Real Time Kernel Learning for Sensor Networks using Principles of Federated Learning

Anthony Kuh
University of Hawaii
E-mail: kuh@hawaii.edu

***Abstract*—This paper considers online kernel learning at the edge. Increasingly, we are seeing more networks consisting of edge processors and a central processor or coordinator. The edge processor could be a sensor, mobile phone, cognitive assistant, and/ or IoT devices. These processors likely have limited power and communication capabilities, but have enough processing ability to make intelligent decisions and learn. With the advent of federated learning, these edge processors do not pass data to the central processor, but assist in learning by adjusting parameters of the learning model based on data the edge processor receives before passing information about the updated model to the central processor. Here we consider where the edge processors use kernel methods to perform the learning. The learning algorithms are online with edge processors adjusting the model parameters using stochastic gradient algorithms. We establish a framework for online federated kernel learning.**

***Index Terms*—federated learning, online kernel methods**

I. Introduction

This paper discusses distributed learning combining principles of federated learning, kernel methods, and adaptive signal processing, . Here we will assume that we have a number of edge processors (sensors) that receive data and a central processor that coordinates activities of the local processors. We can assume that the local processors are sensors that gather information. With advances in electronic devices, integrated circuits, and GPUs, sensors will be tasked to do more than take measurements and gathering data. They can be made more autonomous and intelligent, making local decisions and also assisting in overall learning and decision making. The paper establishes a framework for online federated kernel learning.

In a classical setup, the local sensors would send received data to the central processor for learning and decision making. Communication costs can be quite high if the local sensors send data to the central processor. Federated learning reduces communication costs by having local sensors processing and learning information about their local data eliminating the need for the data to be sent to the central processor [1], [2], [3]. If the local sensor has knowledge of the learning model (weights / parameters of the model) as data is received by the local sensor, the sensor adjusts the weights of the learning model and transmits these to the central processor. The central processor receives the changes in the model and transmit this information to another sensor that has received data and again makes updates to the weights of the model and then transmits the model changes to the central processor. Federated learning also has many other attributes including privacy issues as local sensors do not send data to the central processor and federated learning easily deals with data that is heterogeneous.

Here we will examine real-time learning and decision making where data is received by the sensors and updates are performed online. Specifically, we consider online adaptive learning with kernel functions based on Reproducing Kernel Hilbert Spaces (RKHS), [4]. Kernel methods are popular learning methods where both supervised and unsupervised learning can be implemented by learning a hyperplane in feature space. The optimization problem is posed as a convex optimization problem. Nonlinear methods are learned using kernel functions and by solving problems in the dual observation space where the function to be learned is represented by either a linear sum of kernels evaluated from a dictionary of support vectors or random features that approximate the kernel function. An L_2 (squared error) cost function can be used (turns problem into variations of least squares problems) along with adaptive signal processing methods to develop nonlinear online distributed adaptive filters. Principles of graph signal processing can be applied to account for relationships and correlations between different sensors.

This paper discusses a framework for online federated learning using kernels. In Section 2 we discuss the online federated learning model. Section 3 discusses more specific algorithms and some of the tradeoffs between performance, communication costs, and algorithm complexity. Section 4 presents some numerical simulation results to compare performance of different algorithms and Section 5 summarizes the paper and discusses future research directions.

II. Online Federated Learning Models

Here we learn a function $\hat{y}(x) = \alpha^T \xi(x) + \alpha_0$ where $x \in \mathcal{R}^n$ is an input vector, $\xi(x) \in \mathcal{R}^D$ is a basis vector, α is a vector of weights, and α_0 is a threshold value. Here the task to be learned could be a supervised learning task (e.g. regression, prediction, classification) or could be an unsupervised learning task (e.g. principal component analysis, probability density approximation). For classical kernel methods, $\xi(x) = k(x, \mathcal{B})$ where k represents a kernel function based on Reproducing Kernel Hilbert Spaces (RKHS), [4]. $\mathcal{B}$ represents the set of D vectors forming the dictionary that represent the support vectors.

We also consider where the kernel functions are approximated by random features drawn from a probability distribution , [5]. Examples include using Random Fourier

Features (RFF) where $x, z \in \mathcal{R}^n$ and the kernel function $k(x,z) = k(x-z)$ can be written as a difference of terms and then expressed in terms of an inverse D-dimensional Fourier Transform

$$k(x-z) = \int p(v) \exp(jv^T(x-z))dv$$

where $j^2 = -1$ and $p(v)$ is a probability density function (pdf). We can then approximate the kernel function using sinusoids

$$\xi(x) = (D/2)^{-\frac{1}{2}}[\cos(v_1^T x + b_1), \ldots, \cos(v_D^T x + b_D)]^T. \quad (1)$$

where $v_i, 1 \leq i \leq D$ are drawn at random from $p(v)$ and the phase $b_i, 1 \leq i \leq D$ is uniform from 0 to 2π.

The learning algorithms will be online with N edge processors receiving and processing data with a central processor that assists in coordinating learning. A key factor is reducing communications between different processors as the edge processors could have bandwidth constraints and they could be power constrained (e.g. self-powered wireless sensors with limited power with communication consuming much of the power). Federated learning is ideal in these situations where the edge devices process and learn data, [1], [2]. Then there is no need for the central processor to receive input data. It coordinates learning by sending information about the learning model to the edge processor. The edge processor, when ready updates this information and sends the updated model information to the central processor and possibly other edge processors.

1) Learning Framework: Learning algorithms are based on combining kernel methods and adaptive signal processing. Here we initially assume there is one task to learn where the central processor coordinates the overall learning strategy with edge processing assisting in learning. The basic model can be described by a Least Squares Support Vector Machine, [6]. Here we let $x_i(k)$ be the value of node i at time k, $X(k) = [x_1(k), \ldots, x_N(k)]$ be the matrix of all node values at time k and $\mathbf{X}(\mathbf{k})$ be the tensor describing all node values from time up to time k. Below are some examples of cost functions for a few supervised and unsupervised learning problems.

Learning Task	Cost Function, J()
Regression	$\frac{1}{2m}\|\|Y(m) - \hat{Y}(m)\|\|^2 + \mathcal{R}(\alpha, \mathbf{X}(\mathbf{m}))$
Classification	$\frac{1}{2m}\|\|\mathbf{1} - Y(m) \cdot \hat{Y}(m)\|\|^2 + \mathcal{R}(\alpha, \mathbf{X}(\mathbf{m}))$
1st Principal Component	$\frac{1}{2m}\|\|\hat{Y}(m)\|\|^2 + \mathcal{R}(\alpha, \mathbf{X}(\mathbf{m}))$
Prob. Density Estimation	$\frac{1}{2m}\|\|\mathbf{1}\rho - \hat{Y}(m)\|\|^2 + \mathcal{R}(\alpha, \mathbf{X}(\mathbf{m}))$

Table 1: Supervised and unsupervised learning tasks

For supervised learning $Y(m) = [y(1), \ldots, y(m)]^T$ is a vector of target values for times up to m. For all learning tasks $\hat{Y}(m)$ is the associated learned estimate of $Y(m)$, where $\hat{Y}(m) = [\hat{y}(1), \ldots, \hat{y}(m)]^T$. Here $\hat{y}(k),\ 1 \leq k \leq m$ represents samples drawn from a hyperplane in feature space that approximates $y(k)$ with

$$\hat{y}(k) = \alpha^T \xi(z(k)) + \alpha_0,\ z(k) \subset \mathbf{X}(\mathbf{k})$$

For simple cases $z(k) = x_i(k)$, a sample drawn from one sensor or $z(k) = X(k)$, a sample drawn from all sensors. For more complex cases $z(k)$ will depend on both temporal and spatial samples. For learning problems considered here we assume that the dimension of $z(k)$ is constant. For all tasks $\mathcal{R}$ represents the regularization function for the learning problem. Here $\mathbf{1}$ is a vector of 1s and $\cdot$ is a component by component multiplication of two vectors.

The optimization is based on [6] and is similar to Gaussian processes, [7]. For binary classification we have two parallel hyperplanes. Samples from the positive hyperplane approximate positive points and samples from the negative hyperplane approximate negative points. For Principal Component Analysis (PCA) we obtain the first PCA. Here we assume data is appropriately normalized in the feature space to be zero mean where the goal is to maximize the sample variance of the data from the hyperplane in feature space that projects the data. For probability density estimation the hyperplane in feature space approximate points drawn from a probability density function.

The regularization term typically depends on a term regularizing a quadratic function of the weights α such as $\frac{1}{2}\gamma_1 \alpha^T \mathbf{K} \alpha$ where $\mathbf{K}$ is a matrix of support vector kernels or inner products of RFF and γ_1 is the regularization constant for this cost function. There could be another term describing relationships between the nodes spatially (and possibly temporally) which can use graph signal processing [8]. Connections with a graph of N nodes can be described by the adjacency matrix A which give relationships between different nodes and the Laplacian matrix defined by $L = D - A$ where D is a diagonal matrix where the diagonals contain the sum of the rows of A. Let α^* be a vector associated with the nodes (edge processors). Then an additional regularization term could be $\frac{1}{2}\gamma_2 \alpha^{*T} L \alpha^*$ where γ_2 is the regularization constant for this cost function.

Batch solutions to these learning algorithms involve choosing the basis functions $\xi(z(k))$ and either use support vectors for the dictionary or RFF. Once this is done, the solution to the learning algorithms involve solving a set of linear equations for α and α_0. Here we are interested in online solutions where edge processors assist with the learning.

Note that the online algorithm will operate in the dual observation space. If we use kernels as the basis function to learn, the size of the learning problem grows as the number of observation data. We can limit the size of the support vector machines by adding support vectors if they satisfy a certain criterion that the support vectors sufficiently span the feature space such that linear combinations of support vectors approximate other input data well. The support vectors are added in an online manner according to criterion used. Two popular criterion are the approximate linear dependence criterion (ALD), [9] and the coherence criterion (CC), [10]. It is shown that the number of support vectors in the dictionary for both criterion converge to a finite value. For RFF we can pre-specify the basis functions before we start the algorithm. In recent work, [11], [12] consider using RFF in developing online kernel algorithms for graph signal processing, however they do not consider communications involved with distributed

learning scenarios. Here we develop a general federated learning framework.

General federated learning with kernels

1) Choose model and basis functions to use (kernels).
2) Central processor initializes model (could be choosing basis functions (e.g. support vectors or RFF), once basis functions chosen, then choose weights, α for these basis functions.
3) Central processor chooses edge processor that has new data. This can be done synchronously or asynchronously. Central processor sends update of model to edge processor (e.g. updated support vectors and weights).
4) Edge processor updates model weights via an adaptive filter algorithm in dual space (e.g. stochastic gradient algorithm such as Kernel Least Mean Square (KLMS) algorithm). Edge processor determines whether to update dictionary by adding basis vectors (e.g. data it has received or RFF)
5) Edge processor sends updated model to central processor and possibly neighboring processors (if using a diffusion model).
6) Central processor examines number of iterations and estimate of cost function. If certain criterion not met go to 3).

In the next section we discuss learning algorithms depending on learning task and examine convergence, optimization performance, computational complexity, and communication bandwidth. This also includes varying number of basis vectors in dictionary, tuning hyper-parameters, examining robustness of algorithms due to impairments such as additive noise, examining different classes of algorithms ranging from stochastic gradient type algorithms to more complicated projection based algorithms [13]. We will see there are obvious tradeoffs depending on amount of communication bandwidth needed and desired performance (convergence speed, mean squared error).

III. Algorithm Discussion and Tradeoffs

The central processor is tasked to learn one task by collecting information from edge processors. Here we assume that data comes to each sensor. Let m cumulative data be received by all sensors and let $\hat{y}(m+1)$ denote the update of the iterative algorithm after m data is received. For federated learning we then have

$$\hat{y}(m+1) = A\hat{y}(m) + \sum_{l=1}^{N} c_l(m)\hat{y}_l(m) \tag{2}$$

where $\hat{y}(m)$ is the function the central processor is learning. The term A is a boolean variable that is 1 when we use asynchronous updates, $\hat{y}_l(m)$ are the learning updates at time m of the lth sensor, there are N sensors, and $c_l(m)$ is a weighting given for each sensor. For federated learning we will likely have $c_l(m) = 0$ for most l and m as the central processor does not always update learning parameters when data is received by a sensor. There will also be cases when only a certain subset of the sensors are asked to submit their learning updates with all other sensors having $c_l(m) = 0$.

A key consideration is the updating rules. Assume we use RFF with the basis function information sent to all sensors before learning commences. The simplest updating rule is an asynchronous updating rule (AUR1). When a sensor receives data the sensor sends an acknowledgement to the central processor. The central processor then sends the updated weights αs to the edge processor. The edge processor then updates these weights using a stochastic gradient algorithm such as the Kernel Least Mean Square (KLMS) algorithm. The edge processor then sends the updated weights to the central processor and the central processor waits for an acknowledgement of the next input data that is received by sensor. In this case communications savings may be minimal (sending data versus updates of αs), but more savings can be achieved by compressing information sent, having edge processors update only after having received a prescribed amount of data, and also rejecting data that does not provide sufficient information. For AUR1, the behavior of the learning algorithm is similar to the centralized online learning algorithms and conditions for mean convergence and mean square convergence are easily established. However, as mentioned above communication savings may be minimal.

We can also consider AURp where $p > 1$. Here each sensor has a counter of how much data it has received since it last communicated with the central processor and exchanged information about weight parameters. As each data is received by the sensor, it updates its learning in an online manner. This can be by KLMS, a weighted averaging of KLMS, or an affine projection algorithm. After p data has been received it sends a query to the central processor and sends the cumulative updates it has received since the last update. The central processor updates its model and sends the information to the sensor. Let sensor l be updated at iteration m, then the updates can be described by

$$\alpha(m+1) = \alpha(m) + c_l(m)\alpha_{(l)}(m) \tag{3}$$

$$\alpha_{(l)}(m+1) = \alpha(m+1) \tag{4}$$

where $\alpha(m)$ are the weights at iteration m and $\alpha_{(l)}(m)$ are the cumulative weight updates from sensor l since it was last updated. Here the communication costs are a factor of p less than AUR1.

We also consider a synchronous updating rule after k updates (SURk). In this case each of the N sensors learns on its own updating weights using a kernel LMS algorithm or online learning algoritm. After m cumulative updates from the N sensors where $k > N$ the central processors gets the weights, αs from each of the sensors and after updating. The central processor updates its model and sends the information to the sensor. If a synchronous update occurs at iteration m, this is described by

$$\alpha(m+1) = \sum_{l=1}^{N} c_l(m)\alpha_{(l)}(m) \tag{5}$$

$$\alpha_{(l)}(m+1) = \alpha(m+1) \;\; 1 \leq l \leq N \tag{6}$$

For synchronous updates, $\alpha_{(l)}(m)$ is the current weight values of sensor l at time m. Here the communication savings of SURm is a factor of m/N over AUR1.

Note that convergence for stochastic gradient algorithms such as LMS usually follow an exponential curve if step size μ is not too large. The algorithms AURp and SURk will perform better than if no information is transmitted between each sensor and the central processor. If no information is given between the sensor and central processor convergence rates for AURp and SURm will be a factor of p and k/N slower than a centralized updating system. However, each of the sensors for both algorithms receive information from the central processor so convergence will be quicker.

Another method to reduce communication and computation costs is to use selective sampling methods. In selective sampling data is processed only if it can provide new information for the learning system. One implementation is set membership filtering (SMF) which is an adaptive filtering paradigm that features data-dependent selective update of the filter parameters [14], [15]. We have incorporated SMF for kernel adaptive filtering [16]. SMF can be implemented with the federated learning algorithms considered here and will be considered in a future paper.

IV. Numerical Simulations

Here we consider a nonlinear time series used in [10]. The dynamics are described by the following nonlinear difference equation

$$x(n) = (0.8 - 0.5\exp(-x(n-1)^2))x(n-1)$$
$$-(0.3+0.9\exp(-x(n-1)^2))x(n-2) + 0.1\sin(x(n-1)\pi) + v(n) \tag{7}$$

where we have initial conditions $x(0) = x(1) = 0.1$ and $v(n)$ is independent and identically distributed zero mean Gaussian noise with standard deviation 0.1. We generated 3000 data points from this time series and we considered learning using Random Fourier Features. The data points were randomly drawn with two inputs and one output. A vector of 100 was used from the Random Fourier Features from equation (1). Here we considered a homogenous Federated Learning case with ten sensors randomly getting data from the 3000 data points. We considered the two algorithms considered in Section III, AURp and SURk. Results are shown in Figures 1 and 2. The MSE is from the average squared error of the last 500 data points of the time series.

From Figure 1 we see that AUR5 performance has convergence time between 1.5 and 1.8 times slower than the centralized LMS with communication costs a factor of five times less than AUR1. Each sensor learns based on their data and when the sensor's data is updated by the central processor they get information from updates by the other sensors. The central processor sending updates to the sensors definitely speeds up learning. The performance of AUR10 and AUR15 is slightly worse than AUR5, but these algorithms also benefit from receiving information from the central processor. AUR15

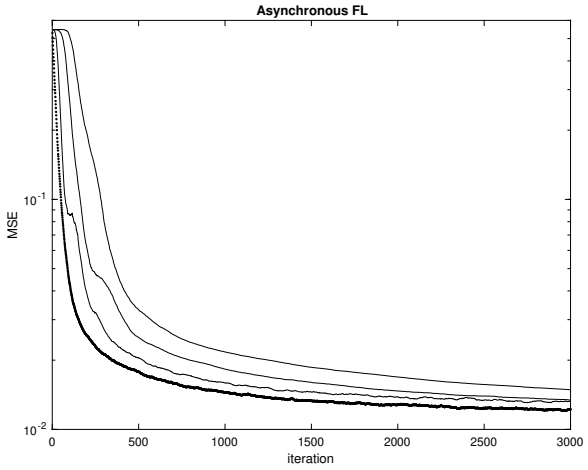

Fig. 1. AURp. Lowest curve is centralized LMS with step size $\mu = 0.2$. Other curves are for $p = 5, 10, 15$ with step size $\mu = 0.8$. Each simulation is averaged over 100 trials.

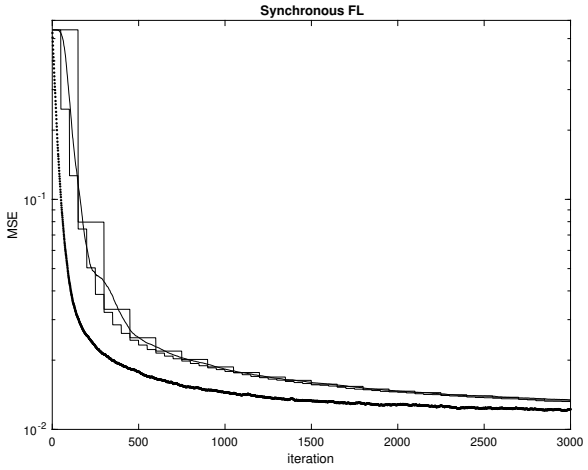

Fig. 2. SURk. Lowest curve is centralized LMS with stepsize $\mu = 0.2$. Other curves are for $k = 50, 100, 150$ with step size $\mu = 0.8$. Each simulation is averaged over 100 trials.

with communication costs less than AUR1 by a factor of fifteen has convergence speed that is about four times slower than centralized LMS. All four simulations converge to a similar steady state MSE. Step sizes for AURp algorithms can be larger than centralized LMS as there is more averaging when the central processor accumulates weights from all sensors and gives its information to the sensors when they update.

From Figure 2 we see that SUR50 performance (which has similar communication costs to AUR5) converges at a slightly slower rate than AUR5. However, SUR100 and SUR150 has similar performance to SUR50. This indicates the importance of the central processor sending weight information to sensors on a regular basis. The performance of SUR100 and AUR10 is similar and the performance of SUR150 is slightly better than

AUR15. An intuitive explanation for this is that updates for SUR150 are regular and occur every 150 iterations. However, for AUR15 the number of iterations between updates could be much longer than 150 iterations and this will likely result in slower convergence for these sensors and overall slower convergence. Step sizes for SURk can also be larger than centralized LMS for similar reasons as AURp.

V. Summary and Further Directions

This paper has established a general framework for real-time distributed learning using kernel methods using principles of federated learning. The learning algorithms can be supervised learning algorithms or unsupervised learning algorithms. The focus is on learning a single task. Kernel methods are used as the solution can be posed in terms of a convex optimization problem. Furthermore we use a squared error cost function with equality constraints resulting in a least squares support vector machine (LS-SVM) [6]. Solutions involve solving a least squares problem either in primal or dual spaces In the dual space we use a dictionary of basis vectors (either support vectors or RFF) and the solution can be expressed as a linear combination of kernel values of a linear filter or Random Fourier Features. This setup is readily amenable to online federated learning where communications between edge processors and the central processor primarily involves transmitting weight updates α. We discuss various updating strategies that balance communications, performance, and computational complexity.

Simulation experiments on a nonlinear time series show some of the benefits of using synchronous and asynchronous federate learning. Convergence rate is slower when updating less often, but the rate of slower convergence is shown to be much less than the rate of communication savings. We have recently discussed other ways to save communication costs by using partial observations where only some of the weights of the Random Fourier Features are updated, [17]. For this case performance is comparable to centralized LMS with significant communication savings. We will also explore how both spatial and temporal relationships between the edge processors have effects on the learning algorithms. Some of these can be incorporated into the regularization parameter that can involve the spatial relationship described by graph signal processing (e.g. Laplacian matrix). Further research will also study a theoretical framework confirming the performance of the different algorithms.

The methods developed here can be applied to a number of applications where distributed processing and learning occurs. This includes applications in power systems where local sensors gather data. In [18] an unsupervised online one class least squares support vector learning algorithm to learn a the probability density function of gathered data. This algorithm was able to detect outlier data and was able to detect bad data on the electrical grid. This algorithm can easily be modified using the federated learning model considered here. There are also many other applications ranging from autonomous systems to healthcare systems to robotics that we hope to address in future research.

Acknowledgements

This work was supported in part by the United States National Science Foundation (NSF) grant ECCS-2142987.

References

[1] J. Konečný, B. McMahan, and D. Ramage. Federated optimization:distributed optimization beyond the datacenter, 2015.
[2] P. Kairouz and et. al. Advances and open problems in federated learning, 2021.
[3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Proc. Magazine*, 37(3):50–60, May 2020.
[4] N. Aronszajn. Theory of reproducing kernels. *Trans. Amer. Math. Soc.*, 68:337–404, 1950.
[5] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008.
[6] J. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least squares support vector machines*. World Scientific Publishing Co., Singapore, 2002.
[7] C. K. Williams and C. E. Rasmussen. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
[8] A. Ortega, P. Frossard, J. Kovačević , and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *IEEE Proceedings*, 106(5):808–828, May 2018.
[9] Y. Engel, S. Mannor, and R. Meir. The kernel recursive least-squares algorithm. *IEEE Trans. on Signal Proc.*, 52(3):2275–2285, Aug. 2004.
[10] C. Richard, J.-C. Bermudez, and P. Honeine. Online prediction of time series data with kernels. *IEEE Trans. on Signal Proc.*, 57(3):1058–1067, Mar. 2009.
[11] V. R. M. Elias, V. C. Gogineni, W. A. Martins, and S. Werner. Adaptive graph filters in reproducing kernel hilbert spaces: Design and performance analysis. *IEEE Trans. on Signal & Information Proc. over Networks*, 7(12):62–74, 2020.
[12] V. R. M. Elias, V. C. Gogineni, W. A. Martins, and S. Werner. Kernel regression on graphs in random fourier features space. In *IEEE ICASSP 2021*, pages 5235–5239, 2021.
[13] S. Haykin. *Adaptive Filter Theory, 5th Ed.* Pearson, 2014.
[14] E. Fogel and Y.-F. Huang. On the value of information in system identification?bounded noise case. *Automatica*, 18(2):229–238, Mar. 1982.
[15] S. Dasgupta and Y.-F. Huang. Asymptotically convergent modified recursive least-squares with data-dependent updating and forgetting factor for systems with bounded noise. *IEEE Trans. on Inform. Theory*, 63(3):383–392, May 1987.
[16] K. Chen, S. Werner, A.Kuh, and Y.-F. Huang. Nonlinear adaptive filtering with kernel set-membership approach. *IEEE Trans. on Signal Proc.*, 68(2):1515–1528, Feb. 2020.
[17] Discussions with V. C. Gogineni, S. Werner, and Y.-F. Huang, 2021.
[18] M. Uddin and A. Kuh. Online least-squares one-class support vector machine for outlier detection in power grid data. In *2016 IEEE ICASSP*, Shanghai, China, Mar. 2016.