Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier

A Neural Network Approach for Online Nonlinear Neyman-Pearson Classification

BASARBATU CAN[1], HUSEYIN OZKAN[1]

[1]Sabanci University, Faculty of Engineering and Natural Sciences, Electronics Engineering, 34956 Tuzla, Istanbul, Turkey (e-mail: [basarbatucan, huseyin.ozkan]@sabanciuniv.edu)

Corresponding author: Basarbatu Can (e-mail: basarbatucan@sabanciuniv.edu).

This work was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under Contract 118E268.

ABSTRACT We propose a novel Neyman-Pearson (NP) classifier that is both online and nonlinear as the first time in the literature. The proposed classifier operates on a binary labeled data stream in an online manner, and maximizes the detection power about a user-specified and controllable false positive rate. Our NP classifier is a single hidden layer feedforward neural network (SLFN), which is initialized with random Fourier features (RFFs) to construct the kernel space of the radial basis function at its hidden layer with sinusoidal activation. Not only does this use of RFFs provide an excellent initialization with great nonlinear modeling capability, but it also exponentially reduces the parameter complexity and compactifies the network to mitigate overfitting while improving the processing efficiency substantially. We sequentially learn the SLFN with stochastic gradient descent updates based on a Lagrangian NP objective. As a result, we obtain an expedited online adaptation and powerful nonlinear Neyman-Pearson modeling. Our algorithm is appropriate for large scale data applications and provides a decent false positive rate controllability with real time processing since it only has $O(N)$ computational and $O(1)$ space complexity (N: number of data instances). In our extensive set of experiments on several real datasets, our algorithm is highly superior over the competing state-of-the-art techniques, either by outperforming in terms of the NP classification objective with a comparable computational as well as space complexity or by achieving a comparable performance with significantly lower complexity.

INDEX TERMS Neyman-Pearson, Online, Nonlinear, Classification, Large scale, Kernel, Neural network

I. INTRODUCTION

DESIGNING a binary classifier with asymmetrical costs for the errors of type I (false positive) and type II (false negative) [1]–[3], or equivalently designing a Neyman-Pearson classifier [4], is required in various applications ranging from facial age estimation [5], multi-view learning [6] and software defect prediction [7] to video surveillance [8] and data imputation [9]. For example, in medical diagnostics, type II error (misdiagnosing as healthy) has perhaps more severe consequences, whereas type I error (misdiagnosing as unhealthy) may result in devastating psychological effects [10]. In this example, the error costs must be set probably asymmetrically for the cost sensitive learning [1], [2] of the desired classifier, however, it could be difficult to determine the right cost structure to be imposed on the errors. Another example with the same difficulty is anomaly detection for the security and surveillance applications. In such applications of detecting anomalies (e.g. accidents, crimes, frauds, violations), the type I error rate must certainly be controlled since giving a false alarm, i.e., false anomaly, too often is frustrating and costly as it draws unnecessary attention from security agents. It is also important to maintain the reliability of detections and not fail to draw attention in the case of a true anomaly. Hence, the user must set the costs of both error types to match the bearable false alarm rate, however, setting that correctly could be again difficult to guarantee to not give a false alarm, for instance, no more than once a day or week. Therefore, it is often more convenient and practical -but technically equivalent [4]- to describe the user needs by the maximum tolerable type I error, cf. [11] and the references therein, instead of having to determine the error costs to meet the tolerance. This leads to the Neyman-

Pearson (NP) characterization of the desired classifier [4] and false positive rate controllability, where the goal is to maximize the detection power, i.e., minimize type II error, while upper-bounding the false positive rate, i.e., type I error, by a user-specified threshold. In this paper, we target and solve the problem of designing a computationally highly efficient NP classifier while achieving powerful nonlinear data modeling with potential applications in, for instance, security, surveillance and diagnostics.

To this goal, as the first time in the literature, we introduce a novel online and nonlinear NP classifier based on a single hidden layer feedforward neural network (SLFN), which is sequentially learned with a Lagrangian non-convex NP objective (i.e. maximum detection power about a controllable user specified false positive rate). We use stochastic gradient descent (SGD) optimization for scalability to voluminous data and online processing with limited memory requirements. During the SGD iterations, we a) sequentially infer the value of the Lagrangian multiplier in a data driven manner to obtain the correspondence between the asymmetrical error costs and the desired type I error rate, and b) update all the SLFN parameters to maximize the detection power (minimize the resulting cost sensitive classification error) at the desired false positive rate. To achieve powerful nonlinear modeling and improve scalability, we use the SLFN in a kernel inspired manner, cf. [12] for the kernel approach to nonlinearity. For this purpose, the hidden layer is initialized with a sinusoidal activation to approximately construct the high dimensional kernel space (of any symmetric and shift invariant kernel under Mercer's conditions, e.g., radial basis function) through the random Fourier features (RFFs) [12]. The output layer follows with identity activation.

The main contribution of our work is that we are the first to propose a Neyman-Pearson (NP) classifier that is both online and nonlinear. Our algorithm -as an important novel addition to the literature- is appropriate for contemporary fast streaming large scale data applications that require real time processing with capabilities of complex nonlinear modeling and false positive rate controllability. In our extensive experiments, the introduced classifier yields significantly better results compared to the competing state-of-the-art NP techniques; either performance-wise (in terms of the detection power and false positive rate controllability) at a comparable computational and space complexity, or efficiency-wise (in terms of complexity) at a comparable performance. The presented study is also the first to design a neural network (as an SLFN) in the context of NP characterization of classifiers, which is expected to open up new directions into deeper architectures since the NP approach has been left surprisingly unexplored in deep learning.

In the following Section II, we discuss state-of-the-art NP classification methods. We provide the problem description in Section III, and then introduce our technique for online and nonlinear NP classification in Section IV. After the experimental evaluation is presented in Section V, the results are analyzed in Section VI and we conclude in Section VII.

II. RELATED WORK

Neyman-Pearson classification has found a wide-spread use across various applications due to the direct control over the false positive rate that it offers, cf. [11] and the references therein. For example, an NP classifier is commonly employed for anomaly detection, where the false positive rate controllability is particularly important. In the one class formulation (due to the extreme rarity of anomalies) of anomaly detection [13]–[16], the NP classification turns out (when the anomalies are assumed uniformly distributed) estimating the minimum volume set (MVS) that covers $1 - \tau$ fraction of the nominal data (τ is the desired false positive rate). Then, an instance is anomalous if it is not in the MVS. A structural risk minimization approach is presented in [13] for learning the MVS based on a class of sets generated by a dyadic tree partitioning. Geometric entropy minimization [14] and empirical scoring [15] can also be used to estimate the MVS, both of which are based on nearest neighbor graphs. The scoring of [15] is later extended to the local anomaly detection in [16] and a new one class support vector machines (SVM) in [17]. Although the algorithms in these examples with batch processing, i.e., not online, have decent theoretical performance guarantees, they are not scalable to large scale data due to their prohibitive computational as well as space complexity and hence they cannot be used in our scenario of fast streaming applications. Online extensions to the original batch one class SVM [18], which can be shown to provide an estimator of the MVS [19], have been proposed for distributed processing [20] and wireless sensor networks [21]. However, neither these online extensions nor the original one class SVM address the false positive rate controllability as they require additional manual parameter tuning for that. In contrast, our proposed online NP classifier directly controls (without parameter tuning) the false positive rate and maximizes the detection power with nonlinear modeling capabilities. Furthermore, NP formulation in the one class setting requires the knowledge of the target density (e.g., anomaly), which is often unknown and thus typically assumed to be uniform; but then the problem can be turned into a supervised binary NP classification by simply sampling from the assumed target density. On the other hand, when there is also data from the target class, the one class formulation in aforementioned studies does not directly address how to incorporate the target data. Hence, our two class supervised formulation of binary NP classification also covers the solution of the one class classification, and our proposed algorithm is consequently more general and applicable in both cases of target data availability.

Among the two class binary NP classification studies (cf. [11] for a survey), plug-in approaches (such as [22] and [23]) based on density estimation as an application of the NP lemma [24] are difficult to be applied in high dimension due to overfitting [14]. Particularly, [22] exploits the expectation-maximization algorithm for density estimation using a neural network with -however- batch processing and manual tuning for finding the threshold to satisfy the NP type I error constraint. In [25], a neural network is trained with symmetric error costs for modeling the likelihood ratio, which is thresholded to match the desired false positive rate but determining the threshold requires additional work. Moreover, the approach of thresholding after training with symmetric error costs (cf. [11] for other examples in addition to [25]) does not yield NP optimality, since NP classification requires training with asymmetric error costs corresponding to the desired false positive rate. Unlike our presented work, approaches in [22], [23], [25] are also not online and do not allow real time false positive rate controllability. Recall that NP classification is equivalent to cost sensitive learning [4] when the desired false positive rate can be accurately translated to error costs, but achieving an accurate translation, i.e., correspondence, is typically nontrivial requiring special attention [4], [26]. This correspondence problem is addressed i) in [4] as parameter tuning with improved error estimations, and ii) in [26] as an optimization with the assumption of class priors and unlabeled data. Besides the exploitation of SVM [4], other classifiers such as logistic regression [27] have also been considered in [28] and incorporated into a unifying NP framework as an umbrella algorithm. We emphasize that these approaches, the SVM based tuning approach [4] and the risk minimization of [24] as well as the umbrella algorithm [28] in addition to the optimization of [26], do not satisfy our computational online processing requirements, as they are batch techniques and not scalable to large scale data.

In most of the contemporary fast streaming data applications, such as computer vision based surveillance [29] and time series analysis [30], computationally efficient processing along with only limited space needs is a crucial design requirement. This is necessary for scalability in such applications which constantly generate voluminous data at unprecedented rates. However, the literature about the Neyman-Pearson classification (cf. [11] for the current state) appears to be fairly limited from this large scale efficient processing point of view. Out of very few examples, a linear-time algorithm for learning a scoring function and thresholding is presented in [31], which is still not an online algorithm (i.e. it is not designed to process data indefinitely on the fly) since batch processing is assumed with large space complexity and processing latency. Moreover, scoring of [31] is similar to the one of [15] but -unlike [15]- trades off NP optimality for linear-time processing. Also, the technique of [31] is restricted to linearly separable data only, and it requires to adjust thresholding for false positive rate controllability which can be seen impractical. The NP technique of [30] is truly online (and one class) but it is strongly restricted to Markov sources, thus fails in the case of general non-Markov data (whereas our proposed algorithm has no such restriction). Another online NP classifier is presented in [32] without strict assumptions unlike [30], but for only linearly separable data while leaving the online generalization to nonlinear setting as a future research direction.

To our best knowledge, online NP classification has not been studied yet in the nonlinear setting. Thus, as the first time in the literature, we solve the online and nonlinear NP classification problem based on a kernel inspired SLFN within the non-convex Lagrangian optimization framework of [32], [33], and use SGD updates for scalability. Our NP classifier exploits Fourier features [12] and sinusoidal activations in the hidden layer of the SLFN (hence the name kernel inspired) to achieve a powerful nonlinear modeling with high computational efficiency and online real time processing capability.

Random Fourier features (RFFs) and also kernels in general have been successfully used for classification and regression of large scale data (please refer to [12], [34], [35] and [36] for examples). Our presented work also exploits RFFs (during SLFN initialization) for large scale learning but, in contrast, for the completely different goal of solving the problem of online nonlinear Neyman-Pearson (NP) classification with neural networks in a non-convex Lagrangian optimization framework. Furthermore, the presented work learns the useful Fourier features with SGD updates beyond the initial randomness. On the other hand, kernels and RFFs have been previously studied in conjunction with neural networks. For example, computational relations from certain kernels to large networks are drawn in [37], and a kernel approximating convolutional neural network is proposed in [38] for visual recognition. In particular, RFFs have been used to learn deep Gaussian processes [39], and for hybridization in deep models to connect linear layers nonlinearly [40]. A radial basis function (rbf) network is proposed in [41] with batch processing, i.e., not online, which briefly discusses a heuristic by varying rbf parameters to manually control the false positive rate. Note that our SLFN is not an rbf network since we explicitly construct (during initialization) the kernel space in the hidden layer without a further need for kernel evaluations. We stress that the hidden layer of our SLFN for NP classification is same as the RFF layer of [42] for kernel learning (a simultaneous development of the same layer). The RFF layer in [42] is proposed as a building block to deep architectures for the goal of kernel learning. However, our goal of designing an online nonlinear NP classifier is completely different. Hence, our formulation, network objective and the resulting training process as well as our algorithm and experimental demonstration in this paper are fundamentally different compared to [42]. Moreover, online processing is not a focus in these studies except that [38] and [39] address scalability to voluminous data; and none of those (including [42] for kernel learning, and [38] and [39] for scalability) consider our goal of NP classification. Finally, we note that the presented study comprehensively extends our previous conference paper [43] that only had certain initial findings of the preliminary version of our algorithm NP-NN presented here. In this paper, compared to our conference paper [43], we additionally 1) introduced Fourier feature learning (such features have been randomly drawn and kept untrained in [43]) in the nonconvex optimization framework of neural networks, 2) performed significantly more extensive experiments with a larger number datasets based on additional performance metrics (such as the NP-score), and 3) analyzed from different perspectives such as statistical significance and complexity.

III. PROBLEM DESCRIPTION

We provide the problem description in this section. Regarding the notation, all vectors are column vectors and they are denoted by boldface lower case letters. For a vector $\boldsymbol{w}$, its transpose is represented by $\boldsymbol{w}'$ and the time index is given as subscript, i.e., $\boldsymbol{w}_t$. Also, a) $1_{\{\cdot\}}$ is the indicator function returning 1 if its argument condition holds, and returning 0, otherwise; and b) $\mathrm{sgn}(\cdot)$ is the sign function returning 1 if its argument is positive, and returning -1, otherwise.

Neyman-Pearson (NP) classification [11] seeks a classifier δ for a d dimensional observation $\mathbb{R}^d \ni \boldsymbol{x}$ to choose one of the two classes $H_y : \boldsymbol{x} \sim p_y(\boldsymbol{x})$ as $\delta(\boldsymbol{x}) = \hat{y} \in \{-1, +1\}$, where $y \in \{-1, +1\}$ (non-target: -1, target: 1) is the true class label and $p_y(\boldsymbol{x})$ are the corresponding conditional probability density functions. The goal is to minimize the type II error (non-detection) rate $\mathrm{P_{nd}}$

$$\begin{aligned}\mathrm{P_{nd}}(\delta) &= \int_{\forall \boldsymbol{x} \in \mathbb{R}^d} 1_{\{\hat{y}=-1\}} p_1(\boldsymbol{x}) d\boldsymbol{x} \\ &= E_1[1_{\{\hat{y}=-1\}}]\end{aligned} \tag{1}$$

(thus, the detection power $\mathrm{P_{td}} = 1 - \mathrm{P_{nd}}$ is maximized) while upper bounding the type I error $\mathrm{P_{fa}}$ (false positive) rate by a user specified threshold τ as

$$\begin{aligned}\mathrm{P_{fa}}(\delta) &= \int_{\forall \boldsymbol{x} \in \mathbb{R}^d} 1_{\{\hat{y}=1\}} p_{-1}(\boldsymbol{x}) d\boldsymbol{x} \\ &= E_{-1}[1_{\{\hat{y}=1\}}] \leq \tau\end{aligned} \tag{2}$$

with E_y being the corresponding expectations. Namely, δ^* is an NP classifier, if it satisfies

$$\delta^* = \arg\min_{\delta} \mathrm{P_{nd}}(\delta) \text{ subject to } \mathrm{P_{fa}}(\delta) \leq \tau.$$

The Neyman-Pearson (NP) lemma [24], [44] states that the likelihood ratio test provides an optimal solution to the constrained optimization above once the false alarm rate of the test is equated to the user specified threshold τ. Moreover, such a likelihood ratio test always exists, and it is unique up to a subset in the observations space that has a zero probability mass under both hypotheses. We refer to [44] for a rigorous proof. Thus, the likelihood ratio $\frac{p_1(\boldsymbol{x})}{p_{-1}(\boldsymbol{x})}$ provides the NP test, i.e.,

$$\begin{aligned}&\delta^*(\boldsymbol{x}) = -1, \text{ if } u(\boldsymbol{x}) = \frac{p_1(\boldsymbol{x})}{p_{-1}(\boldsymbol{x})} - v(\tau) \leq 0, \text{ and} \\ &\delta^*(\boldsymbol{x}) = 1, \text{ otherwise,}\end{aligned} \tag{3}$$

where the offset $v(\tau)$ is chosen to satisfy the false positive rate constraint. Hence, finding the discriminant function u is sufficient for NP testing.

The discriminant function u can be simplified in many cases, and it might be linear or nonlinear as a function of $\boldsymbol{x}$ after full simplification. We provide two corresponding examples in the following. For instance, if the conditional densities $p_y(\boldsymbol{x})$ are both Gaussian with same covariances, then the discriminant is linear. On the other hand, in the example of one class classification [18] with applications to anomaly detection, there is typically no data from the target (anomaly) hypothesis because of the extreme rarity of anomalies, and there is also not much prior information due to the unpredictable nature of anomalies. Hence, the usual approach is to assume that the target density is uniform (with a finite support) [15], i.e., $p_1(\boldsymbol{x}) = c$. Then, the critical region $\mathrm{MVS} = \{\boldsymbol{x} \in \mathbb{R}^d : 1/p_{-1}(\boldsymbol{x}) \leq v(\tau)\}$ for the NP test to decide non-target, i.e., $\delta^*(\boldsymbol{x}) = -1$, is known as the minimum volume set (MVS) [13] covering $1-\tau$ fraction of the non-target instances, i.e., $v(\tau)$ is set with simplification such that $\int_{\boldsymbol{x} \notin \mathrm{MVS} \subset \mathbb{R}^d} p_{-1}(\boldsymbol{x}) d\boldsymbol{x} = \tau$. Consequently, MVS has the minimum volume with respect to the uniform target density and hence maximizes the detection power. Here, the MVS discriminant $u(\boldsymbol{x}) = 1/p_{-1}(\boldsymbol{x}) - v(\tau)$ (after simplification) is generally nonlinear, for instance, even when $p_{-1}(\boldsymbol{x})$ is Gaussian with zero mean unit-diagonal covariance. Therefore, we emphasize that the discriminant u of the NP test[1] might be arbitrarily nonlinear in general. Furthermore, since the discriminant definition requires the knowledge of the conditional densities p_y which are unavailable in most realistic scenarios, the discriminant u is unknown. For this reason, NP classification refers to the data driven statistical learning of an approximation $f^* \in \mathcal{H}$ of the unknown discriminant u based on given two classes of data $\{(\boldsymbol{x}_t, y_t)\}$, where $\mathcal{H}$ is an appropriate set of functions which is sufficiently powerful to model the complexity of u.

As a result, the data driven statistical learning of the NP classifier f^* is obtained as the output of the following NP optimization:

$$u \simeq f^* = \arg\min_{f \in \mathcal{H}} \hat{\mathrm{P}}_{\mathrm{nd}}(f) \text{ subject to } \hat{\mathrm{P}}_{\mathrm{fa}}(f) \leq \tau, \tag{4}$$

where

$$\hat{\mathrm{P}}_{\mathrm{nd}}(f) = \frac{\sum_{\forall t: y_t = 1} 1_{\{f(\boldsymbol{x}_t) \leq 0\}}}{\sum_{\forall t: y_t = 1} 1} \text{ and}$$
$$\hat{\mathrm{P}}_{\mathrm{fa}}(f) = \frac{\sum_{\forall t: y_t = -1} 1_{\{f(\boldsymbol{x}_t) > 0\}}}{\sum_{\forall t: y_t = -1} 1}$$

empirically estimates the type I (expectation in (1)) and type II (expectation in (2)) errors, respectively. For example, [32] studies this optimization in (4) for the set $\mathcal{H}$ of linear discriminants, in which case -however- the resulting linear NP classifier is largely suboptimal in most realistic scenarios; for example, the MVS estimation for anomaly detection requires to learn nonlinear class separation boundaries with a nonlinear discriminant.

Our goal in the presented work is to develop, as the first time in the literature to our best knowledge, an online nonlinear NP classifier for any given user-specified desired

[1]Note that knowing the continues valued discriminant u is equivalent to knowing the discrete valued test δ^* due to one-to-one correspondence, i.e., $\delta^*(\boldsymbol{x}) = \mathrm{sgn}(u(\boldsymbol{x}))$. Hence, in the rest of the paper, we refer to the discriminant as the NP classifier as well.

false positive rate τ with real time processing capability. In particular, we use a kernel inspired single hidden layer feed forward neural network (SLFN), cf. Fig. 1, to model the set $\mathcal{H}$ of nonlinear candidate discriminant functions in (4) as

$$\mathcal{H} = \{f : f(\boldsymbol{x}) = h_o(h_h(\boldsymbol{\alpha}\boldsymbol{x})'\boldsymbol{w} + b), \forall\boldsymbol{\alpha}, \forall\boldsymbol{w}, \forall b\}, \quad (5)$$

where $\boldsymbol{\alpha}$ and $(\boldsymbol{w}, b)$ are the hidden and output layer parameters, and h_h and h_o are the nonlinear hidden and identity output layer activations. We sequentially learn the SLFN parameters based on the NP objective (that is maximizing the detection power about a user-specified false positive rate as given in (4)) with stochastic gradient descent (SGD) to obtain the nonlinear classification boundary, i.e., to estimate the unknown discriminant u, in an online manner while maintaining scalibility to voluminous data.

The data processing in our proposed algorithm is computationally highly efficient and truly online with $O(N)$ computational and $O(1)$ space complexity (N is the total number of processed instances). Namely, we sequentially observe the data $\boldsymbol{x}_t \in \mathbb{R}^d$ indefinitely without knowing a horizon, and decide about its label $\hat{y}_t \in \{1, -1\}$ as $\hat{y}_t = 1$ if the SLFN $f_t \in \mathcal{H}$ at time t provides $f_t(\boldsymbol{x}_t) > 0$, and as $\hat{y}_t = -1$, otherwise. Then, we update our model f_t, i.e., update the SLFN at time t, to obtain $f_{t+1} \in \mathcal{H}$ based on the error $y_t - \hat{y}_t$ via SGD and discard the observed data, i.e., $\boldsymbol{x}_t$ and y_t, without storing. Hence, each instance is processed only once. In this processing framework, $f_t \to f^* \in \mathcal{H}$ models the NP discriminant u in (3). As a result of this processing efficiency, our algorithm is appropriate for large scale data applications.

IV. SLFN FOR ONLINE NONLINEAR NP CLASSIFICATION

In order to learn nonlinear Neyman-Pearson classification boundaries, we use a single hidden layer feed forward neural network (SLFN), illustrated in 1, that is designed based on the kernel approach to nonlinear modeling (cf. [12] and the references therein for the mentioned kernel approach). Namely, the hidden layer is randomly initialized to explicitly transform the observation space (via $\phi_{\boldsymbol{\alpha_1}}$) into a high dimensional kernel space with sinusoidal hidden layer activations by using the random Fourier features [12]. We use a certain variant of the perceptron algorithm [45] as the output layer with identity activation followed by a sigmoid loss. Based on this SLFN, we sequentially (in a truly online manner) learn the network parameters, i.e., the classifier parameters $\boldsymbol{w}_t, b_t$ as well as the kernel mapping parameters $\boldsymbol{\alpha_t}$, through SGD in accordance with the NP optimization objective (4).

In the hidden layer of the SLFN, the randomized initial transformation $\phi_{\boldsymbol{\alpha_1}} : \mathbb{R}^d \to \mathbb{R}^{2D}$ at time $t = 1$,

$$\mathbb{R}^d \ni \boldsymbol{x} \to \tilde{\boldsymbol{x}} = \phi_{\boldsymbol{\alpha_1}}(\boldsymbol{x}) \in \mathbb{R}^{2D}, \quad (6)$$

is constructed based on the fact (as provided in [12]) that any continuous, symmetric and shift invariant kernel can be approximated as $k(\boldsymbol{x}^i, \boldsymbol{x}^j) \triangleq k(\boldsymbol{x}^i - \boldsymbol{x}^j) \approx \phi_{\boldsymbol{\alpha_1}}(\boldsymbol{x}^i)'\phi_{\boldsymbol{\alpha_1}}(\boldsymbol{x}^j)$ with an appropriately randomized kernel feature mapping.

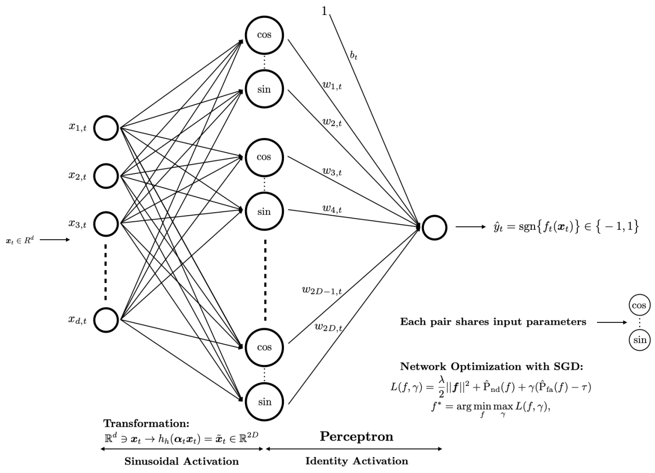
1
b_t
$w_{1,t}$
$w_{2,t}$
$w_{3,t}$
$w_{4,t}$
$w_{2D-1,t}$
$w_{2D,t}$
cos
sin
$x_{1,t}$
$x_{2,t}$
$x_{3,t}$
$x_{d,t}$
$\boldsymbol{x}_t \in R^d$
$\hat{y}_t = \mathrm{sgn}\{f_t(\boldsymbol{x}_t)\} \in \{-1, 1\}$
Each pair shares input parameters
Network Optimization with SGD:
$L(f,\gamma) = \frac{\lambda}{2}||\boldsymbol{f}||^2 + \hat{\mathrm{P}}_{\mathrm{nd}}(f) + \gamma(\hat{\mathrm{P}}_{\mathrm{fa}}(f) - \tau)$
$f^* = \arg\min_f \max_\gamma L(f,\gamma),$
Transformation:
$\mathbb{R}^d \ni \boldsymbol{x}_t \to h_h(\boldsymbol{\alpha}_t\boldsymbol{x}_t) = \tilde{\boldsymbol{x}}_t \in \mathbb{R}^{2D}$
Perceptron
Sinusoidal Activation
Identity Activation

FIGURE 1: The single hidden layer feed forward neural network (SLFN) that we use for online nonlinear Neyman-Pearson (NP) classification is illustrated. The hidden layer is initialized to approximately construct the high dimensional kernel space (e.g., radial basis function) via random Fourier features (RFFs) with sinusoidal activation. The output layer follows with identity activation. This network is compact and strongly nonlinear with expedited learning ability thanks to i) the exponential convergence of the inner products (w.r.t. the number of hidden nodes) in the space of RFFs to the true kernel with an excellent random network initialization, and ii) the learning of Fourier features (instead of relying on randomization) by the data driven network updates. We learn the network parameters sequentially via SGD based on a non-convex Lagrangian NP objective. The result is an exptedited powerful nonlinear NP modeling with high computational efficiency and scalibility.

Note that the kernel $k(\boldsymbol{x}^i, \boldsymbol{x}^j)$ is an implicit access to the targeted high dimensional kernel space as it encodes the targeted inner products. This kernel space is explicitly and approximately constructed by the sinusoidal hidden layer activations of the SLFN in which the new inner products across activations approximate originally targeted inner products. Hence, linear techniques applied to the sinusoidal hidden layer activations can learn nonlinear models. In our method, we use the radial basis function (rbf) kernel[2] $k(\boldsymbol{x}^i, \boldsymbol{x}^j) = \exp(-g||\boldsymbol{x}^i - \boldsymbol{x}^j||^2)$ with the bandwidth parameter g (that is inversely related to the actual bandwidth).

In order to obtain a randomized mapping that explicitly constructs the kernel space, one can apply here the Bochner's theorem by using the derivation in [12]. This theorem states that (quoting from [12]) "a continuous kernel $k(\boldsymbol{x}^i, \boldsymbol{x}^j) = k(\boldsymbol{x}^i - \boldsymbol{x}^j)$ on $\mathbb{R}^d$ is positive definite if and only if $k(\boldsymbol{x}^i - \boldsymbol{x}^j)$ is the Fourier transform of a non-negative measure". Then,

[2]We use the rbf kernel in this study as an example but it is not required. Thus, the presented technique can be straightforwardly extended to any symmetric and shift invariant kernel satisfying the Bochner's theorem, cf. [12].

$$\begin{aligned}
k(\boldsymbol{x}^i-\boldsymbol{x}^j) &= \int_{\mathbb{R}^d} p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}})\exp(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}'(\boldsymbol{x}^i-\boldsymbol{x}^j))d\bar{\boldsymbol{\alpha}}_{\mathbf{1}} \\
&= \int_{\mathbb{R}^d} p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}})\cos(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}'(\boldsymbol{x}^i-\boldsymbol{x}^j))d\bar{\boldsymbol{\alpha}}_{\mathbf{1}} \\
&= \int_{\mathbb{R}^d} p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}})(\cos(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}'\boldsymbol{x}^i)\cos(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}'\boldsymbol{x}^j)+ \\
&\sin(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}'\boldsymbol{x}^i)\sin(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}'\boldsymbol{x}^j))d\bar{\boldsymbol{\alpha}}_{\mathbf{1}} \\
&= E_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}[r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}(\boldsymbol{x}^i)r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}(\boldsymbol{x}^j)'], \qquad (7)
\end{aligned}$$

where the Fourier feature is

$$r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}(\boldsymbol{x}) = [\cos(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}'\boldsymbol{x}), \sin(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}'\boldsymbol{x})] \qquad (8)$$

and $\bar{\boldsymbol{\alpha}}_{\mathbf{1}}$ is sampled from the d dimensional multivariate Gaussian distribution $p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}}) = N(\mathbf{0}, 2g\boldsymbol{I})$ (which is the Fourier transform of the kernel in hand) with $E_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}$ being the corresponding expectation. From the first equation above to the second, we use that $p(\bar{\boldsymbol{\alpha}}_{\mathbf{1}})$ is real since the kernel is real and even. Hence, by replacing the expectation in (7) with the independent and identically distributed (i.i.d) sample mean of the ensemble $\{r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{q}}(\boldsymbol{x}^i)r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{q}}(\boldsymbol{x}^j)'\}_{q=1}^{D}$ of size D, we define our kernel mapping as

$$\begin{aligned}
\tilde{\boldsymbol{x}} &= \phi_{\boldsymbol{\alpha}_{\mathbf{1}}}(\boldsymbol{x}) \\
&= \sqrt{\frac{1}{D}}[r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{1}}(\boldsymbol{x}), r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{2}}(\boldsymbol{x}), \cdots, r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{D}}(\boldsymbol{x})]', \qquad (9)
\end{aligned}$$

which can be directly implemented in the hidden layer of the SLFN, cf. Fig. 1, along with the sinusoidal activation due to the definiton of $r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}}$.

Note that $\boldsymbol{\alpha_t}$ keeps all the hidden layer parameters at time t as a matrix of size $2D \times d$ consisting of $\bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{i}}$'s corresponding to the hidden units, i.e., $\boldsymbol{\alpha_t} = [\bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\mathbf{1}}, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\mathbf{1}}, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\mathbf{2}}, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\mathbf{2}}, \cdots, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{D}}, \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{D}}]'$. And the hidden layer activation is sinusoidal: $h_h(m) = \cos(m)$ and $h_h(m) = \sin(m)$ for the odd and even indexed hidden nodes, respectively, due to the definition in (8). At time $t = 1$, $\boldsymbol{\alpha_1}$ is randomly initialized with an appropriate g of the rbf kernel so that the SLFN starts with approximately constructing the high dimensional kernel space $\bar{\mathcal{H}} = \{f : f(\boldsymbol{x}) = h_o(h_h(\boldsymbol{\alpha}_1\boldsymbol{x})'\boldsymbol{w}+b), \forall\boldsymbol{w}, \forall b\}$ in its hidden layer, and in relation to (6), $\tilde{\boldsymbol{x}} = \phi_{\boldsymbol{\alpha_1}}(\boldsymbol{x}) = h_h(\boldsymbol{\alpha}_1\boldsymbol{x})$. Note that $\bar{\mathcal{H}}$ of the rbf kernel readily provides a powerful nonlinear modeling to the SLFN even if the hidden layer is kept untrained. Thanks to this excellent network initialization, we achieve an expedited process of learning from data. Moreover, in the course of our sequential processing, the SLFN continuously updates and improves the hidden layer, i.e., kernel mapping, parameters as $\boldsymbol{\alpha_t}$. Therefore, we optimize a nonlinear NP classifier in actually the larger space $\mathcal{H} \supset \bar{\mathcal{H}}$ (as our optimization is not restricted to $\boldsymbol{\alpha_1}$ of the random initialization, cf. the definition of $\mathcal{H}$ in (5)) for greater nonlinear modeling capability compared to the rbf kernel.

The SLFN in Fig. 1 that we use for online and nonlinear NP classification is compact in principle since the required number of hidden nodes is relatively small. The reason is that the convergence of the sample mean of the i.i.d. ensemble $\{r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{q}}(\boldsymbol{x}^i)r_{\bar{\boldsymbol{\alpha}}_{\mathbf{1}}^{q}}(\boldsymbol{x}^j)'\}_{q=1}^{D}$ of size D to the true mean $k(\boldsymbol{x}^i, \boldsymbol{x}^j)$ is exponentially fast with the order of $O(e^{-D})$ by Hoeffding's inequality [12]. On the other hand, since random Fourier features are independent of data, further compactification is possible by eliminating irrelevant, i.e., unuseful, Fourier features in a data driven manner, cf. the examples of feature selection in [35] and Nyström method in [46] for this purpose. In contrast, and alternatively, we distill useful Fourier features in the hidden layer activations as a result of the sequential learning of the kernel mapping parameters, i.e., $\bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{i}}$, via SGD. Hence, nodes of the SLFN are dedicated to only useful Fourier features, and thus we achieve a further network compactification by reducing the necessary number of hidden nodes as well as reducing the parameter complexity. Then, one can expect to better fight overfitting with great nonlinear modeling power and NP classification performance. This compactification does also significantly reduce the computational as well as space complexity of our SLFN based classifier, which -together with the SGD optimization- yields scalability to voluminous data. Consequently, the proposed online NP classifier is computationally highly efficient and appropriate for real time processing in large scale data applications.

Remark 1: We obtain a sequence of kernel mapping parameters $\boldsymbol{\alpha}_t$ in the course of data processing. This means that at the end of processing N instances, one can potentially construct a new non-isotropic rbf kernel by estimating the multivariate density of the collection $\{\bar{\boldsymbol{\alpha}}_N^j\}_{j=1}^{D}$ (here, we assume that D is large and the density is multivariate Gaussian. If it is not Gaussian, then one can straightforwardly incorporate a Gaussianity measure into the overall network objective) and then finding out the corresponding non-isotropic rbf kernel by taking back the inverse Fourier transform of the estimated density. Therefore, our algorithm is also kernel-adaptive since it essentially learns a new kernel (and also improves the previous one) at each SGD learning step. This kernel adaptation ability can be improved. For instance, one can start with a random mapping as described and estimate the density of the mapping parameters after convergence, and then re-start with new samples from the converged density. Multiple iterations of this process may yield better kernel adaptation (but re-running would hinder online processing and define batch processing, hence it is out of scope of the present work), which we consider as future work.

In the output layer of the SLFN, we use a certain variant of perceptron [45] with the identity activation, i.e., $h_o(m) = m$. Then, the classification model is defined linearly after the hidden layer kernel inspired transformation as $f(\boldsymbol{x}) = h_o(\langle\boldsymbol{w}, \tilde{\boldsymbol{x}}\rangle) + b = \langle\boldsymbol{w}, \tilde{\boldsymbol{x}}\rangle + b = h_h(\boldsymbol{\alpha x})'\boldsymbol{w} + b$, where $\boldsymbol{w} \in \mathbb{R}^{2D}$ is the normal vector to the linear separator and $b \in \mathbb{R}$ is the bias. Thus, the decision of the SLFN is $\hat{y} = \mathrm{sgn}(f(\boldsymbol{x}))$.

Regarding the overall network objective for sequential learning of the network parameters $\boldsymbol{\alpha}_t, \boldsymbol{w}_t, b_t$ and solving the NP optimization in (4) to obtain our SLFN based online nonlinear NP classifier, we next formulate the NP objective similar to [32] as

$$f^* = \arg\min_{f \in \mathcal{H}} \frac{\lambda}{2}||f||^2 + \hat{\text{P}}_{\text{nd}}(f) \quad \text{subject to } \hat{\text{P}}_{\text{fa}}(f) \leq \tau, \tag{10}$$

where the first term $\lambda/2||f||^2$ is the regularizer for which we use the magnitude of the classifier parameters in the output layer, i.e., $\lambda/2||\boldsymbol{w}||^2$, and λ is the regularization weight. For differentiability, the non-detection $\hat{\text{P}}_{\text{nd}}$ and false positive $\hat{\text{P}}_{\text{fa}}$ error rates are estimated based on data until time t as

$$\begin{aligned}\hat{\text{P}}_{\text{nd}}(f) &= \frac{1}{n_{t_+}} \sum_{t' \in S_1^t} l(f(\boldsymbol{x}_{t'})) \text{ and} \\ \hat{\text{P}}_{\text{fa}}(f) &= \frac{1}{n_{t_-}} \sum_{S_{-1}^t} l(-f(\boldsymbol{x}_{t'}))\end{aligned} \tag{11}$$

with $S_c^t = \{t' : 1 \leq t' \leq t, y_{t'} = c\}$, $n_{t_+} = |S_1^t|$ (set cardinality) and $n_{t_-} = |S_{-1}^t|$. Note that another appropriate function can be used here to obtain a differentiable surrogate for the $0-1$ errors in (4) for estimating the error rates. However, our results in the rest of this paper are based on the sigmoid loss $l(m) = 1/(1+\exp(m))$.

For sequential optimization of the NP objective in (10), we next define the following Lagrangian

$$L(f, \gamma) = \frac{\lambda}{2}||\boldsymbol{f}||^2 + \hat{\text{P}}_{\text{nd}}(f) + \gamma(\hat{\text{P}}_{\text{fa}}(f) - \tau), \tag{12}$$

where τ is the user-specified desired false positive rate and $\gamma \in \mathbb{R}^+$ is the corresponding Lagrange multiplier.

Since the saddle points of (12) correspond to the local minimum of (10), cf. [32] and [33] for the details, we apply the Uzawa approach [33] to search for the saddle points of (12) and learn our parameters in the online setting with SGD updates. To be more precise, we follow the optimization framework of [32] and solve the $\min\max$ optimization $f^* = \arg\min_f \max_\gamma L(f, \gamma)$ via an iterative approach with gradient steps, where one iteration minimizes $L(f, \gamma)$ for a fixed γ and the other maximizes $L(f, \gamma)$ for a fixed f. Note that the fixed-γ minimization

$$\begin{aligned}\arg\min_{f \in \mathcal{H}} L(f, \gamma) &= \arg\min_{f \in \mathcal{H}} \frac{\lambda}{2}||\boldsymbol{f}||^2 + \hat{\text{P}}_{\text{nd}}(f) + \gamma(\hat{\text{P}}_{\text{fa}}(f) - \tau) \\ &= \arg\min_{f \in \mathcal{H}} \frac{\lambda}{2}||\boldsymbol{f}||^2 + \hat{\text{P}}_{\text{nd}}(f) + \gamma\hat{\text{P}}_{\text{fa}}(f)\end{aligned}$$

is a regularized weighted error minimization, where the ratio of the type I error rate cost to the one of type II error rate is γ. Hence, the unknown Lagrange multiplier γ defines (up to a scaling with the prior probabilities) the asymmetrical error costs that correspond to the false positive rate constraint in (4). On the other hand, the gradient ascent updates $\gamma \leftarrow \gamma + \beta\nabla_\gamma L(f, \gamma) = \gamma + \beta(\hat{\text{P}}_{\text{fa}}(f) - \tau)$ in the fixed-f maximization determines the unknown multiplier γ so that the type I error cost is decreased (increased) if the error estimate is below (above) the tolerable rate τ in favor of detection power (true negative detection). This provides an iterative learning of the correspondence between the asymmetrical error costs and the NP constraint.

To this end, inserting the definitions in (11) and (11) into (12) with the regularization $\lambda/2||f||^2 = \lambda/2||\boldsymbol{w}||^2$ yields the overall SLFN objective as follows

$$\begin{aligned}L(f, \gamma) =& \frac{\lambda}{2}||\boldsymbol{w}||^2 + \frac{1}{n_{t_+}} \sum_{1 \leq t' \leq t : y_{t'} = 1} l\Big(y_{t'} f(\boldsymbol{x}_{t'})\Big) \\ &+ \frac{\gamma}{n_{t_-}} \sum_{1 \leq t' \leq t : y_{t'} = -1} l\Big(y_{t'} f(\boldsymbol{x}_{t'})\Big) - \gamma\tau \\ =& \frac{1}{t} \sum_{t'=1}^{t} \left(\frac{\lambda}{2}||\boldsymbol{w}||^2 + \mu_{t'} l\Big(y_{t'} f(\boldsymbol{x}_{t'})\Big) - \gamma\tau\right) \\ =& \frac{1}{t} \sum_{t'=1}^{t} s(f, \gamma, t'),\end{aligned} \tag{13}$$

where $s(f, \gamma, t') = \left(\lambda/2||\boldsymbol{w}||^2 + \mu_{t'} l\Big(y_{t'} f(\boldsymbol{x}_{t'})\Big) - \gamma\tau\right)$ and $\mu_{t'} = t/n_{t_+}$ if $y_{t'} = +1$, and $\gamma t/n_{t_-}$, otherwise.

In order to learn the SLFN parameters for obtaining the proposed online nonlinear NP classifier via the NP optimization explained above, we use stochastic gradient descent (SGD) to sequentially optimize the overall network objective defined in (13). These network parameters are 1) $\boldsymbol{\alpha}$, to project input $\boldsymbol{x}$ to the higher dimensional kernel space, 2) $\boldsymbol{w}$ and b, which are the perceptron parameters of the output layer to classify the projected input $\tilde{\boldsymbol{x}}$, and 3) γ, to learn the correspondence between the error costs and the NP constraint.

Suppose at the beginning of time t, we have an existing model f_t learned with the past data as well as the error costs corresponding to γ_t; and a little later, we observe the instance $\boldsymbol{x}_t$. SGD based optimization takes steps to update f_t and γ_t to obtain f_{t+1} and γ_{t+1} with respect to the partial derivatives of the instantaneous objective $s(f_t, \gamma_t, t)$. Namely, $f_{t+1} = f_t - \eta_t \nabla_f s(f_t, \gamma_t, t)$ and $\gamma_{t+1} = \gamma_t + \beta_t \nabla_\gamma s(f_t, \gamma_t, t)$. Based on the partial derivatives of the instantaneous objective $s(f_t, \gamma_t, t)$ defined in (13), the SGD updates for the SLFN parameters can be computed $\forall i \in \{1, \cdots, D\}$ as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t\Big(\lambda\boldsymbol{w}_t + \mu_t \nabla_{\boldsymbol{w}} l\big(y_t f_t(\boldsymbol{x}_t)\big)\Big)$, $b_{t+1} = b_t - \eta_t\Big(\mu_t \nabla_b l\big(y_t f_t(\boldsymbol{x}_t)\big)\Big)$, $\bar{\boldsymbol{\alpha}}_{\boldsymbol{t+1}}^{\boldsymbol{i}} = \bar{\boldsymbol{\alpha}}_{\boldsymbol{t}}^{\boldsymbol{i}} - \eta_t\Big(\mu_t \nabla_{\bar{\boldsymbol{\alpha}}^{\boldsymbol{i}}} l\big(y_t f_t(\boldsymbol{x}_t)\big)\Big)$, and $\gamma_{t+1} = \gamma_t + \beta_t\Big((1_{\{y_t=-1\}} t/n_{t_-}) l\big(y_t f_t(\boldsymbol{x}_t)\big) - \tau\Big)$, where η_t is the learning rate and β_t is named as the Uzawa gain [33] controlling the learning rate of the Lagrange multiplier. Using the sigmoid $l(m) = 1/(1+\exp(m))$ yields the partial derivatives with $\tilde{\boldsymbol{x}}_t = h_h(\boldsymbol{\alpha}_t \boldsymbol{x}_t)$ as

$$\nabla_{\boldsymbol{w}} l\big(y_t f_t(\boldsymbol{x}_t)\big) = -\tilde{\boldsymbol{x}}_t l^2(y_t f_t(\boldsymbol{x}_t)) \exp(y_t f_t(\boldsymbol{x}_t)) y_t, \tag{14}$$

$$\nabla_b l\big(y_t f_t(\boldsymbol{x}_t)\big) = -l^2(y_t f_t(\boldsymbol{x}_t)) \exp(y_t f_t(\boldsymbol{x}_t)) y_t, \text{ and} \tag{15}$$

$$\nabla_{\bar{\boldsymbol{\alpha}}^i} l\big(y_t f_t(\boldsymbol{x}_t)\big) = -\boldsymbol{x}_t l^2(y_t f_t(\boldsymbol{x}_t)) \exp(y_t f_t(\boldsymbol{x}_t)) y_t \tag{16}$$
$$\times \big(- w_t^{2i-1} \sin(\bar{\boldsymbol{\alpha}}_t^{i\prime} \boldsymbol{x}_t) + w_t^{2i} \cos(\bar{\boldsymbol{\alpha}}_t^{i\prime} \boldsymbol{x}_t)\big),$$

which can be straightforwardly incorporated into the back-propagation.

In our experiments, we obtain an empirical false positive rate estimate $\hat{\mathrm{P}}_{\text{fa}}$ based on a sliding window keeping the $0-1$ errors for a couple hundreds of the past negative data instances, and use the following γ update instead of the aforementioned stochastic one:

$$\gamma_{t+1} = \gamma_t \bigg(1 + \beta_t \Big(\hat{\mathrm{P}}_{\text{fa}} - \tau\Big)\bigg), \text{ when } y_t = -1, \tag{17}$$

which has been observed to yield a more stable and robust performance. Note that this update is directly resulted from (12), and does certainly not disturb real-time online processing since a past window of positive decisions requires almost no additional space complexity (only 200 bits in the case of, for instance, storing binary decisions for 200 past negative instances).

Based on the derivations above, we sequentially update the SLFN at each time in a truly online manner with $O(N)$ (here, N: total number of processed data instances) computational and $O(1)$ space complexity in accordance with the NP objective. Hence, we construct our method called "NP-NN" in Algorithm 1 that can be used in real time for online nonlinear Neyman-Pearson classification. We refer to the Section V of our experimental study for all the details about the input parameters and initializations.

Remark 2: Recall that the goal in NP classification is to achieve the minimum miss rate (maximum detection power) while upper bounding the false positive rate (FPR) by a user-specified threshold τ. Therefore, both aspects (minimum miss rate and its FPR constraint) of this goal should be considered in evaluating the performance of NP classifiers. The NP-score of [4], [47] is defined as

$$\text{NP-score} = \kappa \max(\hat{P}_{\text{fa}}(f) - \tau, 0) + \hat{P}_{\text{nd}}(f), \tag{18}$$

where f is the NP model to be evaluated and κ controls the relative weights of the miss rate (with weight 1) and its FPR constraint (with weight κ if the desired rate is exceeded, and with weight 0 otherwise). Namely, κ controls the hardness of the NP FPR constraint, and a smaller NP-score indicates a better NP classifier. By enforcing a strict hard constraint on FPR with a very large $\kappa \simeq \infty$, one can immediately reject models (while evaluating various models) that violate FPR constraint with even a slight positive deviation from the desired FPR τ (a negative deviation does not violate). However, even though the original NP formulation requires

Algorithm 1 Proposed Online Nonlinear Neyman-Pearson Classifier (NP-NN)

```
1:  Set the desired (or target) false positive rate (TFPR) $\tau$, regularization $\lambda$, number $2D$ of hidden nodes and bandwidth $g$ for the rbf kernel
2:  Initialize the SLFN parameters $\boldsymbol{\alpha}_1$, $\boldsymbol{w}_1$, $b_1$, and learning rates $\eta_1$, $\beta_1$, $\gamma_1$
3:  Set $n_{t_+} = n_{t_-} = 0$, and sliding window size $W_s = 200$
4:  for $t = 1, 2, \ldots$ do
5:      Receive $\boldsymbol{x}_t$ and calculate $\tilde{\boldsymbol{x}}_t = h_h(\boldsymbol{\alpha}_t \boldsymbol{x}_t)$ and $f_t(\boldsymbol{x}_t) = \boldsymbol{w}_t' \tilde{\boldsymbol{x}}_t + b_t$
6:      Calculate the current decision as $\hat{y}_t = \mathrm{sgn}(f_t(\boldsymbol{x}_t))$ and observe $y_t$
7:      Calculate $n_{t_+} = n_{t_+} + 1_{\{y_t=1\}}$ and $n_{t_-} = n_{t_-} + 1_{\{y_t=-1\}}$
8:      Calculate $\mu_t = t/n_{t_+} 1_{\{y_t=1\}} + \gamma t/n_{t_-} 1_{\{y_t=-1\}}$
9:      Update $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \Big(\lambda \boldsymbol{w}_t + \mu_t \nabla_{\boldsymbol{w}} l\big(y_t f_t(\boldsymbol{x}_t)\big)\Big)$, cf. (14)
10:     Update $b_{t+1} = b_t - \eta_t \Big(\mu_t \nabla_b l\big(y_t f_t(\boldsymbol{x}_t)\big)\Big)$, cf. (15)
11:     Update $\bar{\boldsymbol{\alpha}}_{t+1}^i = \bar{\boldsymbol{\alpha}}_t^i - \eta_t \Big(\mu_t \nabla_{\bar{\boldsymbol{\alpha}}^i} l\big(y_t f_t(\boldsymbol{x}_t)\big)\Big)$, cf. (16)
12:     Update $\gamma_{t+1} = \gamma_t \bigg(1 + \beta_t \Big(\hat{\mathrm{P}}_{\text{fa}} - \tau\Big)\bigg)$, if $y_t = -1$, cf. (17) and the explanation about the estimate $\hat{\mathrm{P}}_{\text{fa}}$ of the false positive rate
13:     Update $\eta_{t+1} = \eta_1 (1+\lambda t)^{-1}$ and $\beta_{t+1} = \beta_1 (1+\lambda t)^{-1}$
14: end for
```

a hard constraint, we consider that it is not appropriate to use a hard constraint in practice, as also extensively explained in [47], based on the following two reasons: (1) An NP classifier is typically learned using a set of observations, and that set is itself a random sample from the underlying density of the data. Hence, the estimated FPR $\hat{P}_{\text{fa}}(f)$ of the model is also a random quantity, which is merely an estimator of the unknown true FPR $P_{\text{fa}}(f)$. Note that the true FPR $P_{\text{fa}}(f)$ is actually the one to be strictly constrained, but unavailable. Thus, it is unreliable to enforce a strict hard constraint (with a very large $\kappa \simeq \infty$) on the random estimator $\hat{P}_{\text{fa}}(f)$, and a relatively soft constraint has surely more practical value by allowing a small positive deviation from the desired FPR τ. (2) Also, one might be willing to exchange true negatives in favor of detections with a small positive deviation from the desired FPR τ, when the gain is larger than the loss as the NP-score improves. Consequently, for parameter selections with cross validation in our algorithm design as well as for performance evaluations in our experiments, we opt for a relatively soft constraint and use $\kappa = 1/\tau$ in accordance with the recommendation by the authors [47]. This choice allows a relatively small positive deviation from the desired FPR, and normalizes the deviation by measuring it in a relative percentage manner. For example, the positive deviations 0.1 and 0.001 both degrade the score equally by 50% when the desired rates are 0.2 and 0.002, respectively. Various other

NP studies in the literature do also practically allow small positive deviations from the desired FPR τ. For instance, we observe such a deviation in [32] with theirs and compared algorithms [4] in the case of spambase dataset, in [31] with theirs in the case of heart and breast cancer datasets, and finally in [26] with one of the compared algorithms [48] in all datasets.

A comprehensive experimental evaluation of our proposed technique is next provided based on real as well as synthetic datasets in comparison to state-of-the-art competing methods.

V. EXPERIMENTS

We present extensive comparisons of the proposed kernel inspired SLFN for online nonlinear Neyman-Pearson classification (NP-NN), described in Algorithm 1, with 3 different state-of-the-art NP classifiers. These compared techniques are online linear NP (OLNP) [32], as well as logistic regression (NPROC-LOG) [27] and support vector machines with rbf kernel (NPROC-SVM) [49] in the NP framework of the umbrella algorithm described in [28]. Among these, OLNP (linear NP classification) is an online technique with $O(N)$ computational complexity, whereas NPROC-LOG (linear NP classification) and NPROC-SVM (nonlinear NP classification) are batch techniques with at least $O(N^2)$ computational complexity, where N is the number of processed instances. In contrast, we emphasize that to our best knowledge, the proposed NP classifier NP-NN is both nonlinear and online as the first time in literature, with $O(N)$ computational and negligible space complexity resulting real time nonlinear NP modeling and false positive rate controllability. Consequently, the proposed NP-NN is appropriate for challenging fast streaming data applications.

Since our proposed algorithm NP-NN is the first online and nonlinear NP classifier, there is technically no fully comparable algorithm in the literature. Nevertheless, we set our experiments in the fairest manner by considering comparisons among all possibilities: batch nonlinear (NPROC-SVM), batch linear (NPROC-LOG), online nonlinear (our proposed NP-NN) and online linear (OLNP). Although we compare with NPROC-LOG, it is not particularly strong in terms of efficiency with small computational and space complexity, and it is also not particularly strong in terms of nonlinear modeling capability. For this reason, we mainly concentrate on comparisons to NPROC-SVM in terms of the classification performance, and on comparisons to OLNP in terms of the complexity. NPROC-SVM is extremely powerful and well-known with regards to nonlinear modeling, whereas OLNP is extremely powerful with regards to efficiency both computationally and space-wise. Otherwise, there is no algorithm (except our proposed online and nonlinear algorithm NP-NN) in the literature which is powerful in terms both nonlinear modeling and efficiency. In order to further ensure fairness in our experiments, we pay special attention to the followings. 1) Through extensive cross validations, we optimize the parameters for each algorithm and for each dataset separately in their respective contexts. 2) We run the algorithms on the same exact sequence in each case of our datasets, which is particularly important while comparing online algorithms (our algorithm NP-NN and OLNP). 3) Note that a specific data sequence can favor one algorithm by luck. In order to remove this dependency on sequence order, we run the algorithms on 10 different random permutations (all algorithms are run on the same exact set of randomly permuted data sequences) and report the mean performance along with the standard deviations. 4) We do not rely on a specific performance measure. Instead, we report the performance in terms of the area under the ROC (receiver operating characteristics) curve (AUC) as well as the NP-scores for each target false alarm rate separately. Furthermore, we also report the achieved false alarm rate and the true positive rate. 5) The proposed algorithm NP-NN and the OLNP are online algorithms and thus they do not have separate training and test phases. However, NPROC-SVM is a batch algorithm with separate training and test phases. Hence, to ensure fairness, we use separate training and test phases while comparing with NPROC-SVM (although it is certainly NOT needed for OLNP and the proposed NP-NN). Otherwise, while comparing the proposed online algorithm NP-NN with OLNP on large scale datasets, we test them (NP-NN and OLNP) in the online data processing framework (without separate training and test) that they (NP-NN and OLNP) are originally designed for. 6) We present detailed statistical significance (for fairly evaluating the performance differences) and complexity analyses (for fairly evaluating the complexity and running time differences) in the end as two separate sections. After providing this summary and important remarks about our experimental paradigm, next, we continue with the details and present our results.

We conduct experiments based on various real and synthetic datasets [50], [51] from several fields such as bioinformatics and computer vision, each of which is normalized by either unit-norm (each instance is divided by its magnitude) or z-score (each feature is brought down to zero mean unit variance) normalization before processing. For each dataset, smaller class is designated as the positive (target) class. The details of the datasets are provided in Table 1, where the starred ones and unstarred ones are normalized with unit norm and z-score, respectively. For performance evaluations, we generate 15 random permutations of each dataset, and each random permutation is split into two as training $(\%75)$ and test $(\%25)$ sequences. We strongly emphasize that the processing in the proposed algorithm NP-NN is truly online, meaning that, there are no separate training and test phases. However, since NPROC-LOG and NPROC-SVM are batch algorithms requiring a separate training, we opt to use training/test splits in this first set of experiments for a fair and statistically unbiased robust performance comparison. Such a split is in fact not needed in practice in the case of the proposed NP-NN that -by design- processes data on the fly. Additional experiments based on two larger scale datasets to demonstrate the ideal use-case (i.e. online processing without

TABLE 1: We present performance results of the proposed algorithm NP-NN and the competing algorithms OLNP and NPROC-SVM for each targeted false positive rate (TFPR $\in \{0.05, 0.1, 0.2, 0.3, 0.4\}$) on the datasets in the leftmost column. In each case, we run the algorithms 15 times over 15 different permutations of the dataset, where 75% (25%) is used for training (testing). Therefore, average of the test results are presented with the corresponding standard deviation. For each dataset, the first row is the achieved true positive rate (TPR), the second row is the achieved false positive rate (FPR), the third row is the NP-score, and the fourth row is the area under curve (AUC) of the receiver operating characteristics (ROC) curve of TPR vs TFPR. The best and the second best performing algorithms are signified with fonts in bold style. Overall, we observe that A) the proposed NP-NN and NPROC-SVM outperform (due to their nonlinear modeling) OLNP (based on its results in Fig. 2), B) the proposed NP-NN and NPROC-SVM perform comparably in terms of AUC, and NPROC-SVM performs better in terms of NP-score, where the advantage of NPROC-SVM in terms of NP-score seems to disappear as the data size and/or TFPR increase, and C) the proposed NP-NN with online and real-time processing capabilities at $O(N)$ computational and negligible $O(1)$ space complexity has huge advantages over the competing NPROC-SVM in the case of contemporary large scale fast streaming data applications. Namely, when one has a fast streaming dataset of size in the order of millions or more, then the only choice of high performance in real time is the proposed NP-NN, cf. Fig. 4.

Dataset	OLNP					NPROC-SVM					NP-NN				
(n_-, n_+, d)	**TFPR**$(\tau)\rightarrow$ 0.05	0.1	0.2	0.3	0.4	0.05	0.1	0.2	0.3	0.4	0.05	0.1	0.2	0.3	0.4
Banana	**TPR**$\rightarrow$ 0.177±0.017	0.213±0.014	0.299±0.038	0.444±0.025	0.546±0.032	0.773±0.095	0.884±0.017	0.948±0.012	0.978±0.006	0.994±0.004	0.749±0.188	0.894±0.007	0.946±0.01	0.979±0.005	0.991±0.003
(2924, 2376, 2)	**FPR**$\rightarrow$ 0.052±0.012	0.097±0.015	0.197±0.032	0.309±0.032	0.407±0.023	0.038±0.008	0.078±0.012	0.176±0.017	0.276±0.016	0.383±0.02	0.053±0.006	0.098±0.012	0.2±0.017	0.298±0.016	0.392±0.015
	NP-score$\rightarrow$ 0.935±0.114	0.815±0.064	0.752±0.085	0.613±0.056	0.487±0.026	**0.231**±0.092	**0.116**±0.017	**0.053**±0.012	**0.022**±0.004	**0.011**±0.012	**0.325**±0.179	**0.148**±0.068	**0.089**±0.51	**0.039**±0.031	**0.016**±0.014
	AUC$\rightarrow$0.128±0.012					**0.347**±0.006					**0.347**±0.009				
Breast*	0.908±0.039	0.949±0.025	0.987±0.018	0.985±0.017	0.996±0.008	0.936±0.032	0.974±0.024	0.987±0.013	0.994±0.009	0.994±0.009	0.958±0.026	0.972±0.024	0.985±0.019	0.991±0.01	0.998±0.006
(357, 212, 30)	0.06±0.023	0.108±0.025	0.196±0.043	0.294±0.057	0.401±0.054	0.021±0.014	0.057±0.028	0.13±0.036	0.226±0.054	0.36±0.041	0.053±0.027	0.099±0.027	0.19±0.034	0.273±0.026	0.397±0.04
	0.391±0.324	0.193±0.175	0.085±0.15	0.073±0.119	0.056±0.09	**0.077**±0.054	**0.039**±0.043	**0.013**±0.013	**0.011**±0.016	**0.016**±0.03	**0.261**±0.376	**0.113**±0.205	**0.051**±0.106	**0.015**±0.016	**0.037**±0.057
	0.363±0.005					**0.368**±0.005					**0.368**±0.005				
Pageblocks	0.833±0.015	0.905±0.015	0.955±0.013	0.979±0.011	0.981±0.015	0.925±0.02	0.954±0.025	0.981±0.013	0.992±0.011	0.996±0.005	0.926±0.015	0.967±0.014	0.982±0.012	0.993±0.007	0.994±0.005
(4913, 560, 10)	0.06±0.008	0.095±0.012	0.195±0.11	0.295±0.016	0.397±0.012	0.042±0.009	0.096±0.011	0.188±0.014	0.288±0.016	0.387±0.009	0.052±0.008	0.088±0.017	0.176±0.019	0.283±0.018	0.402±0.03
	0.375±0.16	0.122±0.058	0.055±0.023	0.034±0.026	**0.026**±0.022	**0.09**±0.028	**0.075**±0.059	**0.027**±0.037	**0.013**±0.018	**0.004**±0.005	**0.159**±0.081	**0.061**±0.074	**0.021**±0.015	**0.013**±0.015	0.04±0.05
	0.352±0.004					**0.366**±0.003					**0.366**±0.003				
Bupaliver*	0.311±0.063	0.428±0.1	0.497±0.079	0.558±0.061	0.658±0.051	0±0	0.242±0.082	0.444±0.083	0.603±0.119	0.706±0.096	0.369±0.091	0.456±0.073	0.544±0.087	0.661±0.029	0.739±0.094
(200, 145, 6)	0.064±0.035	0.13±0.063	0.216±0.091	0.3±0.066	0.428±0.074	0±0	0.042±0.048	0.114±0.069	0.2±0.059	0.3±0.086	0.094±0.048	0.12±0.059	0.21±0.061	0.366±0.035	0.442±0.076
	1.129±0.481	1.012±0.413	0.723±0.332	**0.535**±0.132	0.462±0.108	**1**±0	**0.778**±0.073	**0.576**±0.092	**0.397**±0.119	**0.299**±0.089	1.531±0.909	**0.904**±0.365	**0.606**±0.215	0.559±0.125	**0.396**±0.134
	0.187±0.022					0.156±0.026					**0.21**±0.024				
Cod-rna	0.915±0.002	0.977±0.002	0.995±0.001	0.998±0	0.999±0	0.959±0.003	0.983±0.02	0.997±0.003	0.999±0	1±0	0.938±0.002	0.979±0.002	0.995±0.001	0.998±0.001	0.999±0
(36690, 19845, 8)	0.05±0.002	0.097±0.004	0.196±0.004	0.294±0.005	0.395±0.007	0.047±0.003	0.08±0.024	0.145±0.053	0.222±0.051	0.294±0.005	0.054±0.002	0.09±0.003	0.187±0.004	0.292±0.008	0.393±0.006
	0.102±0.026	0.029±0.008	0.006±0.004	**0.003**±0.004	**0.002**±0.003	**0.046**±0.015	**0.017**±0.02	**0.005**±0.005	**0.001**±0	**0**±0	0.14±0.042	**0.021**±0.002	**0.005**±0.001	0.004±0.005	**0.002**±0.004
	0.368±0					**0.371**±0.001					**0.37**±0				
Pen Digits*	0.975±0.015	0.987±0.01	0.996±0.005	0.998±0.002	0.999±0.002	1±0	1±0	1±0	1±0	1±0	0.993±0.005	0.996±0.003	0.999±0.002	0.997±0.003	0.998±0.002
(6714, 780, 16)	0.049±0.009	0.095±0.014	0.202±0.014	0.294±0.014	0.401±0.018	0.04±0.007	0.088±0.006	0.186±0.009	0.279±0.018	0.382±0.011	0.032±0.008	0.088±0.012	0.188±0.015	0.292±0.013	0.393±0.011
	0.084±0.1	0.043±0.05	0.038±0.036	**0.009**±0.016	0.02±0.023	**0.001**±0.004	**0**±0	**0**±0	**0.001**±0.002	**0**±0	**0.007**±0.005	**0.009**±0.014	**0.01**±0.026	**0.009**±0.011	**0.005**±0.009
	0.372±0.002					**0.375**±0					**0.374**±0.001				
Spiral	0.34±0.066	0.336±0.097	0.328±0.088	0.332±0.063	0.516±0.289	0±0	1±0	1±0	1±0	1±0	1±0	0.98±0.063	0.996±0.013	0.996±0.013	1±0
(211, 101, 2)	0.056±0.034	0.125±0.051	0.229±0.075	0.31±0.07	0.463±0.232	0±0	0.052±0.048	0.085±0.051	0.213±0.069	0.335±0.058	0.019±0.024	0.11±0.043	0.229±0.058	0.348±0.071	0.45±0.09
	0.983±0.475	1.002±0.437	0.905±0.266	0.786±0.132	0.793±0.185	1±0	**0.069**±0.172	**0**±0	**0.015**±0.049	**0.015**±0.049	**0.054**±0.17	**0.235**±0.309	**0.222**±0.188	**0.204**±0.174	**0.159**±0.191
	0.141±0.038					**0.325**±0					**0.373**±0.006				
Iris	0.355±0.1	0.455±0.214	0.727±0.086	0.664±0.332	0.955±0.064	0±0	0±0	1±0	1±0	1±0	0.964±0.047	0.982±0.038	1±0	1±0	1±0
(100, 50, 4)	0.116±0.076	0.148±0.082	0.248±0.082	0.316±0.15	0.428±0.15	0±0	0±0	0.096±0.097	0.168±0.106	0.24±0.075	0.1±0.074	0.148±0.087	0.244±0.104	0.308±0.136	0.404±0.106
	2.045±1.386	1.165±0.524	0.593±0.277	0.55±0.397	0.245±0.212	**1**±0	**1**±0	**0.06**±0.19	**0.02**±0.063	**0**±0	**1.136**±1.362	**0.618**±0.735	**0.34**±0.366	**0.2**±0.276	**0.11**±0.173
	0.234±0.048					**0.249**±0.003					**0.37**±0.008				
Svmguide1*	0.606±0.011	0.68±0.012	0.776±0.008	0.831±0.009	0.875±0.009	0.697±0.07	0.852±0.011	0.924±0.007	0.951±0.005	0.965±0.004	0.762±0.022	0.82±0.028	0.884±0.017	0.92±0.017	0.947±0.014
(4000, 3089, 4)	0.051±0.007	0.102±0.01	0.204±0.015	0.299±0.013	0.4±0.015	0.039±0.009	0.082±0.01	0.176±0.021	0.277±0.019	0.371±0.019	0.053±0.006	0.104±0.012	0.204±0.014	0.301±0.02	0.395±0.024
	0.446±0.068	0.369±0.064	0.267±0.045	0.186±0.022	0.141±0.023	**0.319**±0.06	**0.148**±0.011	**0.077**±0.008	**0.052**±0.013	**0.035**±0.005	**0.312**±0.084	**0.245**±0.08	**0.154**±0.049	**0.107**±0.043	**0.072**±0.021
	0.285±0.003					**0.334**±0.004					**0.326**±0.006				
Fourclass1	0.463±0.063	0.516±0.044	0.609±0.054	0.829±0.036	0.912±0.023	1±0	1±0	1±0	1±0	1±0	1±0	1±0	0.986±0.046	1±0	1±0
(555, 307, 2)	0.056±0.023	0.13±0.028	0.233±0.027	0.312±0.024	0.423±0.045	0.033±0.03	0.066±0.027	0.144±0.035	0.254±0.041	0.366±0.034	0.021±0.017	0.065±0.012	0.163±0.029	0.285±0.049	0.353±0.029
	0.75±0.394	0.807±0.238	0.557±0.119	0.225±0.065	0.169±0.087	**0.133**±0.329	**0.009**±0.027	**0.005**±0.016	**0.013**±0.029	**0.003**±0.01	**0.03**±0.096	**0**±0	**0.017**±0.045	**0.043**±0.09	**0.001**±0.005
	0.252±0.015					**0.375**±0					**0.372**±0.006				

separate training/tests phases) of the proposed algorithm NP-NN are presented in Fig. 4.

The rbf kernel bandwidth parameter g (for the proposed NP-NN as well as NPROC-SVM), the error cost parameter C (for NPROC-SVM) and the number $2D$ of hidden nodes (for the SLFN in the proposed NP-NN) are all 3-fold cross-validated (based on NP-score) for each random permutation using the corresponding training sequence by a grid search with $g \in \{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$, $C \in \{0.1, 1, 2, 4\}$ and $D \in \{2, 5, 10, 20, 40, 80, 100\} \times d$, where d is the data dimension. As the regularization has been observed to help little, we opt to use $\lambda \sim 0$ along with SGD learning updates $\eta_t = 0.01$ and $0.1 \geq \beta_t/\eta_t \geq 0.01$, randomly initialized $\boldsymbol{w}_1$ and b_1 (around 0) and $\gamma_1 = 1$ for both the proposed NP-NN and OLNP uniformly in all of our experiments. We directly use the code provided by the authors [28] for NPROC-LOG and NPROC-SVM and also optimize it by the aforementioned cross validation in terms of parameter selection. We observe that for the datasets of relatively short length, algorithms using SGD optimization, i.e., OLNP and NP-NN, improve with multiple passes over the training sequence. Hence, the length of the training sequence of each random permutation is increased by concatenation with additional randomizations for only OLNP and NP-NN (not for NPROC-LOG and NPROC-SVM) during training of both the cross-validation and actual training, resulting in an epoch-by-epoch training procedure. This concatenation is only for training purposes, and hence it is not used in testing and validation, i.e., the actual data size is used in all types of testing to avoid statistical bias and multiple counting. Our proposed algorithm NP-NN does certainly not need such a concatenation approach for data augmentation in the targeted fast streaming data applications (cf. Fig. 4), where data is already abundant and scarcity is not an issue.

We run all the algorithms on the test sequence of each of the 15 random permutations (after training on the corresponding training sequences), and record in each case the achieved false positive rate, i.e., FPR, and true detection rate, i.e., TPR, for the target false positive rates (TFPR) $\tau \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. For performance evaluation, we compare the mean area under curve (AUC) of the resulting 15 receiver operating characteristics (ROC) curves of TFPR vs TPR, as well as the mean of the resulting 15 NP-scores [47], cf. (18) with $\kappa = 1/\tau$. Note that the mean AUC (higher is better) accounts only for the resulting detection power without regard to false positive rate tractability, whereas the mean NP-score (lower is better) provides an overall combined measure. We evaluate with the both (Table 1) in addition to visual presentation of the mean ROC curves (Fig. 2) of FPR and TPR. Table 1 additionally reports the mean TPRs and mean FPRs. We also provide the decision boundaries and the mean convergence of the achieved false positive rate during training for the visually presentable 2-dimensional Banana dataset (Fig. 3). All of our results are provided with the corresponding standard deviations.

We exclude the results of NPROC-LOG in Table 1 (instead we keep NPROC-SVM since it generally performs better than NPROC-LOG) due to the page limitation, as the table gets too wide otherwise. One can access the results of NPROC-LOG from our Fig. 2. Based on our detailed analysis presented in Table 1 along with the visualization with ROC curves in Fig. 2, we first conclude that in general the algorithms NPROC-SVM and the proposed NP-NN with powerful nonlinear classification capabilities significantly outperform the linear algorithms OLNP and NPROC-LOG in terms of both AUC and NP-score, hence the proposed NP-NN and NPROC-SVM better address the need for modeling complex decision boundaries in the contemporary applications. This significant performance difference in favor of nonlinear algorithms NPROC-SVM and the proposed NP-NN is much more clear (especially in terms of the AUC) in highly nonlinear datasets such as Banana, Spiral, Iris, SVMguide1 and Fourclass, as shown in Fig. 2. In the case of a small size dataset that seems linear or less nonlinear (e.g., Bupaliver), although OLNP and NPROC-LOG are both linear by design and targeting this dataset with the right complexity and hence expected to be less affected by overfitting, the proposed NP-NN competes with the both well and even slightly outperforms them in terms of AUC (while staying comparable in terms of NP-score). We consider that this is most probably due to the successful compactification of the SLFN in the proposed NP-NN which reduces the parameter complexity by learning the Fourier features in the hidden layer.

As for the comparison between the nonlinear algorithms NPROC-SVM and the proposed NP-NN, we first strongly emphasize that NPROC-SVM has computational complexity (in the worst case of full number of support vectors) between $O(N^2)$ and $O(N^3)$ in training and $O(N)$ in test, where the space complexity is $O(N)$. On the other hand, the proposed NP-NN is truly online without separate training or test phases, which only requires $O(N)$ computational and $O(1)$ negligible space complexity. Hence, NPROC-SVM cannot be applied in our targeted large scale data processing applications due to its prohibitive complexity; nevertheless, we opt to include it in our experiments to set a baseline that is achievable by batch processing. According to the numeric results in Table 1 and the ROC curves in Fig. 2, we first observe that NPROC-SVM and the proposed NP-NN perform comparably in terms of the AUC, hence our technique (thanks to its computationally highly efficient implementation) can be used in large scale applications (where NPROC-SVM computationally fails) without a loss in classification performance. In addition, our algorithm NP-NN outperforms NPROC-SVM in terms of AUC in 3 datasets; and for small target false positive rate ($\tau = 0.05$), the proposed NP-NN has higher TPR compared to NPROC-SVM in 6 datasets. On the other hand, comparing in terms of the NP-score, NPROC-SVM performs better as a result of enhanced false positive rate controllability due to batch processing. However, this advantage of NPROC-SVM over the proposed NP-NN seems to disappear or decrease as the data size (relative to the

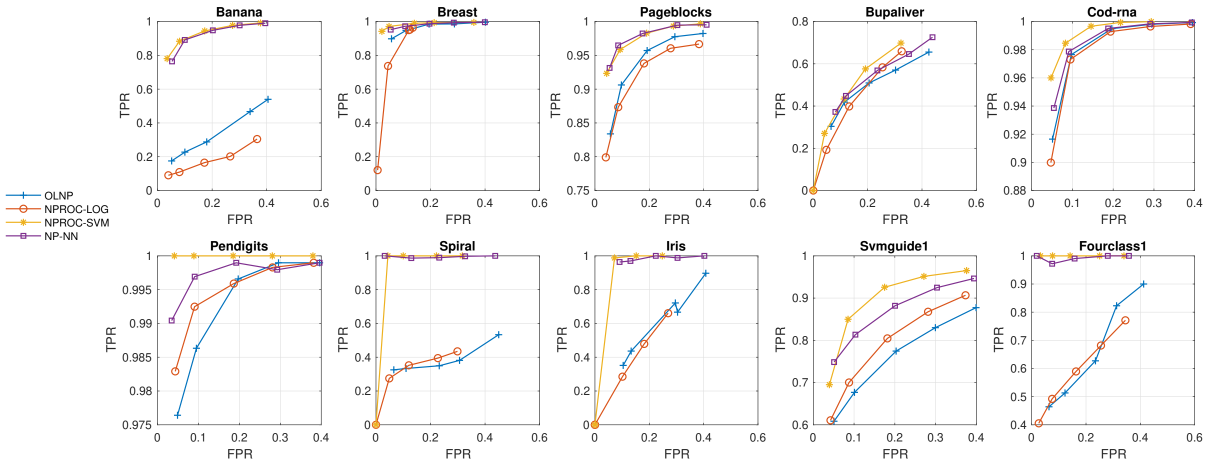

FIGURE 2: Visual presentations of the results in Table 1 are provided via the receiver operating characteristics (ROC) curves for all compared algorithms of OLNP, NPROC-LOG, NPROC-SVM and the proposed NP-NN, based on the achieved true positive rates and achieved false positive rates, i.e., TPR vs FPR, corresponding to the targeted false positive rates TFPR $\in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. Note that the presented ROC curves (TPR vs FPR) are mean curves over 15 trials of random data permutations for which the standard deviations can be followed from Table 1. Overall, in terms of the area under ROC (AUC), we observe that the proposed NP-NN and NPROC-SVM (due to their nonlinear modeling) outperform the other two. On the other hand, the proposed NP-NN performs similarly with NPROC-SVM while providing significant computational advantages. For the quantification of the false positive rate tractability alone, AUC alone and both as a combined measure, we refer to the results in Fig. 3, the AUC scores in Table 1 and the NP-scores in Table 1, respectively.

dimension) and/or the desired false positive rate increases as observed in the cases of, for instance, Banana and Cod-rna datasets. Therefore, we expect no loss (compared to NPROC-SVM) with the proposed NP-NN in terms of false positive rate controllability as well, when data size increases as in the targeted scenario of the big data applications where NPROC-SVM cannot be used. Indeed, we observe a decent nonlinear classification performance and false positive rate controllability with the proposed NP-NN on, for example, the Banana dataset (5300 instances in only 2 dimensions), as clearly visualized in Fig. 3 which shows the false positive rate convergence as well as the nonlinear decision boundaries for various desired false positive rates. Lastly, NPROC-SVM seems to be failing when TFPR requires only a few mistakes in the non-target class. In this case, NPROC-SVM picks zero mistake resulting in zero TPR and a poor NP-score in return. In contrast, the propsed NP-NN successfully handles such situations as demonstrated by, for instance, TFPR=0.05 at Iris dataset in Table 1.

Our experiments in Table 1, Fig. 2 and Fig. 3 include comparisons of the proposed NP-NN with certain batch processing techniques (NPROC-SVM and NPROC-LOG). Hence, we utilize separate training and test phases, along with multiple passes over training sequences (due to small sized datasets in certain cases such as Iris), in those experiments for statistical fairness. However, we emphasize that in the targeted scenario of large scale data applications: 1) one can only use computationally scalable online (such as the proposed NP-NN and OLNP) algorithms, 2) multiple passes are not necessary as the data is abundant, and also 3) one can target for even smaller false positive rates such as 0.01 and 0.005. Therefore, to better address this scenario of large scale data streaming conditions, we conduct additional experiments to compare the online methods (OLNP and the proposed NP-NN) when processing 2 large datasets (after z-score normalization and 15 random permutations) on the fly without separate training and test phases based on just a single pass: covertype (581012 instances in 54 dimensions) and Cod-rna (488565 instances in 8 dimensions, this is the original full scale, for which we previously use in Table 1 a relatively small subset for testing the batch algorithms). We run for τ (TFPR) $\in \{0.005, 0.01\}$ and present the resulting TPR and FPR at each time (in a time-accumulated manner after averaging over 15 random permutations) in Fig. 4. Parameters are set with manual inspection based on a small fraction of the data.

Although the false positive rate constraint is set harder (i.e. smaller as τ (TFPR) $\in \{0.005, 0.01\}$) in this experiment (compared to the smallest TFPR value 0.05 in Table 1), both techniques (OLNP and the proposed NP-NN) successfully converge (the proposed NP-NN appears to converge slightly better) to the target rate (FPR $\rightarrow$ TFPR) uniformly in all cases. Therefore, both techniques promise decent false positive rate controllability (almost perfect) when the data is sufficient. On the other hand, the proposed NP-NN strongly outperforms OLNP in terms of the TPR (again uniformly in all cases), which proves the gain due to nonlinear modeling in the proposed NP-NN. In terms of the NP-score, the proposed

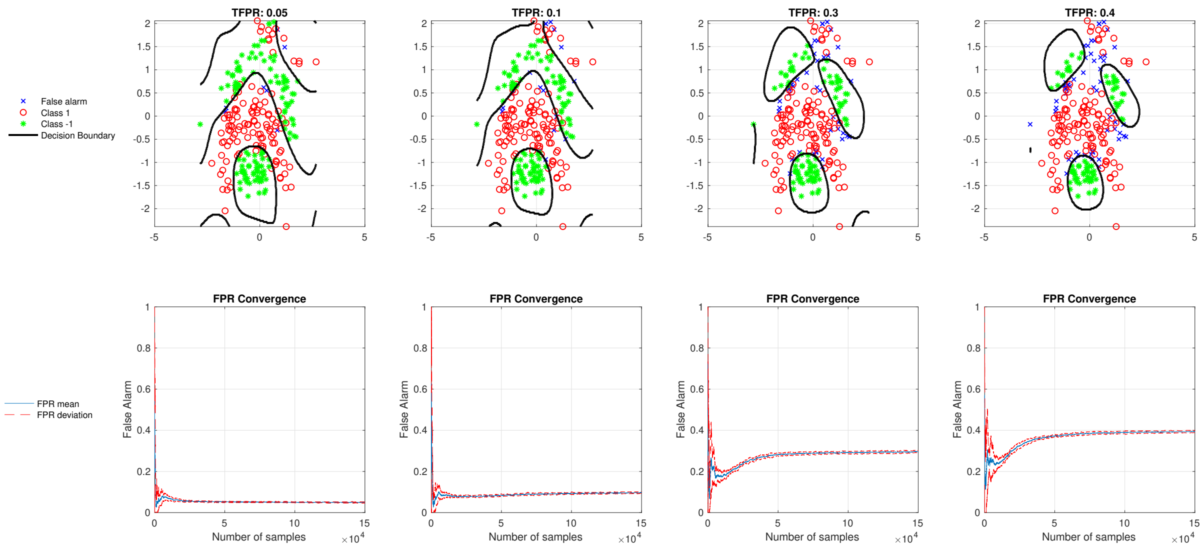

FIGURE 3: Using the visually presentable 2-dimensional Banana dataset, upper graphs show the variation in the decision boundary of the proposed NP-NN as the target false positive rate (TFPR) changes as TFPR $\in \{0.05, 0.1, 0.3, 0.4\}$, and the lower graphs show the mean convergence as well as the standard deviation of the achieved false positive rate (TPR) of the proposed NP-NN over 15 trials of random data permutations with respect to the number of processed instances during training. To better show the convergence, in each trial, length of the training sequence is increased with concatenation resulting in an epoch-by-epoch training of multiple passes. Overall, as indicated by these results, we observe a decent nonlinear modeling as well as a decent false positive rate controllability with the proposed NP-NN.

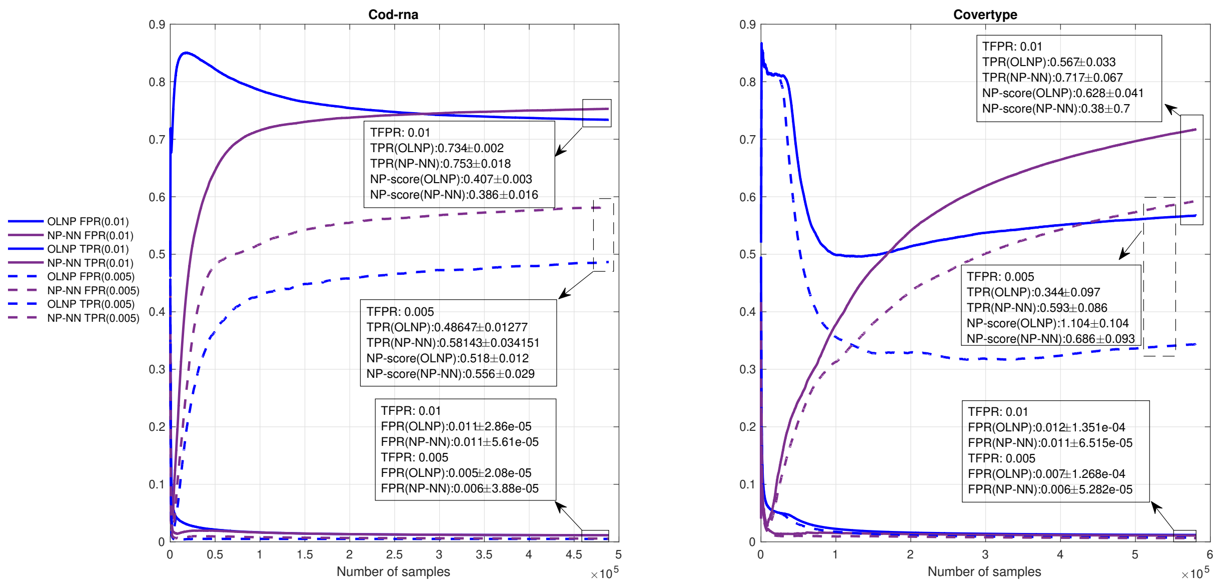

FIGURE 4: We demonstrate the proposed online algorithm NP-NN on two large scale datasets (Cod-rna and Covertype) with relatively small target false positive rates, i.e., TFPR $\in \{0.01, 0.005\}$. The data processing in this case is truly online, and based on only a single pass over the stream without separate training and test phases. Hence, this experiment better demonstres the typical use-case of our algorithm in large scale scenarios. Time accumulated false positive error rate (FPR) and detection rate (TPR) are obtained after averaging over 15 trials of random data permutations. Overall, we observe that the proposed NP-NN and OLNP are both decent and comparable in terms of the false positive rate controllability, whereas the proposed NP-NN strongly outperforms OLNP in terms of both the detection power and NP-score.

NP-NN again strongly outperforms OLNP (except one case, where we observe comparable results). We finally emphasize that the proposed NP-NN achieves this high performance while processing data on the fly in a computation- as well as space-wise extremely efficient manner, in contrast to failing batch techniques in large scale streaming applications due to complexity and failing linear techniques due to insufficient modeling power.

VI. ANALYSIS OF RESULTS AND DISCUSSION

In the previous Section V, we have compared our proposed online algorithm NP-NN with the competing algorithms OLNP and NPROC-SVM based on 10 datasets in terms of NP-scores and AUC. In this section, we next provide a statistical significance analysis of the observed performance differences as well as a detailed complexity analysis with running times, and then discuss our findings.

A. STATISTICAL SIGNIFICANCE ANALYSIS

A manual inspection of the error bars (Table 1) is already indicative about the significance of the performance differences among the compared algorithms, which reinforces our previous discussions/conclusions. In this section, we further quantify the statistical significance for all pairwise performance differences by following the recommendations of the highly comprehensive study in [52], and opt to use the Wilcoxon signed-rank test (WSRT) for this purpose. Note that a paired t-test can also be considered to measure the significance for each dataset separately across the 10 different runs. However, since such runs are based on random permutations of the dataset with k-fold cross validation, the train/test sets inevitably overlap and the independence assumption of the t-test fails to hold, compromising (under estimating) the standard error estimation. Thus, a paired t-test is not recommended after k-fold cross validation. Then, one can use 5×2 cross validation as a remedy, again followed by the paired t-test. However, the paired t-test has its own weaknesses such as the failing Gaussian assumption when the number of data samples is relatively small (≤ 30), the issue of commensurability and the sensitivity to the outliers. Thus, we use the Wilcoxon signed-rank test (WSRT) across all datasets as recommended (as a better alternative to the paired t-test) in [52]. The advantages of WSRT are: testing across datasets (not separately for each) satisfies the independence, ranking is less sensitive to outliers, Gaussian distribution is not assumed, and the issue of commensurability is relieved as it is considered only qualitatively. We refer to [52] for an elegant treatment of this topic of comparing classifiers statistically.

We conduct the WSRT for all pairwise performance differences (in Table 1) of the compared algorithms across all datasets, and present the corresponding T-statistics (of WSRT) in Table 2. We consider the AUC differences as well as the NP-score differences for the target false positive rates $\tau \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. Note that when the T-statistic is less than 3 (or less than 8 but greater than 3), then the difference is highly statistically significant at the level $p < 0.01$ (or still significant but not highly at the level $0.01 < p < 0.05$). We consider that a difference is insignificant if the corresponding T-statistic is greater than 8 since then the significance is $p > 0.05$. The critical values here (3 and 8) are from the two sided t table [52]. We observe that all pairwise performance differences are statistically highly significant, except the three single-starred (Table 2) ones that are only significant (not highly) and except the three double-starred ones that are insignificant. Note that the sign “+" (or “-") in Table 2 is used to indicate that the performance difference is in favor of the algorithm x (or y) in the comparison x vs y.

	NP-NN vs OLNP	NP-NN vs NPROC-SVM	NPROC-SVM vs OLNP
$\tau = 0.05$	$7^{*}(+)$	$18^{**}(-)$	$1(+)$
0.1	$0(+)$	$16.5^{**}(-)$	$0(+)$
0.2	$0(+)$	$2.5(-)$	$0(+)$
0.3	$6.5^{*}(+)$	$0.5(-)$	$0(+)$
0.4	$2.5(+)$	$1.5(-)$	$0(+)$
AUC	$0(+)$	$25^{**}(+)$	$6^{*}(+)$

TABLE 2: We present the T-statistics of Wilcoxon signed-rank significance tests [52] for the pairwise comparisons of the algorithms based on their performance results (across all 10 datasets) in Table 1. The last row is based on the AUC scores whereas the others are based on the NP scores. All performance differences are highly significant with the level $p < 0.01$, except the three single-starred cases where the performance differences are significant (but not highly) since the level is $0.01 < p < 0.05$ as well as the three double-starred cases where the performance differences are considered insignificant since the level is $p > 0.05$. Also, for a comparison x vs y, we use the sign “+" (or “-") if the performance difference is in favor of the algorithm x (y).

This statistical significance analysis firmly supports our performance conclusions in Section V. First, our algorithm strongly (cf. the high performance differences in favor of our online algorithm NP-NN *within* each dataset of Table 1) outperforms OLNP (uniformly in all cases of NP-scores or AUC *across* datasets as seen in Table 2) and the performance difference is statistically either highly significant (in four cases of Table 2) or significant (in the remaining two cases of Table 2). Second, our algorithm performs comparably with NPROC-SVM in terms of AUC as well as in terms of NP-score (two cases of $\tau = 0.01$ and $\tau = 0.05$) since the corresponding performance differences are not significant. In the remaining three cases, NPROC-SVM performs better than NP-NN. On the other hand, we emphasize that our online algorithm NP-NN provides huge computational advantages compared to NPROC-SVM despite their, for instance, comparable AUC performance as well as comparable NP-score performances in two cases. We next provide our complexity analysis.

B. COMPLEXITY ANALYSIS

Recall that our algorithm NP-NN is an online algorithm with the computational complexity $O(NDd)$, where N is the number of data instances, $2D$ is the number of hidden units and d is the data instance dimension. The compared algorithm OLNP is also online with the computational complexity $O(Nd)$. Note that the online algorithms (NP-NN and OLNP) do not have separate training and test phases. Whereas the compared NPROC-SVM is a batch algorithm (not online) with the computational complexity $O(N_{\text{tr}}^2 N_{\text{SV}} d)$ in training and $O(N_{\text{test}} N_{\text{SV}} d)$ in test (cf. [53]), where the number of support vectors $N_{\text{SV}} \leq N_{\text{tr}}$ can be as large as N_{tr} (N_{tr} and N_{test} are the number of instances in training and test sets). Hence, the online algorithms (NP-NN and OLNP) are computationally highly efficient and scalable, however, the batch algorithm NPROC-SVM is prohibitively complex both computationally and space-wise. Note that, also, the online algorithms (NP-NN and OLNP) require negligible space complexity that is only $O(1)$.

We point out that the main complexity difference between the algorithm NPROC-SVM and the online algorithms NP-NN and OLNP stems from the number of data instances (N) whereas the same difference between our proposed algorithm NP-NN and OLNP stems from the number of hidden units ($2D$), i.e., the kernel space expansion cost, in NP-NN. We next observe these differences in terms of the actual running times, and devise an experiment for this purpose that is controlled with respect to the variables N and D. This experiment is based on three datasets with the properties $(n_-, n_+, d) \in \{(5000, 5000, 5), (50000, 50000, 5), (150000, 150000, 5)\}$. Here, D is set as $D = 5 \times d = 25$, $N = n_- + n_+$ and n_+ (n_-) is the number of class 1 (class -1) labeled instances, and the datasets are randomly generated as bimodal Gaussian distributed with random means and covariances around 0. We run the algorithms 10 times on random permutations of such datasets and report in Table 3 the mean running times in seconds (s) with the corresponding standard deviations. Note that we report separately for training and test phases in the case of NPROC-SVM. TFPR is set as $\tau = 0.1$. Our online algorithm NP-NN and the compared algorithm OLNP are implemented in MATLAB, and the compared algorithm NPROC-SVM is implemented in Python with the original code provided in [28]. Computations are conducted on a computer containing 2.2 GHz Quad-Core Intel i7. Our mean running time observations (Table 3) are in line with the complexity analysis above. The online algorithms NP-NN and OLNP both run fast where NP-NN takes slightly more time due to the cost of the hidden layer of our SLFN with $2D$ hidden units. Note that, here, we use a straightforward unoptimized MATLAB implementation (for both NP-NN and OLNP), and thus the kernel space expansion cost in the hidden layer of our algorithm NP-NN can be significantly reduced with a more CPU-friendly and optimized implementation in a low level language. Therefore, our algorithm NP-NN can significantly speed up and the running time difference between NP-NN and OLNP can further decrease. On the other hand, the algorithm NPROC-SVM quickly becomes impractical and prohibitively complex, as the required running time polynomially increases as $\sim 2000\times$ when the number of data instances increases only $30\times$. Hence, NPROC-SVM is not scalable.

C. DISCUSSION

As extensively demonstrated in Section V, Section VI-A and Section VI-B, in summary, our performance results (in terms of both the AUC and NP-score) and significance as well as complexity analyses statistically significantly and firmly conclude that our online algorithm NP-NN either outperforms the state-of-the-art (e.g. the compared algorithm OLNP in our experiments) at a comparable complexity, or performs comparably (to the compared algorithm NPROC-SVM in our experiments) while providing huge computational and space advantages. In the following, we explain the underlying reason for the superiority of our proposed algorithm NP-NN over the state-of-the-art (i.e. the competing algorithms OLNP and NPROC-SVM).

The kernel inspired SLFN of the proposed NP-NN has two benefits: expedited powerful nonlinear modeling and scalability. Namely, first, it enables an excellent network initialization as random Fourier features (RFFs) are already sufficiently powerful to learn complex nonlinear decision boundaries even when kept untrained. This speeds up and enhances the learning of complex nonlinearities by relieving the burden of network initialization. Second, the hidden layer is compactified thanks to the exponential rate of improvement in approximating the high dimensional kernel space due to Hoeffding's inequality [12]. As a result, the number of hidden nodes, parameter complexity and the computational complexity of forward-backward network evaluations reduce, and therefore the scalability substantially improves while also mitigating overfitting. Moreover, thanks to the learning of the hidden layer, the randomly initialized Fourier features are continuously improved during SGD steps for even further compactification and better nonlinear modeling. We point out that the competing algorithm NPROC-SVM is also powerfully nonlinear but it does not exploit Fourier features and explicit kernel space expansion for scalability and computational as well as space-wise efficiency. Whereas the competing algorithm OLNP is also online and efficient but it is not designed to model nonlinearities. Hence, our online NP classifier is powerfully nonlinear (which clearly explains the superiority over the competing algorithm OLNP) and computationally highly efficient with $O(N)$ processing and negligible $O(1)$ space complexity (which clearly explains the superiority over the competing algorithm NPROC-SVM), where N is the number of data instances.

While the learning (i.e. optimizing) of Fourier features significantly strengthen the capability of nonlinear modeling, our method can find features that are only locally optimal due to the nonconvexity of the optimization we studied. This appears as a limitation of our technique. On the other hand, those learned locally optimal Fourier features do not necessarily define a proper kernel which satisfies the Bochner's theorem that we use to initialize the network with the radial basis function kernel. This is perhaps not a limitation but

Dataset (n_-, n_+, d)	OLNP	NPROC-SVM (train)	NPROC-SVM (test)	NP-NN
Dataset 1 (5000, 5000, 5)	0.197 ± 0.033 s	0.931 ± 0.238 s	0.014 ± 0.001 s	0.547 ± 0.048 s
Dataset 2 (50000, 50000, 5)	1.444 ± 0.028 s	240.153 ± 6.325 s	0.170 ± 0.008 s	3.709 ± 0.091 s
Dataset 3 (150000, 150000, 5)	4.265 ± 0.087 s	1899.047 ± 27.088 s	0.454 ± 0.009 s	10.877 ± 0.134 s

TABLE 3: Mean running times are reported along with standard deviations for all algorithms across 10 different trials. Datasets are generated randomly as bimodal Gaussian distributed with random mean and covariances around 0. Training and test observations are provided separately for the algorithm NPROC-SVM since it is not online.

certainly an indicator of that our method is not appropriate for learning a nonisotropic rbf kernel. Then, if desired, one can attempt to enforce positive definiteness of the weights in the hidden layer of our SLFN as a constraint to the network optimization, which would be a point of improvement. Similarly, it is true that Fourier features is a powerful means for nonlinear classification but it also makes it prone to overfitting. For this reason, one has to carefully tune the parameters kernel bandwidth and the hidden layer size for which we successfully use extensive cross validations. However, in the case of a very large dimensionality (e.g. images), overfitting may still appear as a limitation. For this issue, we suggest using a strong regularization while incorporating the introduced network as the fully connected layers of a deep architecture. Then, as a remedy, the earlier layers can be designed to extract features and reduce dimensionality for our network following in the deeper layers.

VII. CONCLUSION

We considered binary classification with particular regard to i) a user defined constraint on the type I error (false positive) rate that requires false positive rate (FPR) controllability, ii) nonlinear modeling of complex decision boundaries, and iii) computational scalability to voluminous data with online processing. To this end, we propose a computationally highly efficient online algorithm to determine the pair of asymmetrical type I and type II error costs to satisfy the FPR constraint and solve the resulting cost sensitive nonlinear classification problem in the non-convex sequential optimization framework of neural networks. The proposed algorithm is essentially a Neyman-Pearson classifier, which is based on a single hidden layer feed forward neural network (SLFN) with decent nonlinear classification capability thanks to its kernel inspired hidden layer. The SLFN that we use for Neyman-Pearson classification is compact in principle for two reasons. First, the hidden layer exploits -during initialization- the exponential convergence of the inner products of random Fourier features to the true kernel value with sinusoidal activation. Second, learning of the hidden layer parameters, i.e., Fourier features, help to improve the randomly initialized Fourier features. Consequently, the required number of hidden nodes, i.e., the required number of network parameters and Fourier features, can be chosen relatively small. This reduces the parameter complexity and thus mitigates overfitting while significantly reducing the computational as well as space complexity. Then the output layer follows as a perceptron with identity activation. We sequentially learn the SLFN parameters through stochastic gradient descent based on a Lagrangian non-convex optimization to goal of Neyman-Pearson classification. This procedure minimizes the type II error rate about the user specified type I error rate, while producing classification decisions in the run time. Overall, the proposed algorithm is truly online and appropriate for contemporary fast streaming data applications with real time processing and FPR controllability requirements. Our online algorithm was experimentally observed to either outperform (in terms of the detection power and false positive rate controllability) the state-of-the-art competing techniques with a comparable processing and space complexity, or perform comparably with the batch processing techniques, i.e., not online, that are -however- computationally prohibitively complex and not scalable.

REFERENCES

[1] F. Feng, K. Li, J. Shen, Q. Zhou, and X. Yang. Using cost-sensitive learning and feature selection algorithms to improve the performance of imbalanced classification. IEEE Access, 8:69979–69996, 2020.
[2] X. Xi, Z. Yu, Z. Zhan, Y. Yin, and C. Tian. Multi-task cost-sensitive-convolutional neural network for car detection. IEEE Access, 7:98061–98068, 2019.
[3] H. Masnadi-Shirazi and N. Vasconcelos. Cost-sensitive boosting. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(2):294–309, 2011.
[4] Mark A Davenport, Richard G Baraniuk, and Clayton D Scott. Tuning support vector machines for minimax and Neyman-Pearson classification. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(10):1888–1898, 2010.
[5] J. Lu, V. E. Liong, and J. Zhou. Cost-sensitive local binary feature learning for facial age estimation. IEEE Transactions on Image Processing, 24(12):5356–5368, 2015.
[6] J. Wan and F. Zhu. Cost-sensitive canonical correlation analysis for semi-supervised multi-view learning. IEEE Signal Processing Letters, 27:1330–1334, 2020.
[7] M. Liu, L. Miao, and D. Zhang. Two-stage cost-sensitive learning for software defect prediction. IEEE Transactions on Reliability, 63(2):676–686, 2014.
[8] Venkatesh Saligrama and Zhu Chen. Video anomaly detection based on local statistical aggregates. IEEE International Conference on Computer Vision and Pattern Recognition, pages 2112–2119, 2012.
[9] Huseyin Ozkan, Ozgun Soner Pelvan, and Suleyman S Kozat. Data imputation through the identification of local anomalies. IEEE Transactions on Neural Networks and Learning Systems, 26(10):2381–2395, 2015.
[10] Liangxiao Jiang, Chaoqun Li, and Shasha Wang. Cost-sensitive bayesian network classifiers. Pattern Recognition Letters, 45:211–216, 2014.
[11] Xin Tong, Yang Feng, and Anqi Zhao. A survey on Neyman-Pearson classification and suggestions for future research. Wiley Interdisciplinary Reviews: Computational Statistics, 8(2):64–81, 2016.
[12] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. Advances in Neural Information Processing Systems, pages 1177–1184, 2008.
[13] Clayton D Scott and Robert D Nowak. Learning minimum volume sets. Journal of Machine Learning Research, 7:665–704, 2006.
[14] Alfred O Hero. Geometric entropy minimization (gem) for anomaly detection and localization. Advances in Neural Information Processing Systems, pages 585–592, 2007.
[15] Manqi Zhao and Venkatesh Saligrama. Anomaly detection with score functions based on nearest neighbor graphs. Advances in Neural Information Processing Systems, pages 2250–2258, 2009.
[16] Venkatesh Saligrama and Manqi Zhao. Local anomaly detection. Artificial Intelligence and Statistics, pages 969–983, 2012.

[17] Yuting Chen, Jing Qian, and Venkatesh Saligrama. A new one-class svm for anomaly detection. IEEE International Conference on Acoustics, Speech and Signal Processing, pages 3567–3571, 2013.

[18] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. Neural Computation, 13(7):1443–1471, 2001.

[19] Régis Vert and Jean-Philippe Vert. Consistency and convergence rates of one-class svms and related algorithms. Journal of Machine Learning Research, 7:817–854, 2006.

[20] Xuedan Miao, Ying Liu, Haiquan Zhao, and Chunguang Li. Distributed online one-class support vector machine for anomaly detection over networks. IEEE Transactions on Cybernetics, (99):1–14, 2018.

[21] Yang Zhang, Nirvana Meratnia, and Paul Havinga. Adaptive and online one-class support vector machine-based outlier detection techniques for wireless sensor networks. IEEE International Conference on Advanced Information Networking and Applications Workshops, pages 990–995, 2009.

[22] Roy L Streit. A neural network for optimum Neyman-Pearson classification. IEEE International Joint Conference on Neural Networks, pages 685–690, 1990.

[23] Xin Tong. A plug-in approach to Neyman-Pearson classification. Journal of Machine Learning Research, 14:3011–3040, 2013.

[24] Clayton Scott and Robert Nowak. A Neyman-Pearson approach to statistical learning. IEEE Transactions on Information Theory, 51(11):3806–3819, 2005.

[25] María-Pilar Jarabo-Amores, David De la Mata-Moya, Roberto Gil-Pita, and Manuel Rosa-Zurera. Radar detection with the Neyman–Pearson criterion using supervised-learning-machines trained with the cross-entropy error. EURASIP Journal on Advances in Signal Processing, 2013(1):44, 2013.

[26] Shuchen Kong, Weiwei Shen, Yingbin Zheng, Ao Zhang, Jian Pu, and Jun Wang. False positive rate control for positive unlabeled learning. Neurocomputing, 367:13–19, 2019.

[27] David R Cox. The regression analysis of binary sequences. Journal of the Royal Statistical Society: Series B (Methodological), 20(2):215–232, 1958.

[28] Xin Tong, Yang Feng, and Jingyi Jessica Li. Neyman-Pearson classification algorithms and np receiver operating characteristics. Science Advances, 4(2):eaao1659, 2018.

[29] Cewu Lu, Jianping Shi, and Jiaya Jia. Abnormal event detection at 150 fps in matlab. IEEE International Conference on Computer Vision, pages 2720–2727, 2013.

[30] Huseyin Ozkan, Fatih Ozkan, and Suleyman S Kozat. Online anomaly detection under markov statistics with controllable type-i error. IEEE Transactions on Signal Processing, 64(6):1435–1445, 2015.

[31] Ao Zhang, Nan Li, Jian Pu, Jun Wang, Junchi Yan, and Hongyuan Zha. Tau-fpl: Tolerance-constrained learning in linear time. Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[32] Gilles Gasso, Aristidis Pappaioannou, Marina Spivak, and Léon Bottou. Batch and online learning algorithms for nonconvex Neyman-Pearson classification. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3):1–19, 2011.

[33] Hirofumi Uzawa. Iterative methods for concave programming. Studies in Linear and Nonlinear Programming, 6:154–165, 1958.

[34] Jing Lu, Steven CH Hoi, Jialei Wang, Peilin Zhao, and Zhi-Yong Liu. Large scale online kernel learning. Journal of Machine Learning Research, 17(1):1613–1655, 2016.

[35] Fatih Porikli and Huseyin Ozkan. Data driven frequency mapping for computationally scalable object detection. IEEE International Conference on Advanced Video and Signal Based Surveillance, pages 30–35, 2011.

[36] Zhuang Wang, Koby Crammer, and Slobodan Vucetic. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. Journal of Machine Learning Research, 13(Oct):3103–3131, 2012.

[37] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. Advances in Neural Information Processing Systems, pages 342–350, 2009.

[38] Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. Advances in Neural Information Processing Systems, pages 2627–2635, 2014.

[39] Kurt Cutajar, Edwin V Bonilla, Pietro Michiardi, and Maurizio Filippone. Random feature expansions for deep gaussian processes. International Conference on Machine Learning, pages 884–893, 2017.

[40] Siamak Mehrkanoon and Johan AK Suykens. Deep hybrid neural-kernel networks using random fourier features. Neurocomputing, 298:46–54, 2018.

[41] David Casasent and Xuewen Chen. Radial basis function neural networks for nonlinear fisher discrimination and Neyman–Pearson classification. Neural Networks, 16(5-6):529–535, 2003.

[42] Jiaxuan Xie, Fanghui Liu, Kaijie Wang, and Xiaolin Huang. Deep kernel learning via random fourier features, 2019.

[43] Basarbatu Can, Mine Kerpicci, and Huseyin Ozkan. Online kernel-based nonlinear neyman-pearson classification.

[44] H Vincent Poor. An introduction to signal detection and estimation. Springer Science & Business Media, 2013.

[45] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review, 65(6):386, 1958.

[46] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. Advances in Neural Information Processing Systems, pages 476–484, 2012.

[47] Clayton Scott. Performance measures for Neyman–Pearson classification. IEEE Transactions on Information Theory, 53(8):2852–2863, 2007.

[48] Marthinus Du Plessis, Gang Niu, and Masashi Sugiyama. Convex formulation for learning from positive and unlabeled data. International Conference on Machine Learning, pages 1386–1394, 2015.

[49] Corinna Cortes and Vladimir Vapnik. Support-vector networks. Machine learning, 20(3):273–297, 1995.

[50] Dheeru Dua and Casey Graff. Uci machine learning repository (2017). URL http://archive. ics. uci. edu/ml, 37, 2017.

[51] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2(3):1–27, 2011.

[52] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. Journal of Machine learning research, 7(Jan):1–30, 2006.

[53] Léon Bottou and Chih-Jen Lin. Support vector machine solvers. Large scale kernel machines, 3(1):301–320, 2007.