Received: 1 August 2022 | Revised: 27 September 2022 | Accepted: 7 October 2022

DOI: 10.1002/sta4.514

ORIGINAL ARTICLE

WILEY

Adaptive supervised learning on data streams in reproducing kernel Hilbert spaces with data sparsity constraint

Haodong Wang[1] | Quefeng Li[1] | Yufeng Liu[1,2,3,4,5]

[1]Department of Statistics and Operations Research, The University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 27599, USA

[2]Department of Biostatistics, The University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 27599, USA

[3]Department of Genetics, The University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 27599-7264, USA

[4]Carolina Center for Genome Sciences, The University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 27599, USA

[5]Lineberger Comprehensive Cancer Center, The University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 27599, USA

Correspondence
Yufeng Liu, Department of Statistics & Operations Research, The University of North Carolina at Chapel Hill, 354 Hanes Hall, CB3260, Chapel Hill, NC 27599, USA.
Email: yfliu@email.unc.edu

Funding information
National Institute of General Medical Sciences, Grant/Award Number: 126550; National Science Foundation, Grant/Award Number: DMS2100729; National Institute on Aging, Grant/Award Number: AG073259

Data are generated at an unprecedented rate and scale these days across many disciplines. The field of streaming data analysis has emerged as a result of new data collection and storage technologies in various areas, such as air pollution monitoring, detection of traffic congestion, disease surveillance, and recommendation systems. In this paper, we consider the problem of model estimation for data streams in reproducing kernel Hilbert spaces. We propose an adaptive supervised learning method with a data sparsity constraint that uses limited storage spaces and can handle non-stationary models. We demonstrate the competitive performance of the proposed method using simulations and analysis of the bike sharing dataset.

KEYWORDS
algorithms, data stream, kernel regression, machine learning, reproducing kernel Hilbert space, sparsity, statistical learning

1 | INTRODUCTION

With the advance in technology, the volume of data generation is increasing at a very rapid rate. Due to the challenges of big data in many applications, streaming data analysis has attracted considerable attention. Supervised learning methods analyzing streaming data need to address several challenges, such as limited storage and concept drift. Specifically, the amount of memory required by the algorithms becomes infeasible as the number of samples in data streams increases (Langford et al., 2009). Moreover, sometimes the data stream exhibits a phenomenon referred to as concept drift (Gama et al., 2014), in which the underlying model evolves, causing the model constructed using old samples to become not applicable to new observations. Traditional machine learning algorithms may not be able to provide a good model as they may not adapt to the new changes.

The stochastic gradient descent (SGD) algorithm (Robbins & Monro, 1951), which can efficiently handle large-scale data sets, has gained increasing attention in developing supervised learning tools for data streams (Hazan et al., 2007; Littlestone, 1988; Rosenblatt, 1958). Given a convex loss function and a training set, researchers can use the SGD to obtain a sequence of models that converge to the optimal model. For many supervised learning problems, linear models can be suboptimal when the response has a nonlinear relationship with the predictors.

To improve the flexibility of the model, various nonlinear regression models (Hastie et al., 2009) can be used. Online learning with kernels (Kivinen et al., 2004) embeds the model in a reducing kernel Hilbert space (RKHS) (Aronszajn, 1950) to address the nonlinear relationship in the model. Because the regression function is assumed to be in an RKHS, it is common to take the squared norm of the regression function as the penalty. By the representer theorem (Kimeldorf & Wahba, 1971), the resulting regression function can be represented as a linear combination of kernel functions determined by the training data. In addition to the typical squared norm penalty, Zhang et al. (2016) introduced a data sparsity constraint. They showed that the regression model with the data sparsity constraint can have competitive prediction performance for various problems, especially when the sample size is small or moderate, or a sparse representation of the data can reasonably approximate the underlying function. Similar to the properties of support vector machine (SVM) (Boser et al., 1992; Cortes & Vapnik, 1995), the data points corresponding to kernel functions with nonzero coefficients are called support vectors (SVs).

The size of SVs grows linearly over time, posing storage and computational problems for these models. This may result in increasing storage space and training time. To resolve this issue, researchers have developed several different approaches. A family of algorithms, called "budget online kernel learning", has been proposed to bound the number of SVs with a fixed budget. Cavallanti et al. (2007) and Zhao et al. (2012) discarded one of the existing SVs uniformly during the training process. Dekel et al. (2008) discarded the oldest SVs during the training process. Orabona et al. (2008) used a new kernel function to approximate the removed SVs. These methods may suffer information loss when removing or approximating the SVs.

Another promising strategy is to explore the functional approximation techniques for achieving scalable kernel learning (Lu et al., 2016). The key idea is to construct a kernel-induced feature representation such that the inner product of instances in the new feature space can effectively approximate the kernel function. Because of the approximation, the model can suffer from high variation. As pointed out by Sun et al. (2018), the number of random features needed for consistent estimation grows when the number of SVs increases.

Many machine learning algorithms focus on a fixed model, where the relationship between the responses and the covariates does not change over time. However, learning a fixed function may not always be suitable for data streams. Many data streams are nonstationary. As a result, the underlying model may change over time. This problem is also known as concept drift, where the conditional distribution of the response given the predictors changes over time. Concept drift can affect the learner's performance if not handled properly. There are many algorithms in the literature for this issue. Schaul et al. (2013) introduced the vSGD for nonstationary models. In particular, in each step of the vSGD, the algorithm determines the learning rate adaptively to minimize the loss function by using a quadratic approximation of the objective function. One drawback of vSGD is that it may not be able to capture the model correctly when it changes rapidly.

Another common way to deal with concept drift is to detect changes and react accordingly. Concept drift can be detected by its effect on characteristic features of the model, such as the regression or classification accuracy. Such quantitative features can be accompanied by statistical tests to assess their significance. Such tests can rely on some well-known statistics, such as the Hoeffding bound (Frias-Blanco et al., 2014), or suitable distances such as the Hellinger distance (Ditzler & Polikar, 2011). These indirect methods rely on statistical power of the tests.

In this paper, we consider a supervised learning problem on data streams with the regression function in an RKHS. Our proposed method has several important features. First, by using random feature approximation, the proposed method does not need to store all the previous data and uses limited storage space and training time even when the total sample size is enormous. In addition, the variation of our model and the error induced by random feature approximation is reduced by using the data sparsity constraint and a shrinkage parameter. Finally, this method can also handle nonstationary models. In particular, at time t, our approach finds the best model in an RKHS by using the previously estimated model and kernel functions generated by the data we observe at time t. It updates the model by a shrinkage parameter and random feature approximation. Numerical studies in simulated and real data applications confirm that the proposed method performs competitively for data streams in both stationary and nonstationary problems.

The remainder of this paper is organized as follows. In Section 2, the problem background and the model are introduced. The simulated and real data examples are used to demonstrate the effectiveness of our proposed method in Sections 3 and 4, respectively.

2 | METHODOLOGY

2.1 | Problem setup and notation

We consider the supervised learning problem when the observations arrive sequentially. The goal is to recover the underlying mean function. At each time t, we are given a set of n_t instances $\{(x_i^t, y_i^t), i = 1, \ldots, n_t\}$ as our training set, where $t = 1, \ldots T, n_t$ is the number of data we receive at time t, T is the total number of times we observe, $\mathbf{x}_i^t \in \mathbb{R}^p$ is the p-dimensional covariate vector of the ith observation, and $y_i^t \in \mathbb{R}$ is the response of the ith observation. We consider fitting the model in an RKHS $\mathcal{H} = \{f | f : \mathbb{R}^p \rightarrow \mathbb{R}\}$ with a reproducing kernel function $K(\cdot,\cdot)$. The data at time t are observed according to the model

$$Y^t = f_t(\mathbf{X}^t) + \epsilon, \tag{1}$$

where $Y^t \in \mathbb{R}, \mathbf{X}^t \in \mathbb{R}^p, f_t \in \mathcal{H}$, and $\epsilon \in \mathbb{R}$ is the random noise. While traditional learning algorithms assume the data are sampled from a fixed model, here we assume that the model f_t may vary as a function of time t. Because we want to fit the model on data streams, with possibly an infinite number of observations, it is unrealistic to store all the data. Our goal is to fit our model with limited storage space.

2.2 | Proposed method

2.2.1 | Adaptive kernel learning on data streams

First, we describe the general adaptive kernel learning model on data streams. Given the training data $\{(x_i^t, y_i^t), i = 1, \ldots, n_t\}$ at time t, we consider the penalized regression problem that only uses these n_t samples

$$\tilde{f}_t(\mathbf{x}) = \arg\min_{f_t \in \mathcal{H}} \frac{1}{n_t} \sum_{i=1}^{n_t} L(f_t(\mathbf{x}_i^t), y_i^t) + \lambda J(f_t), \tag{2}$$

where L is a convex and differentiable loss function which measures the goodness of fit of f_t, J is a penalty function on f_t to avoid overfitting, and λ is a tuning parameter that controls the magnitude of penalty $J(f_t)$. By the representer theorem (Kimeldorf & Wahba, 1971), the estimated function in (2) can be written as

$$\tilde{f}_t(\mathbf{x}) = \sum_{i=1}^{n_t} \tilde{\alpha}_{t,i} K(\mathbf{x}_i^t, \mathbf{x}), \tag{3}$$

where $\tilde{\alpha}_{t,i}$ is the coefficients to be estimated, and we let $\tilde{\boldsymbol{\alpha}}_t = (\tilde{\alpha}_{t,1}, \ldots, \tilde{\alpha}_{t,n_t})^\top$. To learn our model in RKHS, it is common to use the regular squared norm penalty, which aims to solve the following optimization problem:

$$\tilde{f}_t(\mathbf{x}) = \arg\min_{f_t \in \mathcal{H}} \frac{1}{n_t} \sum_{i=1}^{n_t} L(f_t(\mathbf{x}_i^t), y_i^t) + \lambda \|f_t\|_{\mathcal{H}}^2, \tag{4}$$

where $\|f_t\|_{\mathcal{H}}$ is the norm of f_t in RKHS $\mathcal{H}$.

The kernel representation of the regression function is similar to the knot structure in the smoothing splines. Each observation in the training data can be regarded as a knot in a multidimensional space. For large sample size problems, the solution to (4) is known to be consistent with desirable theoretical properties. However, because the sample size n_t in each time is usually small in practice, using all kernel functions for the representation may introduce a similar issue as using too many knots in spline regression. For spline regression, it is known that too many knots may lead to overfitting and unnecessary fluctuation in the resulting estimator. To obtain the estimators with a sparse kernel function representation, Zhang et al. (2016) proposed the data sparsity penalty to constrain the estimated kernel function coefficient vector $\tilde{\boldsymbol{\alpha}}_t$ in an ℓ_1-ball. As shown in Zhang et al. (2016), the data sparsity model is desirable in this case because it can deliver estimators with a sparse kernel function representation. Hence, we follow their method and use the data sparsity penalty in our model as well. By (2) and (3), we aim to solve the following optimization problem with the data sparsity constraint

$$\hat{\boldsymbol{\alpha}}_t = \arg\min_{\boldsymbol{\alpha}_t} \left[\frac{1}{n_t} \sum_{i=1}^{n_t} L\left(\sum_{j=1}^{n_t} \alpha_{t,j} K(\mathbf{x}_j^t, \ \mathbf{x}_i^t), \ y_i^t \right) + \lambda \|\boldsymbol{\alpha}_t\|_1 \right], \tag{5}$$

where $\boldsymbol{\alpha}_t = (\alpha_{t,1}, \ldots, \alpha_{t,n_t})^\top$ and $\|\boldsymbol{\alpha}_t\|_1$ refers to the ℓ_1-norm of $\boldsymbol{\alpha}_t$. However, model (5) only uses the training data at time t without previous information. For $t > 1$, in order to use both the observations we receive at time t and the previous models we estimated before time t, we rewrite our estimated function as

$$\hat{f}_t(\mathbf{x}) = \hat{\gamma}_t \hat{f}_{t-1}(\mathbf{x}) + \sum_{j=1}^{n_t} \hat{\alpha}_{t,j} K\left(\mathbf{x}_j^t, \mathbf{x}\right), \quad \text{where } \gamma_t \in [0,1]. \tag{6}$$

Here, the adaptive weight $\hat{\gamma}_t$ illustrates how the model changes from time $t-1$ to time t. If the underlying true model f_t does not change, $\hat{\gamma}_t$ is expected to be 1 when $\hat{f}_{t-1}$ is a good estimator of f_{t-1}.

In summary, for the training dataset $\{(x_i^t, y_i^t), i = 1, \ldots, n_t\}$ and the model $\hat{f}_{t-1}(\boldsymbol{x})$ estimated at time $t-1$, our adaptive kernel learning model solves the following optimization at time t

$$(\hat{\gamma}_t, \hat{\boldsymbol{\alpha}}_t) = \arg\min_{\gamma_t, \boldsymbol{\alpha}_t} \left[\frac{1}{n_t} \sum_{i=1}^{n_t} L\left(\gamma_t \hat{f}_{t-1}(\boldsymbol{x}_i^t) + \sum_{j=1}^{n_t} \alpha_{t,j} K(\boldsymbol{x}_j^t, \ \boldsymbol{x}_i^t), \ y_i^t \right) + \lambda \|\boldsymbol{\alpha}_t\|_1 \right] \text{subject to} \gamma_t \in [0,1]. \tag{7}$$

2.2.2 | Adaptive kernel learning on data streams with adjusted learning rate

When the underlying true model f_t does not change from time $t-1$ to time t, model (7) uses $\sum_{j=1}^{n_t} \hat{\alpha}_{t,j} K(\boldsymbol{x}_j^t, \boldsymbol{x}_i^t)$ to fit the residual of our last model $y_i^t - \hat{f}_{t-1}(\boldsymbol{x})$ at time $t-1$. Hence, the bias of our model is reduced. However, $\hat{f}_t(\boldsymbol{x})$ is highly correlated to $\hat{f}_{t-1}(\boldsymbol{x})$ due to the sequential modeling process when $\hat{\gamma}_t = 1$. Compared to model (5), which only uses the data at time t, our model (7) has a smaller bias but a relatively larger variation. One advantage of the data sparsity constraint is that it can deliver estimators with a sparse kernel function representation. Hence, it is a much simpler model with a relatively small variation.

To balance the bias and variation, we introduce a shrinkage parameter ν. After we solve the optimization problem (7), if the solution $\hat{\gamma}_t = 1$, then the estimator $\hat{f}_t(\boldsymbol{x})$ is updated as

$$\hat{f}_t(\boldsymbol{x}) = \hat{f}_{t-1}(\boldsymbol{x}) + \nu \sum_{j=1}^{n_t} \hat{\alpha}_{t,j} K\left(\boldsymbol{x}_j^t, \boldsymbol{x}_i^t\right). \tag{8}$$

The shrinkage parameter $0 < \nu \le 1$ controls the learning rate of our model.

If $\hat{\gamma}_t \neq 1$, the underlying true model f_t may be changed from time $t-1$ to t and it is not necessary to use the shrinkage parameter. Then the estimator $\hat{f}_t(\boldsymbol{x})$ is still updated as

$$\hat{f}_t(\boldsymbol{x}) = \hat{\gamma}_t \hat{f}_{t-1}(\boldsymbol{x}) + \sum_{j=1}^{n_t} \hat{\alpha}_{t,j} K\left(\boldsymbol{x}_j^t, \boldsymbol{x}_i^t\right). \tag{9}$$

The learning rate parameter ν balances the learning speed and convergence rate trade-off of our model. With a large ν, our model can estimate f_t well with only a few batches of data but may converge to a suboptimal model. On the other hand, with a small ν, our model needs more batches of data to estimate f_t well but will converge to a model with better prediction.

Based on our numerical experience, we find that when the number of batches T is smaller than 50, or if we know that the model f_t is frequently changing, it is recommended to use a large ν such as 1. In contrast, when the number of batches T is larger than 1000 and the model f_t does not change, our model can eventually have a better prediction with a smaller ν, and it is recommended to use a small ν such as 0.3. In all other cases, in order to achieve the best performance, it is recommended to perform a grid search and choose the one that can minimize the testing error in predicting the responses.

2.2.3 | Adaptive kernel learning on data streams with limited storage space

In order to solve the optimization problem (7), we need to evaluate $K(\boldsymbol{x}_i^t, \boldsymbol{x}_j^{t'})$ for all the covariates $\boldsymbol{x}_j^{t'}$ we receive until time $t-1$ to calculate $\hat{f}_{t-1}(\boldsymbol{x}_i)$, where time $t' \le t-1$. Because the total sample size $n = \sum_{t=1}^{T} n_t$ can be very large, it is impossible for us to store all the data due to the limited storage. Here, we adopt random feature approximation (Lu et al., 2016) to store our model $\hat{f}_t(\boldsymbol{x})$ with limited storage for future use. The key idea is to construct a kernel-induced feature representation $z(\boldsymbol{x})$ such that the inner product of instances in the new feature space can effectively approximate the kernel function as

$$K\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \approx z(\boldsymbol{x}_i)^\top z(\boldsymbol{x}_j),$$

where $z(\boldsymbol{x}) \in \mathbb{R}^D$ is a function of $\boldsymbol{x}$, and D is the dimension of the function. A common random feature approximation technique, random Fourier features, can be used in shift-invariant kernels (Rahimi & Recht, 2007). A shift-invariant kernel is a family of reproducing kernel functions that can be written as $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\Delta \boldsymbol{x})$, where k is some function and $\Delta \boldsymbol{x} = \boldsymbol{x}_1 - \boldsymbol{x}_2$ is the difference between two instances. Examples of shift-invariant kernels include some widely used kernels, such as the Gaussian and Laplace kernels. By performing an inverse Fourier transform of the shift-invariant kernel function, one can obtain

$$K(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i - \mathbf{x}_j) = \int_{\mathbb{R}^p} p(\mathbf{u}) e^{i\mathbf{u}^\top (\mathbf{x}_i - \mathbf{x}_j)} d\mathbf{u},$$

where

$$p(\mathbf{u}) = \left(\frac{1}{2\pi}\right)^p \int_{\mathbb{R}^p} e^{-i\mathbf{u}^\top (\Delta \mathbf{x})} k(\Delta \mathbf{x}) d(\Delta \mathbf{x}),$$

which is a proper probability density function calculated from the Fourier transform of function $k(\Delta\mathbf{x})$. More specifically, for a Gaussian kernel $K(\mathbf{x}_i,\mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/(2\sigma^2))$, where σ is the width of the Gaussian kernel, we have the corresponding random Fourier component $\mathbf{u}$ with the distribution $p(\mathbf{u}) = \mathcal{N}(0, \sigma^{-2} I)$. Then for a continuous, positive-definite and shift-invariant kernel function, according to the Bochner theorem (Rudin, 1990), the kernel function can be expressed as

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= \int_{\mathbb{R}^p} p(\mathbf{u}) e^{i\mathbf{u}^\top (\mathbf{x}_i - \mathbf{x}_j)} d\mathbf{u} \\ &= \mathbb{E}_{\mathbf{u}}\left[\cos(\mathbf{u}^\top \mathbf{x}_i)\cos(\mathbf{u}^\top \mathbf{x}_j) + \sin(\mathbf{u}^\top \mathbf{x}_i)\sin(\mathbf{u}^\top \mathbf{x}_j)\right] \\ &= \mathbb{E}_{\mathbf{u}}\left[[\sin(\mathbf{u}^\top \mathbf{x}_i),\ \cos(\mathbf{u}^\top \mathbf{x}_i)] \cdot [\sin(\mathbf{u}^\top \mathbf{x}_j),\ \cos(\mathbf{u}^\top \mathbf{x}_j)]\right], \end{aligned}$$

where the operator $\cdot$ refers to the dot product between two vectors. Then any shift-invariant kernel function can be expressed by the expectation of the inner product between original data's new representation, where the new representation of the data is $z(\mathbf{x}) = [\sin(\mathbf{u}^\top \mathbf{x}), \cos(\mathbf{u}^\top \mathbf{x})]^\top$. We can sample $D \in \mathbb{N}$ number of random Fourier components $\mathbf{u}_1,\ldots\mathbf{u}_D$ independently for constructing the new representation as

$$z(\mathbf{x}) = \left(\sin(\mathbf{u}_1^\top \mathbf{x}),\ \cos(\mathbf{u}_1^\top \mathbf{x}),\ \ldots,\ \sin(\mathbf{u}_D^\top \mathbf{x}),\ \cos(\mathbf{u}_D^\top \mathbf{x})\right)^\top.$$

Rahimi and Recht (2007) used Hoeffding's inequality to show that the difference between $\mathbf{z}(\mathbf{x})'\mathbf{z}(\mathbf{y})$ and $k(\mathbf{x},\mathbf{y})$ decays exponentially fast in D. The kernel learning task in the original input space can be approximated by solving a linear learning task in the new feature space.

Using the above approximation, when $t = 1$, the model $\hat{f}_1(\mathbf{x}) = \sum_{i=1}^{n_1} \hat{\alpha}_{1,i} K(\mathbf{x}_i^1,\mathbf{x})$ can be rewritten as

$$\hat{f}_1(\mathbf{x}) = \sum_{i=1}^{n_1} \hat{\alpha}_{1,i} K(\mathbf{x}_i^1, \mathbf{x}) \approx \sum_{i=1}^{n_1} \hat{\alpha}_{1,i} z(\mathbf{x}_i^1)^\top z(\mathbf{x}) = \hat{\mathbf{w}}_1^\top z(\mathbf{x}),$$

where $\hat{\mathbf{w}}_1 = \sum_{i=1}^{n_1} \hat{\alpha}_{1,i} z(\mathbf{x}_i^1)$. Let $\mu_t = \nu$ when $\hat{\gamma}_t = 1$ and $\mu_t = 1$ when $\hat{\gamma}_t \neq 1$. Similarly, the model $\hat{f}_2(\mathbf{x}) = \hat{\gamma}_2 \hat{f}_1(\mathbf{x}) + \mu_2 \sum_{i=1}^{n_2} \hat{\alpha}_{2,i} K(\mathbf{x}_i^2,\mathbf{x})$ can be written as

$$\hat{f}_2(\mathbf{x}) \approx \hat{\gamma}_2 \hat{\mathbf{w}}_1^\top z(\mathbf{x}) + \mu_2 \sum_{i=1}^{n_2} \hat{\alpha}_{2,i} z(\mathbf{x}_i^2)^\top z(\mathbf{x}) = \hat{\mathbf{w}}_2^\top z(\mathbf{x}),$$

where $\hat{\mathbf{w}}_2 = \hat{\gamma}_2 \hat{\mathbf{w}}_1 + \mu_2 \sum_{i=1}^{n_2} \hat{\alpha}_{2,i} z(\mathbf{x}_i^2)$. By induction, when $t > 1$, for given model at time $t-1$ as $\hat{f}_{t-1}(\mathbf{x}) = \hat{\mathbf{w}}_{t-1}^\top z(\mathbf{x})$, our estimated function $\hat{f}_t(\mathbf{x}) = \hat{\gamma}_t \hat{f}_t(\mathbf{x}) + \mu_t \sum_{i=1}^{n_t} \hat{\alpha}_{t,i} K(\mathbf{x}_i^t,\mathbf{x})$ can be written as

$$\hat{f}_t(\mathbf{x}) \approx \hat{\gamma}_t \hat{\mathbf{w}}_{t-1} z(\mathbf{w}) + \mu_t \sum_{i=1}^{n_t} \hat{\alpha}_{t,i} z(\mathbf{x}_i^t)^\top z(\mathbf{x}) = \hat{\mathbf{w}}_t^\top z(\mathbf{x}), \tag{10}$$

where $\hat{\mathbf{w}}_t = \hat{\gamma}_t \hat{\mathbf{w}}_{t-1} + \mu_t \sum_{i=1}^{n_t} \hat{\alpha}_{t,i} z(\mathbf{x}_i^t)$. Then the optimization problem (7) can be written as

$$(\hat{\gamma}_t, \hat{\boldsymbol{\alpha}}_t) = \arg\min_{\gamma_t, \boldsymbol{\alpha}_t} \left[\frac{1}{n_t} \sum_{i=1}^{n_t} L\left(\gamma_t \hat{\mathbf{w}}_{t-1}^\top z(\mathbf{x}_i^t) + \sum_{j=1}^{n_t} \alpha_{t,j} z\left(\mathbf{x}_j^t\right)^\top z(\mathbf{x}_i^t),\ y_i^t\right) + \lambda \|\boldsymbol{\alpha}_t\|_1\right] \text{subject to} \gamma_t \in [0,1], \tag{11}$$

where $\hat{\mathbf{w}}_{t-1}$ is the coefficient vector of the previous model we estimated at time $t-1$. If $\hat{\gamma}_t = 1$, then the estimator $\hat{\mathbf{w}}_t$ is updated as

$$\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \nu \sum_{i=1}^{n_t} \hat{\alpha}_{t,i} z(\mathbf{x}_i^t).$$

If $\hat{\gamma}_t \neq 1$, the estimator $\hat{\mathbf{w}}_t$ is updated as

$$\hat{\mathbf{w}}_t = \hat{\gamma}_t \hat{\mathbf{w}}_{t-1} + \sum_{i=1}^{n_t} \hat{\alpha}_{t,i} z(\mathbf{x}_i^t).$$

Then instead of keeping all the data to evaluate kernel functions at each time, we need to keep a D-dimensional vector $\hat{\mathbf{w}}_t$.

The optimization problem (11) can be solved iteratively by the proximal gradient descent algorithm with a projection step in each iteration. More specifically, in the kth iteration, we first use the projected gradient descent method to update $\hat{\gamma}_{t,k}$ by

$$\hat{\gamma}_{t,k} = \begin{cases} 1 & z_1 \geq 1 \\ z_1 & z_1 \in (0,1) \\ 0 & z_1 \leq 0, \end{cases}$$

where

$$z_1 = \hat{\gamma}_{t,k-1} - \frac{\omega_t}{n_t} \sum_{i=1}^{n_t} \nabla_{\hat{\gamma}_{t,k-1}} L\left(\hat{\gamma}_{t,k-1} \hat{\mathbf{w}}_{t-1}^{\top} z(\mathbf{x}_i^t) + \sum_{j=1}^{n_t} \hat{\alpha}_{t,j,k-1} z\left(\mathbf{x}_j^t\right)^{\top} z(\mathbf{x}_i^t), y_i^t \right).$$

Then we use the proximal gradient descent method to update $\hat{\alpha}_{t,j,k}$ for $1 \leq j \leq n_t$ by

$$\hat{\alpha}_{t,j,k} = \text{sign}(z_2) \max(0, |z_2| - \omega_t \lambda),$$

where

$$z_2 = \hat{\alpha}_{t,j,k-1} - \frac{\omega_t}{n_t} \sum_{i=1}^{n_t} \nabla_{\hat{\alpha}_{t,j,k-1}} L\left(\hat{\gamma}_{t,k-1} \hat{\mathbf{w}}_{t-1}^{\top} z(\mathbf{x}_i^t) + \sum_{j=1}^{n_t} \hat{\alpha}_{t,j,k-1} z\left(\mathbf{x}_j^t\right)^{\top} z(\mathbf{x}_i^t), y_i^t \right).$$

Here, ω_t is the step-size, ∇ is the partial derivative operator, and

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0. \end{cases}$$

By using random feature approximation, we use $\sum_{i=1}^{n_t} \hat{\alpha}_{t,i} z\left(\mathbf{x}_i^t\right)^{\top} z(\mathbf{x})$ to approximate the function $\sum_{i=1}^{n_t} \hat{\alpha}_{t,i} K\left(\mathbf{x}_i^t, \mathbf{x}\right)$ in the model (10). This induces an approximation error $\|\sum_{i=1}^{n_t} \tilde{\alpha}_{t,i} z\left(\mathbf{x}_i^t\right)^{\top} z(\mathbf{x}) - \sum_{i=1}^{n_t} \tilde{\alpha}_{t,i} K\left(\mathbf{x}_i^t, \mathbf{x}\right)\|^2$. Data sparsity constraint can also reduce this approximation error. As pointed out by Sun et al. (2018), the dimension D of $z(x)$ needed for a consistent estimation grows when the number of nonzero coefficients in our model increases. Hence, when we use the random feature approximation with a fixed dimension D, the more nonzero coefficients $\tilde{\alpha}_{t,i}$ we use in the model (10), the larger approximation error it may generate. With the data sparsity constraint, there are fewer nonzero coefficients in our model. Hence our model has a lower approximation error compared to the model without data sparsity constraint.

Algorithm 1 below describes the major steps of the Incremental Adaptive Data Sparsity Kernel (IADSK) learning method for a given shift-invariant kernel function $K(\mathbf{x}_1, \mathbf{x}_2) = k(\Delta \mathbf{x})$, the number of random Fourier components D, the learning rate ν, and the loss function L.

Algorithm 1 Incremental Adaptive Data Sparsity Kernel learning (IADSK) method

```
Generate $D$ random Fourier components $\mathbf{u}_1, \ldots \mathbf{u}_D$ independently with the distribution
    $$p(\mathbf{u}) = \left(\frac{1}{2\pi}\right)^d \int e^{-i\mathbf{u}^\top(\Delta\mathbf{x})} k(\Delta\mathbf{x}) d(\Delta\mathbf{x}).$$
Constructing the random feature function as
    $$z(\mathbf{x}) = \left(\sin\left(\mathbf{u}_1^\top\mathbf{x}\right), \cos\left(\mathbf{u}_1^\top\mathbf{x}\right), \ldots, \sin\left(\mathbf{u}_D^\top\mathbf{x}\right), \cos\left(\mathbf{u}_D^\top\mathbf{x}\right)\right)^\top.$$
Let $\hat{\mathbf{w}}_0 = 0$.
**for** $t = 1$ to $T$ **do**
    Compute $\hat{\gamma}_t$ and $\hat{\boldsymbol{\alpha}}_t$ by
        $$(\hat{\gamma}_t, \hat{\boldsymbol{\alpha}}_t) = \arg\min_{\gamma_t, \boldsymbol{\alpha}_t} \left[\frac{1}{n_t}\sum_{i=1}^{n_t} L\left(\gamma_t \hat{\mathbf{w}}_{t-1}^\top z(\mathbf{x}_i^t) + \sum_{j=1}^{n_t} \alpha_{t,j} z\left(\mathbf{x}_j^t\right)^\top z(\mathbf{x}_i^t), y_i^t\right) + \lambda_t \|\boldsymbol{\alpha}_t\|_1\right] \quad \text{subject to } \gamma_t \in [0,1],$$
where tuning parameter $\lambda_t$ is chosen by cross validation.
    Compute $\hat{\mathbf{w}}_t$ by
        $$\hat{\mathbf{w}}_t = \begin{cases} \gamma_t \hat{\mathbf{w}}_{t-1} + \sum_{i=1}^{n_t} \hat{\alpha}_{t,i} z\left(\mathbf{x}_i^t\right) & \text{if } \gamma_t < 1; \\ \gamma_t \hat{\mathbf{w}}_{t-1} + \nu \sum_{i=1}^{n_t} \hat{\alpha}_{t,i} z\left(\mathbf{x}_i^t\right) & \text{if } \gamma_t = 1. \end{cases}$$
    Compute $\hat{f}_t$ by
        $$\hat{f}_t(\mathbf{x}) = \hat{\mathbf{w}}_t^\top z(\mathbf{x}).$$
**end for**
```

3 | NUMERICAL STUDY

In this section, we perform three numerical studies to compare the efficiency of our proposed method (IADSK) with different learning rates ν and two other methods. In particular, we choose $\nu = 1$,0.5 and 0.3 for IADSK. The other two methods include

- Fourier online gradient descent (FouGD) method (Lu et al., 2016), which is an online kernel learning method using random Fourier features for approximating kernel functions.
- Incremental Adaptive Ridge Kernel (IARK) learning method with different learning rates ν, which uses the squared norm penalty $\|f\|_{\mathcal{H}}^2$ instead of the data sparsity penalty, where $\|f\|_{\mathcal{H}}$ is the norm of f in RKHS $\mathcal{H}$. In particular, we also choose learning rate $\nu = 1$,0.5 and 0.3 for our proposed method.

In our numerical study, we use the Gaussian kernel and the ℓ_2-loss as our loss function in our training model (11). Then $L(\hat{f}(\mathbf{x}),y) = (y - \hat{f}(\mathbf{x}))^2$ and $K(\mathbf{x}_i,\mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/(2\sigma^2))$. Let the number of random features D be 30. For the first two examples, we aim to compare our method with other methods when the model is stationary. For the first example, we generate the data by

$$Y_t = 3\exp((X_t - 0.5)^2) + \exp((X_t - 1)^2) + \epsilon.$$

For the second example, we generate the data by

$$Y_t = 10\exp(X_t^2) + \epsilon.$$

Let $\epsilon \sim N(0,0.1)$ and $X_t \in \mathbb{R}$ follow a uniform distribution within $[-1,1]$. In both examples, we let the size of each batch of our training samples be 10, 20, or 40, and generate 3000 batches of training samples in total.

For the third example, we aim to compare our method with other methods when the model is non-stationary. We generate the data by

$$Y_t = 3\exp((X_t - 0.5)^2) + \exp((X_t - 1)^2) + \epsilon,$$

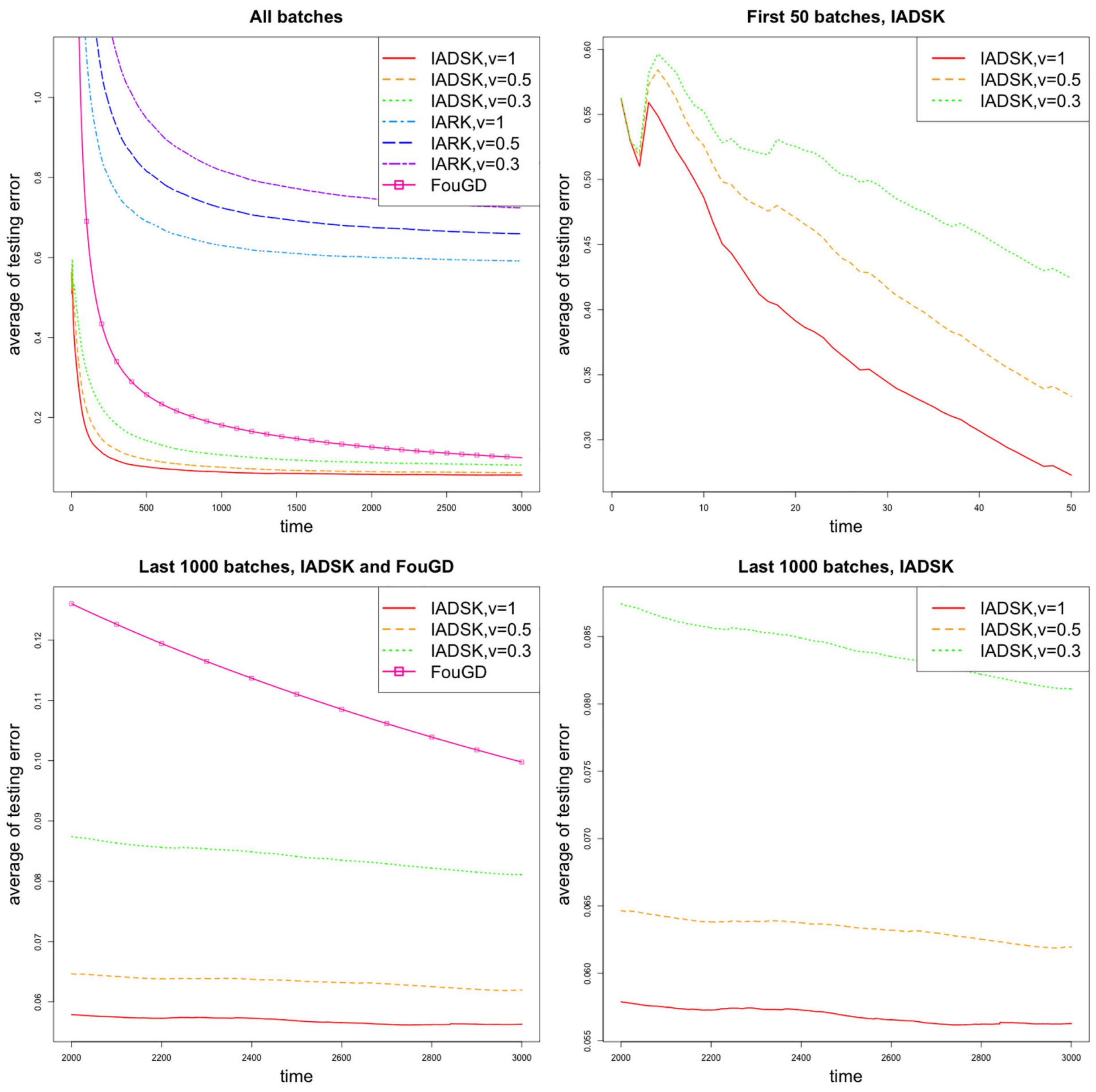

FIGURE 1 Performance comparison of different methods for Example 1 with 10 samples in each batch. The top left figure compares the performance of all methods for all 3000 batches of data. The top right figure compares the performance of IADSK with different learning rates for the first 50 batches of data. The bottom left figure compares the performance of FouGD and IADSK with different learning rates for the last 1000 batches of data. The bottom right figure compares the performance of IADSK with different learning rates for the last 1000 batches of data

when $t \in [1{,}500]$, and

$$Y_t = 3\exp((X_t + 0.5)^2) + \exp((X_t + 1)^2) + \epsilon,$$

when $t \in [501{,}1000]$. Let $\epsilon \sim N(0{,}0.1)$ and $X_t \in \mathbb{R}$ follow a uniform distribution within $[-1{,}1]$. We let the size of the each batch of our training samples be 20, and generate 1000 batches of training samples in total.

For each example, we repeat the simulation 50 times. Based on our numerical experience, $\sigma = 1$ appears to work well, so we set $\sigma = 1$ for all numerical examples. In practice, one can also tune σ using cross validation on the training data or using a separated tuning dataset. To evaluate the prediction performance of the algorithms at time t, we generate 100 testing samples $\{(X^t_{i,\text{text}}, Y^t_{i,\text{test}}), i = 1, \ldots, 100\}$. Then we use the average testing error from time 1 until time t as the criterion

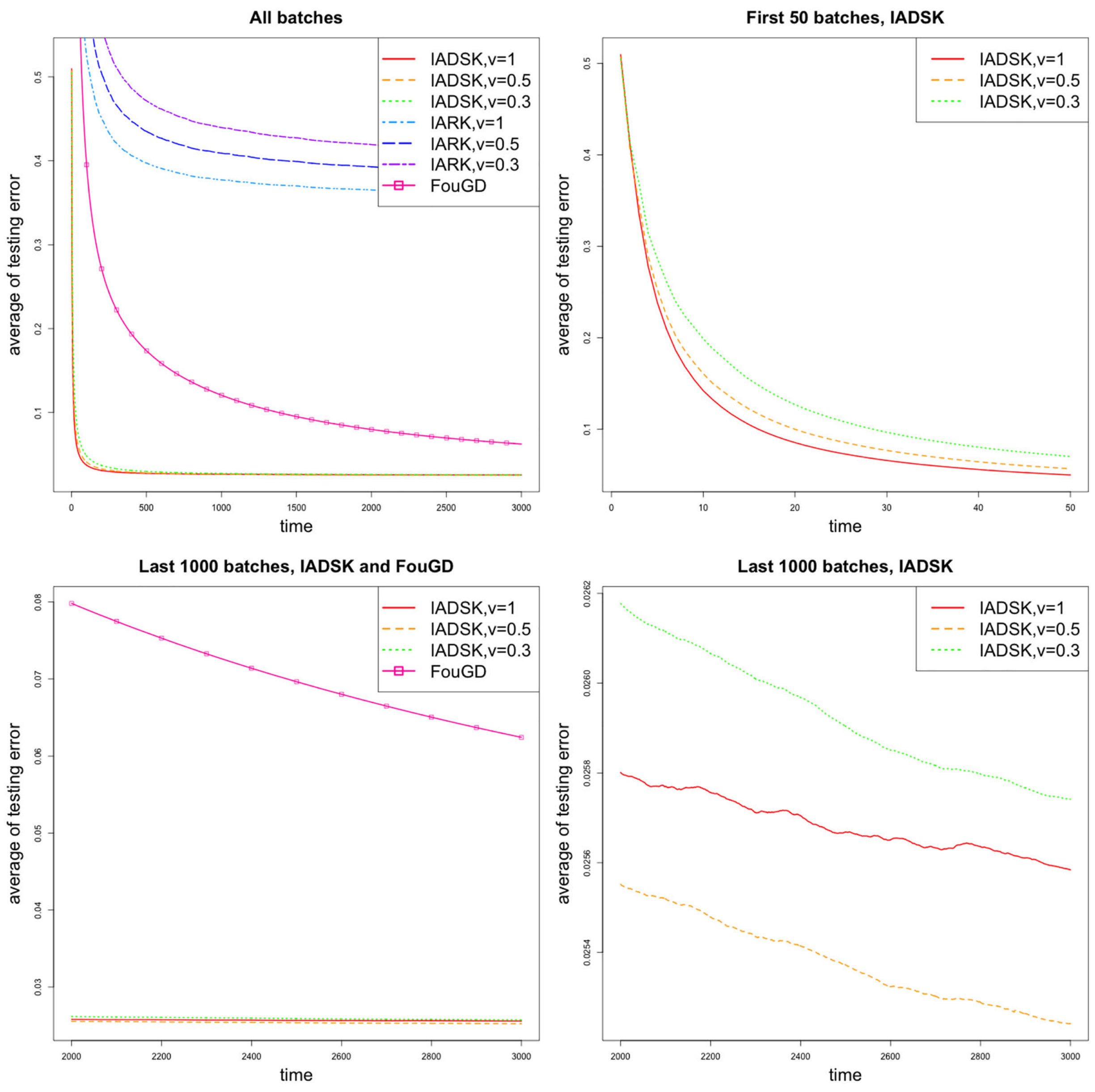

FIGURE 2 Performance comparison of different methods for Example 1 with 20 samples in each batch. The top left figure compares the performance of all methods for all 3000 batches of data. The top right figure compares the performance of IADSK with different learning rates for the first 50 batches of data. The bottom left figure compares the performance of FouGD and IADSK with different learning rates for the last 1000 batches of data. The bottom right figure compares the performance of IADSK with different learning rates for the last 1000 batches of data

$$\frac{1}{t}\sum_{i=1}^{t}\frac{1}{100}\sum_{j=1}^{100}\left(Y_{j,\text{test}}^{i}-\hat{Y}_{j,\text{test}}^{i}\right)^{2},$$

where $\hat{Y}_{j,\text{test}}^{t}=\hat{f}_{t}(X_{j,\text{text}}^{t})$ is the prediction using our estimated model $\hat{f}_t$ at time t. In addition, after we plot the performance of all methods, we zoom in some parts of the plot to highlight the comparison of different methods. In particular, for the first and second examples, we first plot the performance of all methods for all batches. Second, we plot the performance of IADSK with three different learning rates for the first 50 batches. Then we plot the performance of FouGD and IADSK with three different learning rates for the last 1000 batches. Finally we plot the performance of IADSK with three different learning rates for the last 1000 batches.

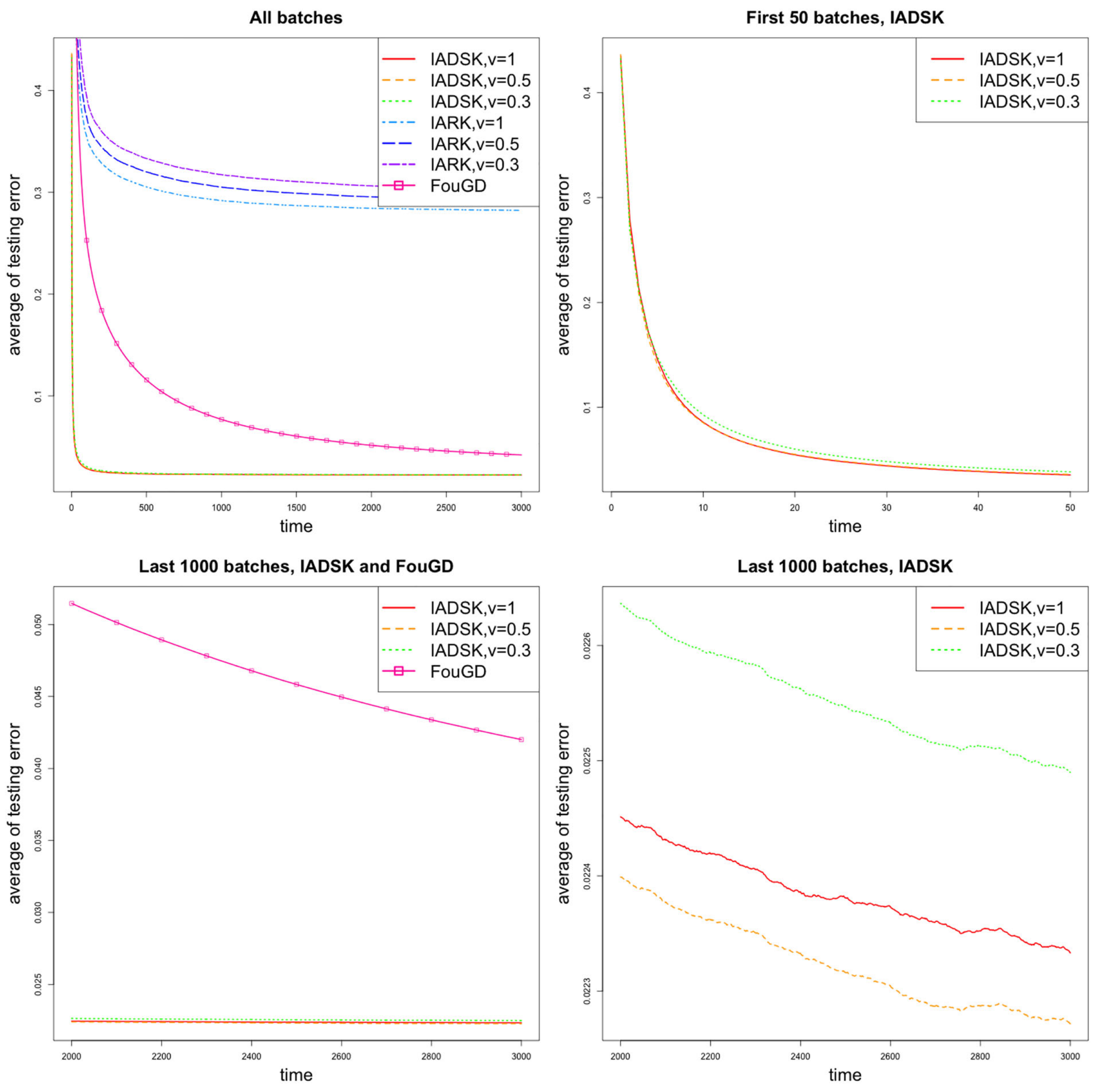

FIGURE 3 Performance comparison of different methods for Example 1 with 40 samples in each batch. The top left figure compares the performance of all methods for all 3000 batches of data. The top right figure compares the performance of IADSK with different learning rates for the first 50 batches of data. The bottom left figure compares the performance of FouGD and IADSK with different learning rates for the last 1000 batches of data. The bottom right figure compares the performance of IADSK with different learning rates for the last 1000 batches of data

For the third example, we first plot the performance of IADSK with three different learning rates, IARK with three different learning rates, and FouGD for all batches. Then we plot the performance of FouGD and IADSK with 3 different learning rates for time $t \in [400,600]$. Finally, we plot the performance of IADSK with three different learning rates for time $t \in [400,600]$.

We report the simulation results in Figures 1–7. Figures 1–3 show the results of Example 1 with 10, 20, and 40 training samples in each batch, respectively. Figures 4–6 show the results of Example 2 with 10, 20, and 40 training samples in each batch, respectively. Figure 7 shows the result of Example 3.

Compared with the other two methods, our proposed IADSK method always delivers better prediction than FouGD and IARK. Specifically, the average testing error of IADSK with $\nu = 1$ decreases much faster than the other methods when $t \le 50$, and IADSK with $\nu = 1$ performs better when t is small.

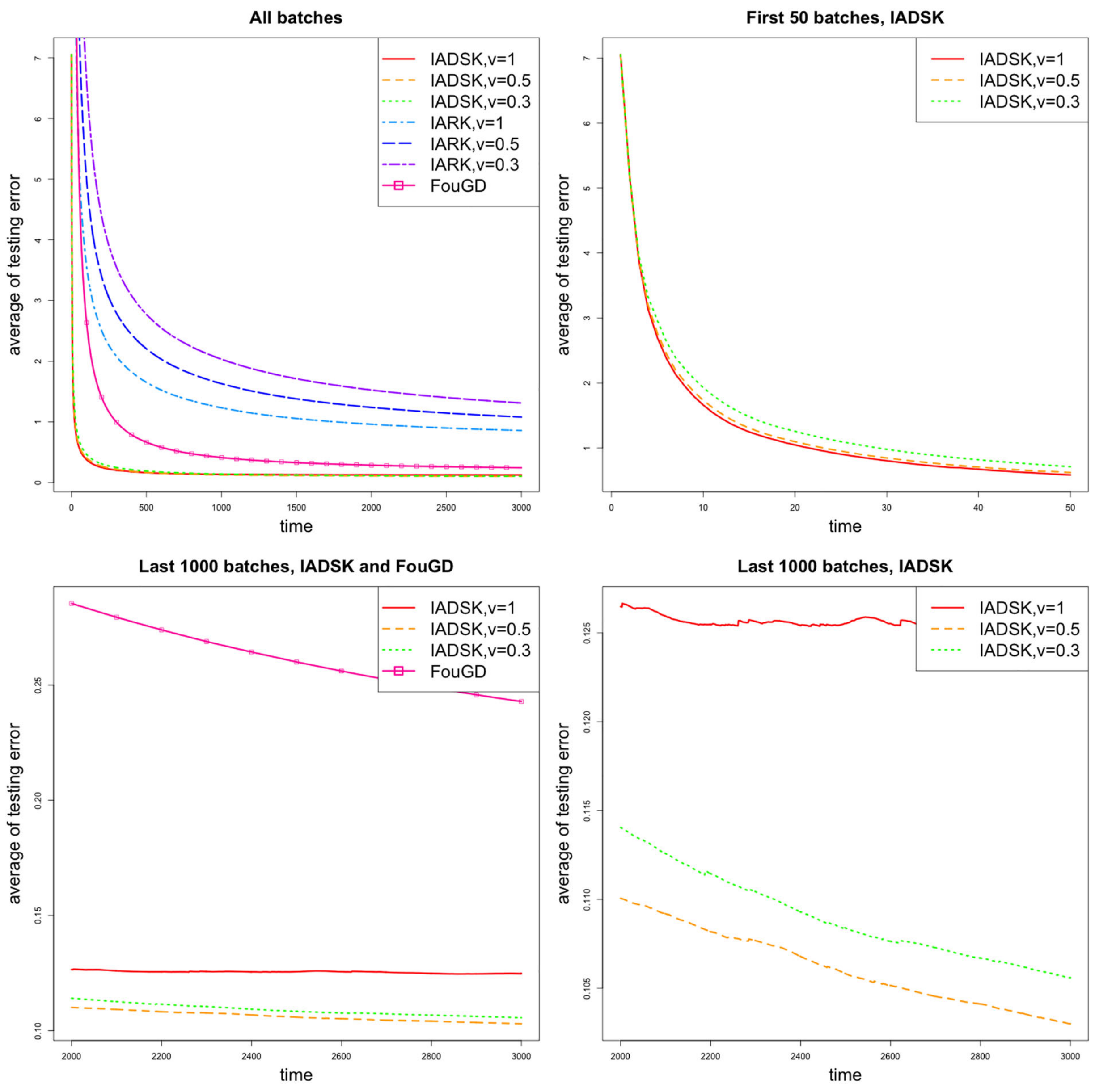

FIGURE 4 Performance comparison of different methods for Example 2 with 10 samples in each batch. The top left figure compares the performance of all methods for all 3000 batches of data. The top right figure compares the performance of IADSK with different learning rates for the first 50 batches of data. The bottom left figure compares the performance of FouGD and IADSK with different learning rates for the last 1000 batches of data. The bottom right figure compares the performance of IADSK with different learning rates for the last 1000 batches of data

For the first example when $n_t = 10$, Figure 1 shows that the IADSK with $\nu = 1$ always produces smaller average testing errors than the other methods. But as time t becomes larger, the testing error of FouGD decreases faster than that of IADSK and IARK.

For the first example when $n_t = 20$ or 40 and the second example when $n_t = 10,20$ or 40, Figures 2–6 show that although the average testing error of IADSK with $\nu = 1$ is smaller than all the other methods when $t \le 50$, when $t > 2000$, the testing error of IADSK with $\nu = 0.3$ or 0.5 becomes smaller than IADSK with $\nu = 1$.

For the third example, the lower right plot in Figure 7 shows that the model change at time $t = 500$ has more impact on the performance of FouGD than our proposed method. In addition, the lower-left plot shows that the average testing errors of IADSK with $\nu = 1$ decrease faster than IADSK with $\nu = 0.5$ or 0.3 after the model changes.

We also compare the computational cost of different methods. For the computational speed, we report the total training time of IADSK, IARK and FouGD for Example 1 with 20 training samples in each batch in Table 1. Because FouGD processes the data points one by one, it deals fewer data points than the two incremental learning methods IADSK and IARK. However, IADSK and IARK can still handle large samples relatively

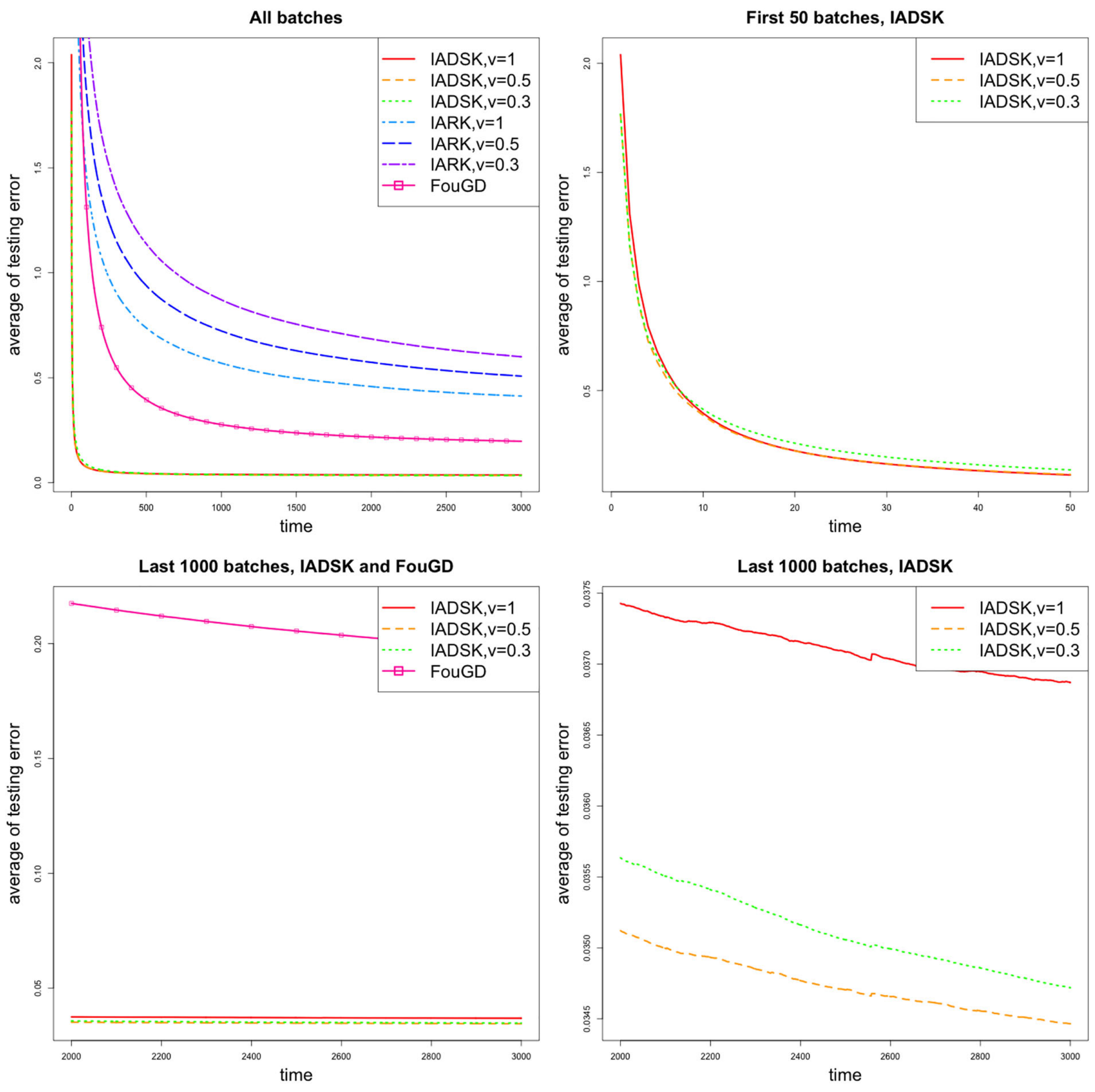

FIGURE 5 Performance comparison of different methods for Example 2 with 20 samples in each batch. The top left figure compares the performance of all methods for all 3000 batches of data. The top right figure compares the performance of IADSK with different learning rates for the first 50 batches of data. The bottom left figure compares the performance of FouGD and IADSK with different learning rates for the last 1000 batches of data. The bottom right figure compares the performance of IADSK with different learning rates for the last 1000 batches of data

efficiently. In terms of the storage complexity, all methods need to store D p-dimensional random Fourier components, and D-dimensional $\hat{\mathbf{w}}_t$ and γ_t. IADSK and IARK need to store the kernel matrix for each batch of training data so their storage complexity is $O(n_t^2+Dp)$, while the storage complexity of FouGD is $O(Dp)$.

4 | EXPERIMENTS ON REAL DATA

We demonstrate the performance of our proposed model using the bike sharing dataset from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset). The dataset is the hourly usage of a bike sharing system within 2 years in Washington, D.C., USA. In this section, we will use different methods to predict the hourly bike rental counts by some seasonal and environmental factors such

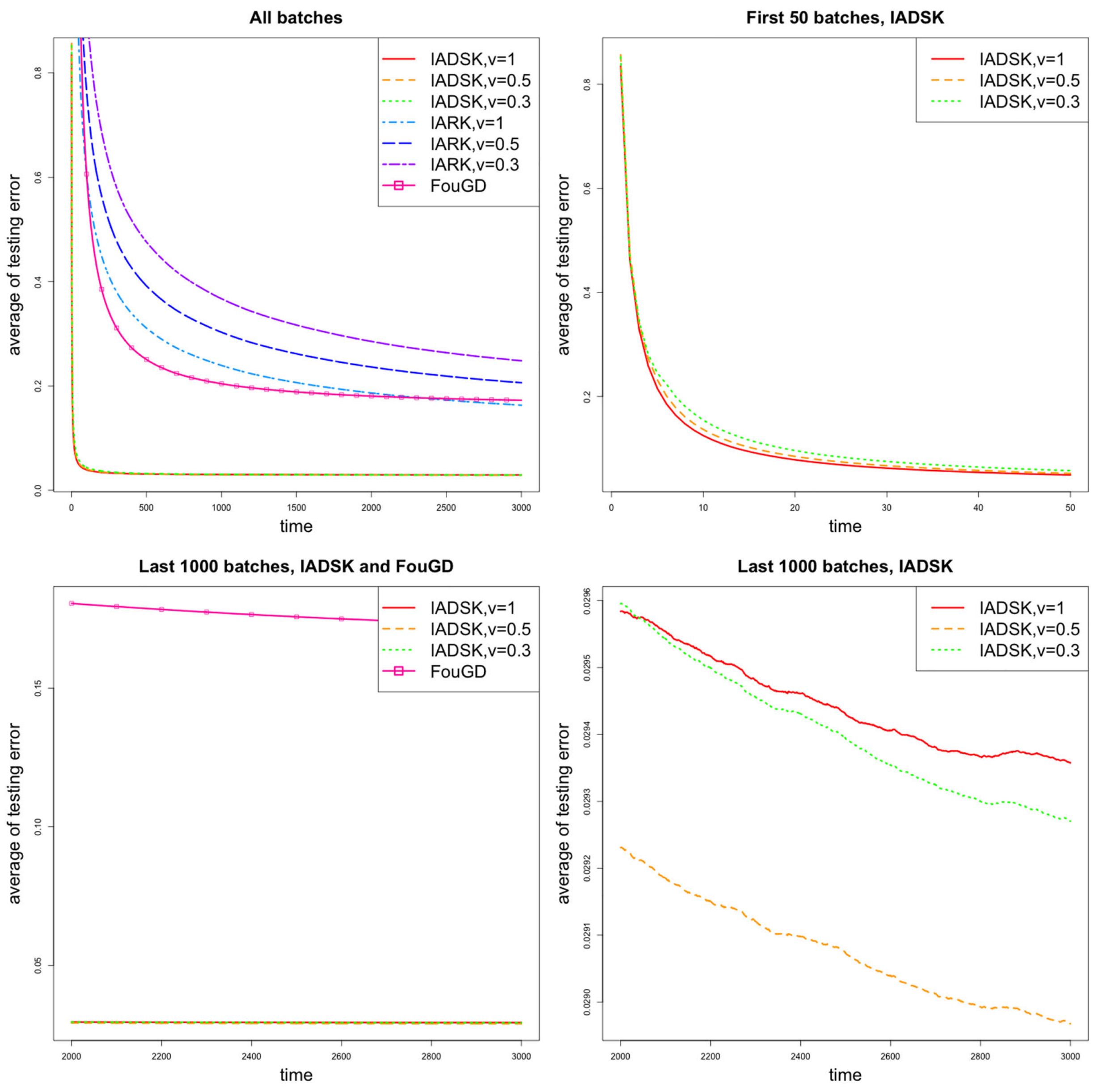

FIGURE 6 Performance comparison of different methods for Example 2 with 40 samples in each batch. The top left figure compares the performance of all methods for all 3000 batches of data. The top right figure compares the performance of IADSK with different learning rates for the first 50 batches of data. The bottom left figure compares the performance of FouGD and IADSK with different learning rates for the last 1000 batches of data. The bottom right figure compares the performance of IADSK with different learning rates for the last 1000 batches of data

as holiday, temperature, wind speed, and humidity. There are two main reasons why we think this dataset is suitable for our proposed method. First, the data are collected by the data stream procedure. Second, there are more than 17,000 samples in the dataset, which can be large for direct analysis using the traditional supervised learning methods.

In our experiments, we divide the data into the training and testing sets. The testing set consists of 196 randomly selected subjects, and the training set consists of T batches of samples, and each batch contains n_t subjects, where T and n_t are specified in each experiment. The analyses were repeated 30 times for each method using different data partitions. We use the Gaussian kernel and the ℓ_2-loss in our training model (11). To evaluate the result, we use the average testing error from time 1 until time t as the criterion

$$\frac{1}{t}\sum_{i=1}^{t}\frac{1}{100}\sum_{j=1}^{100}\left(Y_{j,\text{test}}^{i}-\hat{Y}_{j,\text{test}}^{i}\right)^2.$$

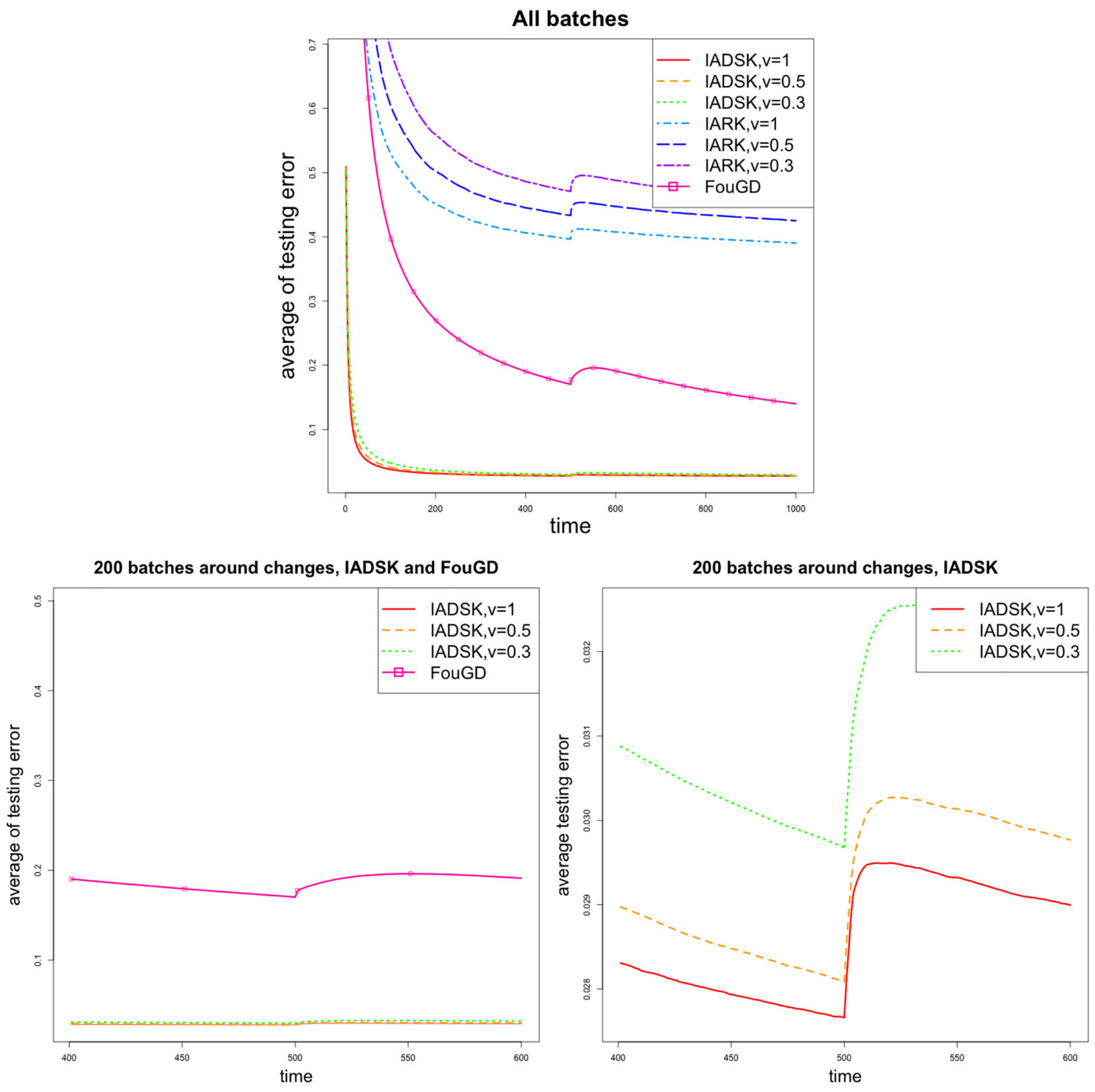

FIGURE 7 Performance comparison of different methods for Example 3 with 20 samples in each batch. The top figure compares the performance of all methods for all 3000 batches of data. The bottom left figure compares the performance of FouGD and IADSK with different learning rates for time $t \in [400,600]$. The bottom right figure compares the performance of IADSK with different learning rates for time $t \in [400,600]$p>

TABLE 1 Total training time comparison of different methods for Example 1 with 20 samples in each batch

Methods	IADSK	IARK	FouGD
Training time	198 s	186 s	12 s

In the first experiment, we let $n_t = 24$ and $T = 416$. In the second experiment, we let $n_t = 72$ and $T = 222$. The results are plotted in Figure 8. Both results indicate that our proposed IADSK method delivers the best prediction among all methods. Although the average testing errors of IADSK with $\nu = 1$ are smaller than all the other methods when $t \leq 50$, when $t > 150$, the testing errors of IADSK with $\nu = 0.5$ become smaller than those of IADSK with $\nu = 1$.

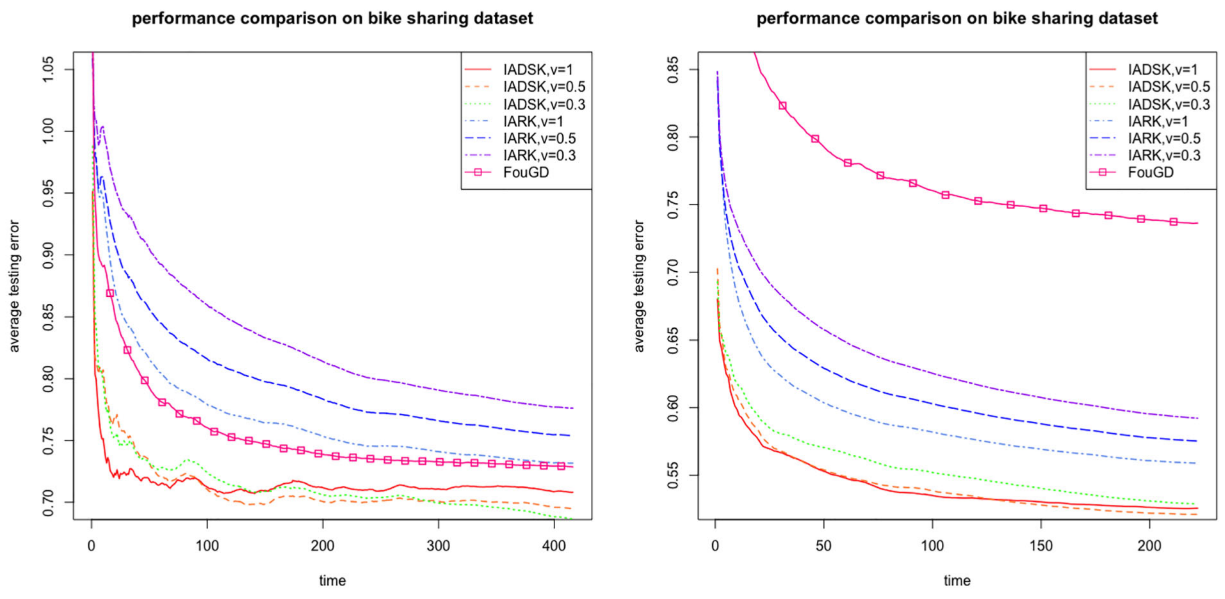

FIGURE 8 Performance comparison of different methods for the bike sharing dataset. The left figure compares the performance of all methods for the bike sharing dataset with 24 samples in each batch. The right figure compares the performance of all methods for the bike sharing dataset with 72 samples in each batch

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in UCI Machine Learning Data Repository at https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset.

ORCID

Yufeng Liu https://orcid.org/0000-0002-1686-0545

REFERENCES

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, *68*(3), 337–404.
Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152.
Cavallanti, G., Cesa-Bianchi, N., & Gentile, C. (2007). Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, *69*(2), 143–167.
Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.
Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2008). The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, *37*(5), 1342–1372.
Ditzler, G., & Polikar, R. (2011). Hellinger distance based drift detection for nonstationary environments. In *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*, pp. 41–48.
Frias-Blanco, I., del Campo-Ávila, J., Ramos-Jimenez, G., Morales-Bueno, R., Ortiz-Diaz, A., & Caballero-Mota, Y. (2014). Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, *27*(3), 810–823.
Gama, J., Žliobaitundefined, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computating Surveys*, *46*(4), 1–37.
Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, Vol. 2. Springer.
Hazan, E., Rakhlin, A., & Bartlett, P. (2007). Adaptive online gradient descent. *Advances in Neural Information Processing Systems*, *20*.
Kimeldorf, G., & Wahba, G. (1971). Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, *33*(1), 82–95.
Kivinen, J., Smola, A. J., & Williamson, R. C. (2004). Online learning with kernels. *IEEE Transactions on Signal Processing*, *52*(8), 2165–2176.
Langford, J., Li, L., & Zhang, T. (2009). Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, *10*(3), 777–801.
Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*(4), 285–318.
Lu, J., Hoi, S. C. H., Wang, J., Zhao, P., & Liu, Z.-Y. (2016). Large scale online kernel learning. *Journal of Machine Learning Research*, *17*(47), 1.
Orabona, F., Keshet, J., & Caputo, B. (2008). The projectron: A bounded kernel-based perceptron. In *Proceedings of the 25th international conference on machine learning*, pp. 720–727.
Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, *20*.
Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, *1951*, 400–407.
Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386.
Rudin, W. (1990). *Fourier Analysis on Groups* (1st ed.).: Wiley.
Schaul, T., Zhang, S., & LeCun, Y. (2013). No more pesky learning rates. In S. Dasgupta, & D. McAllester, (Eds.), Proceedings of the 30th International Conference on Machine Learning. Proceedings of Machine Learning Research (Vol. *28*, pp. 343–351). PMLR, Atlanta, Georgia, USA.

Sun, Y., Gilbert, A., & Tewari, A. (2018). But how does it work in theory? Linear SVM with random features. *Advances in Neural Information Processing Systems*, *31*, 48109.

Zhang, C., Liu, Y., & Wu, Y. (2016). On quantile regression in reproducing kernel hilbert spaces with the data sparsity constraint. *The Journal of Machine Learning Research*, *17*(1), 1374–1418.

Zhao, P., Wang, J., Wu, P., Jin, R., & Hoi, S. C. H. (2012). Fast bounded online gradient descent algorithms for scalable kernel-based online learning. arXiv preprint arXiv:1206.4633.

How to cite this article: Wang, H., Li, Q., & Liu, Y. (2023). Adaptive supervised learning on data streams in reproducing kernel Hilbert spaces with data sparsity constraint. *Stat*, *12*(1), e514. https://doi.org/10.1002/sta4.514