

[Belkin, 2018] Mikhail Belkin. Approximation beats concentration? an approximation view on inference with smooth radial kernels. In *COLT*, volume 75, pages 1348–1361, 2018.

[Calandriello *et al.*, 2017a] Daniele Calandriello, Alessandro Lazaric, and Michal Valko. Efficient second-order online kernel learning with adaptive embedding. In *NIPS*, pages 6140–6150, 2017.

[Calandriello *et al.*, 2017b] Daniele Calandriello, Alessandro Lazaric, and Michal Valko. Second-order kernel online convex optimization with adaptive sketching. In *ICML*, volume 70, pages 645–653, 2017.

[CAO *et al.*, 2017] Lele CAO, Fuchun SUN, Hongbo LI, and Wenbing HUANG. Advancing the incremental fusion of robotic sensory features using online multi-kernel extreme learning machine. *Frontiers of Computer Science*, 11:276, 2017.

[Cavallanti *et al.*, 2007] Giovanni Cavallanti, Nicolò Cesa-Bianchi, and Claudio Gentile. Tracking the best hyperplane with a simple budget perceptron. *Mach. Learn.*, 69(2-3):143–167, 2007.

[Charikar *et al.*, 2002] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *ICALP*, volume 2380, pages 693–703, 2002.

[CHEN *et al.*, 2022] Cheng CHEN, Weinan ZHANG, and Yong YU. Efficient policy evaluation by matrix sketching. *Frontiers of Computer Science*, 16:165330, 2022.

[Gao *et al.*, 2022] Chongming Gao, Shijun Li, Wenqiang Lei, Jiawei Chen, Biao Li, Peng Jiang, Xiangnan He, Jiaxin Mao, and Tat-Seng Chua. Kuairec: A fully-observed dataset and insights for evaluating recommender systems. In *CIKM*, pages 540–550, 2022.

[Hazan *et al.*, 2007] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Mach. Learn.*, 69(2-3):169–192, 2007.

[Hoi *et al.*, 2012] Steven C. H. Hoi, Jialei Wang, Peilin Zhao, Rong Jin, and Pengcheng Wu. Fast bounded online gradient descent algorithms for scalable kernel-based online learning. In *ICML*, 2012.

[Hu *et al.*, 2015] Junjie Hu, Haiqin Yang, Irwin King, Michael R. Lyu, and Anthony Man-Cho So. Kernelized online imbalanced learning with fixed budgets. In *AAAI*, pages 2666–2672, 2015.

[Kane and Nelson, 2014] Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, 2014.

[Kivinen *et al.*, 2001] Jyrki Kivinen, Alexander J. Smola, and Robert C. Williamson. Online learning with kernels. In *NIPS*, pages 785–792, 2001.

[Liu and Liao, 2015] Yong Liu and Shizhong Liao. Eigenvalues ratio for kernel selection of kernel methods. In *AAAI*, pages 2814–2820, 2015.

[Lu *et al.*, 2016a] Jing Lu, Steven C. H. Hoi, Jialei Wang, Peilin Zhao, and Zhiyong Liu. Large scale online kernel learning. *J. Mach. Learn. Res.*, 17:47:1–47:43, 2016.

[Lu *et al.*, 2016b] Jing Lu, Peilin Zhao, and Steven C. H. Hoi. Online sparse passive aggressive learning with kernels. In *SDM*, pages 675–683, 2016.

[Orabona *et al.*, 2008] Francesco Orabona, Joseph Keshet, and Barbara Caputo. The projectron: a bounded kernel-based perceptron. In *ICML*, volume 307, pages 720–727, 2008.

[Sahoo *et al.*, 2019] Doyen Sahoo, Steven C. H. Hoi, and Bin Li. Large scale online multiple kernel regression with application to time-series prediction. *ACM Trans. Knowl. Discov. Data*, 13(1):9:1–9:33, 2019.

[Shalev-Shwartz, 2012] Shai Shalev-Shwartz. Online learning and online convex optimization. *Found. Trends Mach. Learn.*, 4(2):107–194, 2012.

[Singh *et al.*, 2012] Abhishek Singh, Narendra Ahuja, and Pierre Moulin. Online learning with kernels: Overcoming the growing sum problem. In *MLSP*, pages 1–6, 2012.

[Wang and Vucetic, 2010] Zhuang Wang and Slobodan Vucetic. Online passive-aggressive algorithms on a budget. In *AISTATS*, volume 9, pages 908–915, 2010.

[Wang *et al.*, 2016] Shusen Wang, Luo Luo, and Zhihua Zhang. SPSD matrix approximation vis column selection: Theories, algorithms, and extensions. *J. Mach. Learn. Res.*, 17:49:1–49:49, 2016.

[Wang *et al.*, 2018] Guanghui Wang, Dakuan Zhao, and Lijun Zhang. Minimizing adaptive regret with one gradient per iteration. In *IJCAI*, pages 2762–2768, 2018.

[Williams and Seeger, 2000] Christopher K. I. Williams and Matthias W. Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, pages 682–688, 2000.

[Woodruff, 2014] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Found. Trends Theor. Comput. Sci.*, 10(1-2):1–157, 2014.

[Zhang and Liao, 2018] Xiao Zhang and Shizhong Liao. Online kernel selection via incremental sketched kernel alignment. In *IJCAI*, pages 3118–3124, 2018.

[Zhang and Liao, 2019] Xiao Zhang and Shizhong Liao. Incremental randomized sketching for online kernel learning. In *ICML*, volume 97, pages 7394–7403, 2019.

[Zhang *et al.*, 2023] Xiao Zhang, Ninglu Shao, Zihua Si, Jun Xu, Wenhan Wang, Hanjing Su, and Ji-Rong Wen. Reward imputation with sketching for contextual batched bandits. *NIPS*, pages 64577–64588, 2023.

[Zhdanov and Kalnishkan, 2013] Fedor Zhdanov and Yuri Kalnishkan. An identity for kernel ridge regression. *Theor. Comput. Sci.*, 473:157–178, 2013.

[Zhu and Xu, 2015] Changbo Zhu and Huan Xu. Online gradient descent in function space. *arXiv*, 2015.

[Zinkevich, 2003] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, pages 928–936, 2003.